ERCEMAPI 2010

Escola Regional de Computação Ceará - Maranhão - Piauí

Livro Texto dos Minicursos

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Iális Cavalcante de Paula Júnior
Andréa Carneiro Linhares

ERCEMAPI 2010

Escola Regional de Computação Ceará - Maranhão - Piauí

IV Escola Regional de Computação
Ceará - Maranhão - Piauí

11 e 12 de novembro de 2010
Sobral - CE

Livro Texto dos Minicursos

Editora:
Sociedade Brasileira de Computação - SBC

Organizadores:
Iális Cavalcante de Paula Júnior
Andréa Carneiro Linhares

Realização:
Universidade Federal do Ceará - UFC

Promoção:
Sociedade Brasileira de Computação - SBC

Escola Regional de Computação Ceará, Maranhão, Piauí (2010: Sobral, CE)

Escola Regional de Computação Ceará, Maranhão, Piauí: livro texto dos minicursos, 11 e 12 de novembro de 2010. [livro eletrônico] / organizadores, Iális Cavalcante de Paula Júnior, Andréa Carneiro Linhares. – Sobral: Sociedade Brasileira de Computação – SBC, 2010.

142p. : il. ; livro eletrônico.

Realização: Universidade Federal do Ceará - UFC.

Promoção: Sociedade Brasileira de Computação – SBC.

ISBN 978-85-7669-251-5

1. Computação; Ciência da Computação; I. Paula Júnior, Iális Cavalcante de. II. Linhares, Andréa Carneiro. III. Título.

CDD 004.1

Prefácio

A Escola Regional de Computação Ceará - Maranhão - Piauí (ERCEMAPI) é um evento anual, realizado desde 2007, cujo objetivo é promover a disseminação de conhecimentos e experiências agregando isso às instituições de ensino superior dos estados do Ceará, Maranhão e Piauí. Apesar de sua ênfase regional, desde sua primeira edição, contou com a participação de convidados e palestrantes de diversas localidades do país. Esta escola regional é promovida pela Sociedade Brasileira de Computação (SBC).

Esta quarta edição da ERCEMAPI foi a primeira a publicar, em forma de livro, os textos de apoio dos minicursos apresentados dentro da programação do evento. Tópicos recentes de pesquisa e temas atuais de interesse das comunidades dos estados do Ceará, Maranhão e Piauí são abordados de forma prática e didática.

Foram recebidas vinte e uma propostas de minicursos, todas de muito bom nível, o que demandou um minucioso trabalho de seleção das mesmas. A edição de 2010 oferece, sete minicursos aos participantes da ERCEMAPI com temas nas áreas de: aplicações para TV digital, algoritmos de visão computacional com OpenCl e CUDA, desenvolvimento *web*, testes de aceitação, tradução e interpretação de linguagens de domínio específico, empreendedorismo e computação em nuvens.

Este volume se inicia com uma discussão sobre o desenvolvimento de aplicações declarativas para TV digital, na linguagem NCL, mescladas com código imperativo em NCLua. Primeiramente, são apresentados conceitos básicos de NCL e Lua e, em seguida, abordados os paradigmas de autoria declarativa e imperativa, entremeados por exemplos de crescente dificuldade descritos ao longo do curso.

O Capítulo 2 leva ao estudo da otimização de algoritmos de visão computacional. Por quase sempre abordarem complexas estruturas de dados, o processamento desses algoritmos pode requerer um auxílio para a redução de seu custo computacional. A integração de OpenCL, que instrui execuções em múltiplas instâncias simultaneamente, e CUDA, levando o processamento gráfico à unidade de *hardware* específica, contribuem para a execução mais precisa dos algoritmos aqui apresentados.

O Capítulo 3, aborda-se a ferramenta livre Joomla para criação de *websites*. Seu uso é muito difundido na *internet* e diversas organizações vêm adotando-a principalmente em seus veículos de divulgação e comunicação. Uma característica que amplia sua adoção é que não se exige conhecimento em programação por parte do usuário de Joomla, mesmo produzindo páginas *web* na linguagem PHP.

O Capítulo 4 volta-se para desenvolvimento de *software* dirigido por testes de aceitação. Apresenta-se aqui uma prática de programação muito mais relacionada a definir as expectativas quanto ao comportamento do código de um software, e fazer com que isso guie a implementação que está sob teste.

O Capítulo 5 apresenta uma visão geral das linguagens de domínio específico (do inglês: *Domains Specific Languages* - DSLs) e a ferramenta ANTLR (*ANother Tool for Language Recognition*). Esta é direcionada para a construção de tradutores e interpretadores para DSLs, ou seja, uma ferramenta para implementação de linguagens de programação, que traz um desenvolvimento de *software* com representação mais natural que linguagens já conhecidas.

O Capítulo 6 engloba conceitos de empreendedorismo em Tecnologia da Informação (TI), como e-empreendedor e negócios digitais. Além disso, inclui-se nessa discussão o papel do profisisonal de TI como gestor em uma organização. É interessante e motivante notar na Computação exemplos e discussões sobre os infindáveis campos de atuação, recursos gratuitos de apoio na *internet*, maior diversificação e capacidade de abrangência de um mercado global.

O Capítulo 7 encerra este livro trazendo à tona uma das fortes tendências assumidas na *internet*: computação em nuvens. A partir da apresentação de definições e conceitos a cerca de *cloud computing*, é apresentada a montagem de uma infraestrutura privada para a disponibilização de serviços em uma rede de computadores específica.

O material aqui compilado é resultado de um grande esforço por parte dos autores e será de utilidade não apenas aos participantes dos minicursos, mas também a todos aqueles que desejem aprofundar seus conhecimentos nos diversos assuntos tratados.

Sobral-CE, novembro de 2010.

Iális Cavalcante de Paula Júnior
Andréa Carneiro Linhares

Sumário

Desenvolvendo aplicações interativas para TV Digital utilizando NCL e Lua 01
Fábio de Jesus Lima Gomes (IFPI), Danilo Araújo Portela (IFPI), Laécio Freitas Chaves (IFPI), Marcelo de Miranda Clementino (IFPI)

Utilização do CUDA e OpenCL em Algoritmos de Visão Computacional 21
Thiago S. Santos (UFPI), Kalyf A. B. Lima (UFPI), Kelson R. T. Aires (UFPI)

Joomla 1.6 40
Alexandre Marques Albano da Silveira (UFC - *Campus* de Sobral), Andréa Carneiro Linhares (UFC - *Campus* de Sobral), Clemilson Costa Santos (UFC - *Campus* de Sobral)

Desenvolvimento Dirigido por Testes de Aceitação 59
Pedro de Alcântara Santos Neto (UFPI), Antonio Gabriel Di Atlanta Valente (UFPI), Ítalo Linhares de Araújo (UFPI)

Tradução e interpretação de linguagens de domínio específico com a ferramenta ANTLR 83
Cairo Antonio Oliveira Rocha (UFPI), Raimundo Santos Moura (UFPI)

Empreendedorismo e Gestão em TI 101
Paulo César Coutinho dos Santos (UESPI)

Cloud Computing: Montagem uma Infraestrutura Privada 121
Julio Cesar Damasceno (UFPE), Fernando Antonio Aires Lins (UFPE), Márcio Almeida de Macêdo (UFPE)

Comitê de Programa

Coordenação Geral:
Iális Cavalcante de Paula Júnior (UFC - *Campus* de Sobral)

Coordenação do Comitê de Programa:
Andréa Carneiro Linhares (UFC - *Campus* de Sobral)

Comitê de Programa:
André Castelo Branco Soares (UFPI)
Carlos Alexandre Rolim Fernandes (UFC - *Campus* de Sobral)
Clemilson Costa dos Santos (UFC - *Campus* de Sobral)
Daniel Benevides da Costa (UFC - *Campus* de Sobral)
Davi Romero de Vasconcelos (UFC - *Campus* de Quixadá)
Emanuel Ferreira Coutinho (UFC)
Enyo José Tavares Gonçalves (UFC - *Campus* de Quixadá)
Erick Aguiar Donato (UFC - *Campus* de Sobral)
Fabiana Gomes Marinho (UFC)
Fernando Rodrigues de Almeida Júnior (UFC - *Campus* de Sobral)
Francisco Heron Carvalho Junior (UFC)
Francisco Nivando Bezerra (IFCE - Maracanaú)
Gabriel Antoine Louis Paillard (UFC)
George Allan Menezes Gomes (UFC)
Gerardo Valdisio Rodrigues Viana (UECE)
Gilzamir Ferreira Gomes (UVA - Sobral)
Gisele Azevedo de Araújo (UFC - *Campus* de Sobral)
Glaudiney Moreira Mendonça Júnior (UFC)
Iális Cavalcante de Paula Júnior (UFC - *Campus* de Sobral)
José Alex Pontes Martins (UVA)
José Riverson Araújo Cysne Rios (UFC)
Lincoln Souza Rocha (UFC - *Campus* de Quixadá)
Luis Henrique Pequeno Almeida (UFC)
Marcelino Cavalcante Pequeno (UFC)
Mardson Freitas de Amorim (UFC - *Campus* de Sobral)
Miguel Franklin de Castro (UFC)
Pedro de Alcântara dos Santos Neto (UFPI)
Rafael Castro de Andrade (UFC)
Raimir Holanda Filho (Unifor)
Rodrigo de Melo Souza Veras (UFPI)
Tarcisio Ferreira Maciel (UFC)
Wellington Wagner Ferreira Sarmento (UFC)
Wendley Souza da Silva (UFC - *Campus* de Sobral)

ERCEMAPI 2010

Escola Regional de Computação Ceará - Maranhão - Piauí

Livro Texto dos Minicursos

Organizadores:

Ialis Cavalcante de Paula Júnior

Andréa Carneiro Linhares

Capítulo 1

Desenvolvendo aplicações interativas para TV Digital utilizando NCL e Lua

Fábio de Jesus Lima Gomes, Danilo Araújo Portela, Laécio Freitas Chaves, Marcelo de Miranda Clementino.

Abstract

This course aims to disseminate and discuss concepts involved in developing applications for Interactive Digital TV using the Ginga-NCL. Besides presenting the main aspects of Ginga and its subsystem Ginga-NCL, this chapter presents the key elements and concepts of the languages NCL and Lua through examples and the integration between both languages in the same application.

Resumo

Este minicurso visa disseminar e abordar conceitos envolvidos no desenvolvimento de aplicações para TV Digital Interativa utilizando o ambiente Ginga-NCL. Além de apresentar os principais aspectos do Ginga e de seu subsistema Ginga-NCL, este capítulo também apresenta os principais elementos e conceitos da Linguagem NCL e Lua através de exemplos práticos e também a integração entre elas numa mesma aplicação.

1.1. Introdução

O Middleware Ginga, desenvolvido pela PUC-Rio e pela UFPB, é o padrão do Sistema Brasileiro de Digital como camada de aplicações [ABNT 2007]. O Ginga é uma camada de software que dá suporte à execução de aplicações interativas nos conversores digitais, esses últimos instalados nas casas dos telespectadores. A arquitetura do Ginga é composta tanto por um ambiente declarativo quanto por um imperativo, o que permite a alternativa mais adequada quando se está desenvolvendo aplicações para TV Digital. É

possível, inclusive, fazer uso de uma ponte para desenvolver aplicações declarativas com partes imperativas e vice-versa.

O ambiente imperativo do Ginga, também chamado de máquina de execução, é a parte responsável pelo suporte a aplicações desenvolvidas usando a linguagem de programação Java. O ambiente declarativo, conteúdo abordado neste capítulo, também chamado de máquina de apresentação, interpreta aplicações desenvolvidas em Neste Context Language (NCL).

Linguagens declarativas são, em geral, criadas para um determinado foco ou domínio e permitem a especificação de aplicações em um nível mais alto de abstração quando comparadas a linguagens imperativas. Linguagens declarativas são voltadas para especificar a intenção final e não uma sequência passo a passo para resolver um determinado problema, como e o caso de linguagens imperativas.

Na implementação de um sistema de TVDI em um país deve-se levar em consideração a realidade socioeconômica e cultural do mesmo. Como veremos a seguir as diferenças entre os sistemas existentes se dá devido as diferentes necessidades de cada país, assim como devido às tecnologias disponíveis no momento em que foram criados.

1.2. SBTVD e Ginga

O sistema de TV Digital Interativa Brasileiro, chamado de SBTVD-T (Sistema Brasileiro de Televisão Digital – Terrestre) foi implantado no final do ano de 2007 e possui o Ginga como middleware. O Ginga possui dois subsistemas responsáveis pela implementação das aplicações interativas, são eles o Ginga-NCL e o Ginga-J. O Ginga-NCL utiliza uma linguagem declarativa baseada em XML chamada NCL (Nested Context Language) para especificação de aspectos de interatividade, sincronismo espaço-temporal entre objetos de mídia, adaptabilidade e suporte a múltiplos dispositivos, já o Ginga-J utiliza a linguagem Java ara a construção das aplicações.

Em um ambiente onde existem vários tipos, marcas e modelos diferentes de hardwares responsáveis para a decodificação do sinal digital (set-top-box), existe o problema da incompatibilidade entre os mesmos, o que poderia ser um problema em se falando de construção de aplicações que serão distribuídas indistintamente para todos os receptores. Como garantir que uma aplicação que execute bem em determinado modelo de set-top-box, funcione em outro?

Para responder essa pergunta e solucionar tal problema é que surge o conceito de *middleware*. O *middleware* é uma camada de software que fica entre o sistema operacional e as aplicações. Ele é o responsável por prover um conjunto de funções comum, padronizando o desenvolvimento de programas interativos. Uma vez que todas as aplicações sejam construídas utilizando essas funções comuns, o *middleware* fica encarregado de se comunicar com os hardwares diferentes, agindo como um interprete entre as aplicações, escritas seguindo um padrão comum, e os sistemas operacionais e hardwares diferentes.

Como dito anteriormente, o Ginga é o middleware do SBTVD. Definido basicamente por pesquisadores das universidades UFPB e PUC-RJ, com participação de

pesquisadores da UFRN e UERN, ele está dividido em três partes: O Ginga-NCL, Ginga-J e Ginga Common Core, como exemplificado na arquitetura da Figura 1.

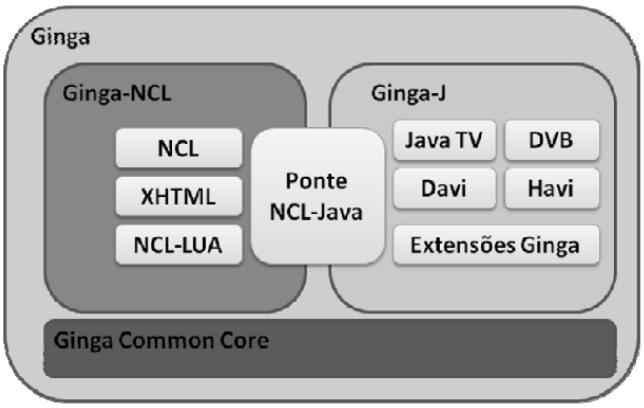


Figura 1.1 – Arquitetura do Ginga

Como podemos ver na Figura 1 as aplicações escritas usando o Ginga-NCL pode utilizar tags NCL, tags HTML e scripts escritos na linguagem Lua. Já a parte Ginga-J permite escrever programas usando a linguagem Java. Ainda é permitido a criação de programas híbridos, no sentido de serem implementados em NCL e Java. Pois o Java pode ser considerado como uma mídia do NCL, e o Java pode manipular o NCL. A seguir entraremos em maiores detalhes sobre o Ginga-NCL e o Ginga-J.

1.3. Aplicações para TVDI

As aplicações de TV Digital Interativa podem estar ou não semanticamente associadas ao conteúdo do áudio ou vídeo principal. Adicionalmente, elas podem definir ou não relações de sincronismo entre objetos de mídia que a compõem, entre eles o conteúdo principal (vídeo e áudio).

Uma aplicação de correio eletrônico, por exemplo, está sempre disponível e não possui relação semântica com o conteúdo televisivo exibido. Por outro lado, uma aplicação que calcula o "nível de estresse" em um programa de saúde, por exemplo, poderia estar disponível durante toda sua exibição. É conveniente notar que, mesmo não havendo relações de sincronismo dessas aplicações com o áudio e vídeo principal, tais aplicações podem ser compostas de objetos que mantêm relações de sincronismo entre si Ainda um terceiro tipo de aplicação é composto por aplicações em que existe não só relação seus objetos de mídia e o áudio e vídeo principal do programa televisivo, mas também um a relação de sincronismo. Esse é exatamente o caso dos chamados programas não lineares.

O termo não linear vem em contraposição à forma sequencial – linear – que caracteriza os programas para a TV analógica. Nesses últimos, existe um e apenas um caminho, sequencial de exibição. Ao contrário, os programas não lineares são compostos de múltiplas cadeias de exibição, da região onde o telespectador está inserido etc. O exemplo mais simples de um programa não linear é aquele onde, em um dado instante de exibição, o usuário telespectador pode escolher entre formas alternativas de sua continuação. Note assim que o programa deixa de poder ser

representado por uma linha de tempo e passa a ter um fluxo de exibição que pode ser especificado por um grafo.

Na grande maioria dos casos, a linguagem declarativa tende a ser a preferencial no desenvolvimento dos programas não lineares. Mais ainda, como em programas não lineares o sincronismo intermídia sem a interação do usuário deve ser tão ou mais importante que a interatividade, o sincronismo de mídias em sua forma mais ampla, e não a interatividade deve ser o foco das linguagens declarativas, como é o caso da linguagem NCL, proposta para o SBTVD.

Outro aspecto importante para as aplicações e a adaptabilidade. Ao agregar capacidade computacional à TV, torna-se possível fazer com que o terminal de acesso adapte o conteúdo de acordo com informações de contexto referentes às preferências ou localização do usuário, ou ainda à disponibilidade atual do terminal de acesso (capacidade de processamento, memória disponível etc.). A adaptação pode não envolver apenas o conteúdo de cada objeto de mídia individualmente, mas também a própria forma de apresentação. NCL provê suporte a ambos os tipos de adaptação.

O suporte a múltiplos dispositivos de exibição é também uma característica importante no suporte à interatividade em um sistema de TV digital. Através de múltiplos dispositivos de exibição será possível, por exemplo, que a interação de um usuário com o programa de TV traga novos objetos a serem exibidos em seu dispositivo particular de interação, sem que apareçam na tela da TV, não atrapalhando, assim, uma audiência coletiva.

1.4. NCL (Nested Context Language)

A linguagem NCL (*Nested Context Language*) é uma linguagem declarativa para autoria de documentos hipermídia baseados no modelo conceitual NCM (*Nested Context Model*) e foi desenvolvida utilizando uma estrutura modular, seguindo os princípios adotados pelo W3C.

Base do Ginga-NCL, a NCL define uma separação bem demarcada entre o conteúdo e a estrutura de um documento, provendo um controle não invasivo da ligação entre o conteúdo e sua apresentação e leiaute. Baseada no NCM (Nested Context Model), a NCL se utiliza dos conceitos usuais de nós e elos para descrever documentos hipermídia. Os nós (*nodes*) representam fragmentos de informações, que podem ser de mídia ou de contexto, e elos (links) são usados para definição de relacionamentos entre os nós [Soares e Rodrigues 2005].

A figura 1.2 representa como esses nós podem ser aninhados, formando um conjunto de nós e elos, permitindo tornar a estrutura de um documento mais enxuta e organizada.

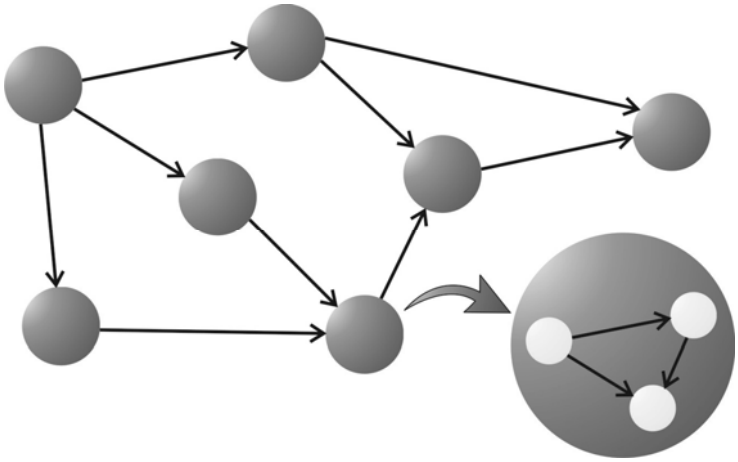


Figura 1.2. Representação do modelo NCM

Um documento NCL é escrito em XML, e a estrutura de um documento NCL deve possuir um cabeçalho de arquivo, um cabeçalho do programa e o encerramento do documento. Então para construir um documento hipermídia além da estrutura acima, é necessário definir o que se quer tocar, onde, como e quando tocar. A listagem 1.1 mostra a estrutura básica de um documento NCL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
      <head>
            <!-- Onde? Aqui definimos regiões -->
            <!-- Como? Aqui definimos descritores -->
      </head>
      <body>
            <!-- O que? Aqui definimos nós de mídia -->
            <!-- Quando? Aqui definimos os elos -->
      </body>
</ncl>
```

Listagem 1.1. Estrutura de um documento NCL

Segundo Sant'Anna, Soares, e Cerqueira (2008), a estrutura básica de um documento NCL define um elemento raiz, chamado de <ncl>, e seus elementos filho, o elemento <head> e <body>, seguindo a terminologia adotada por outros padrões W3C.

1.6.1. Definindo Regiões

Uma região nada mais é que uma área na tela, que pode ser uma TV, celular, ou qualquer outro equipamento compatível com a tecnologia, onde será exibido o conteúdo de um nó de mídia. Uma região deve ser definida no cabeçalho do programa (<head>), dentro da seção de base de regiões (<regionBase>). Abaixo segue um exemplo de como definimos regiões em um documento NCL (listagem 1.2).

```
<head>
      <regionBase>
            <region id="rgTela" height="100%" width="100%" >
                  <region id="rgVideo1" height="50%" width="50%"
                  left="10%" top="30%" />
            </region>
      </regionBase>
</head>
```

Listagem 1.2. Definição de regiões

De acordo com a listagem acima, ao definir uma região, atribuímos a ela um id único no documento, nome que será referenciado nos descritores das mídias associadas a esta região. Uma região pode ser aninhada à outra região para facilitar a definição de posicionamento e dimensionamento relativo entre regiões. Os atributos *height, width, left, e top* definem a altura, largura, a coordenada esquerda e superior da região. Utilizamos valores em percentagem, mas também podemos inserir valores em pixels. Podemos ainda definir ainda os seguintes atributos de regiões:

- ***title*:** titulo da região.
- ***right**:* coordenada direita da região.
- ***bottom*:** coordenada inferior da região.
- **zIndex:** é utilizado para indicar, no caso de regiões sobrepostas, quais regiões aparecem sobre quais outras.

Vale ressaltar que os atributos de posicionamento *left* e *width* tem precedência sobre o atributo *right*, assim como os atributos *top* e *height* tem procedência sobre o atributo *bottom*, caso todos os eles estejam especificados [Soares Neto 2007].

1.6.2. Definindo Descritores

É através de descritores que associamos uma mídia a uma região. Mas também é através dos descritores que podemos definir a forma como a mídia será apresentada, por exemplo, o volume de uma mídia de áudio, o grau de transparência de uma mídia de imagem, dentre outras propriedades. Os descritores devem ser definidos no cabeçalho do programa (<head>), dentro da seção de base de descritores (<descriptorBase>). Toda mídia que usar aquele descritor, estará associada à região correspondente. Na listagem 1.3 conferimos um exemplo de como definir um descritor:

```
<head>
      <descriptorBase>
            <descriptor id="descTela" region="rgTela"/>
            <descriptor id="descVideo1" region="rgVideo1"/>
      </descriptorBase>
</head>
```

Listagem 1.3. Definição de descritores.

Deve-se, ao definir um descritor, atribuir um nome único ao atributo *id* para ser utilizado nas referências ao descritor, e associar ao atributo *region*, uma região onde se deseja que a mídia associada seja apresentada. Além desses atributos, podemos definir a duração ideal do objeto de mídia associado ao descritor através do atributo *explicitDur* ou até definir efeitos de transação para inicio e termino da apresentação de uma mídia através dos atributos *transIn* e *transOut*, respectivamente.

A NCL define ainda dentro de um elemento de descritor, um elemento chamado *descriptorParam*. O *descritptorParam* define um parâmetro do descritor como um par (<propriedade, valor>) onde se pode ajustar propriedades como volume (*soundLevel*), no caso de uma mídia que contem áudio, ou também a intensidade da transparência (*transparency*) de um uma imagem. Abaixo, na listagem 1.4, segue um exemplo de como utilizar o *descriptorParam*.

```
<descriptor id="descVideo1" region="rgVideo1">
      <descriptorParam name="soundLevel" value="0.5" />
</descriptor>
```

Listagem 1.4. Definição de um parâmetro de descritor.

Dentre os parâmetros reservados para descritores, podem-se destacar os seguintes:

- *fit:* Modificará a mídia, especialmente vídeo e imagem, para que melhor se apresente em uma determinada região, e poderá receber como valor *fill, hidden, meet, meetBest ou slice.*
- *bounds:* modificará a posição e as dimensões do objeto de mídia. Trate-se de quatro números separados por vírgula, na ordem <left, top, width, height>, em formato de percentagem ou numero de pixels.
- *transparency:* numero real entre 0 e 1, onde 0 significa totalmente opaca e 1 totalmente transparente.

1.6.3. Definindo nós de mídia

Na NCL, nós de mídia representam o conteúdo que desejamos apresentar em um programa audiovisual interativo, que pode ser desde vídeos, imagens, textos, dentre outros. Todo nó de mídia é definido dentro de um contexto, que no NCL o elemento

body é o contexto onde contem todos os nós do documento, sejam nós de mídia ou contexto. A figura 1.3 ilustra um documento com seis nós de mídia, onde cinco estão dentro de contextos aninhados ao *body*.

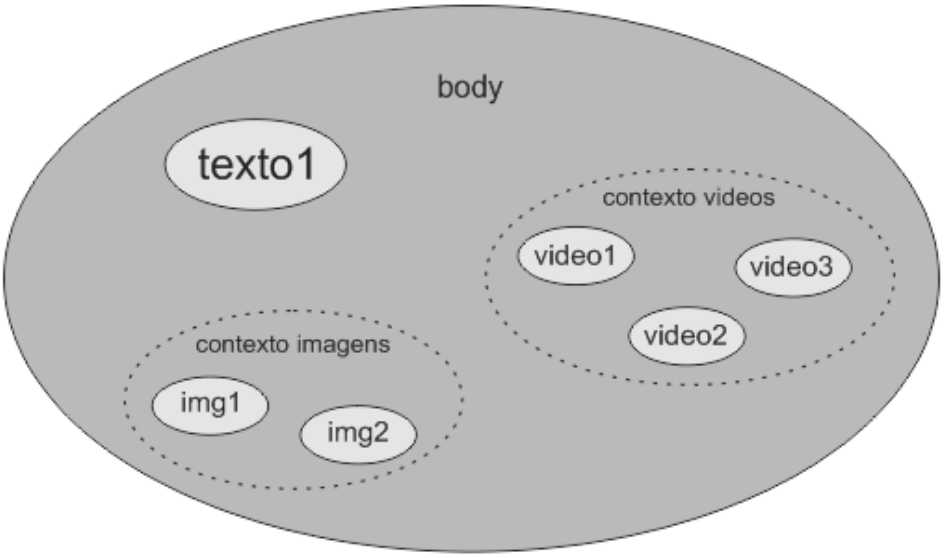


Figura 1.3. Representação de nós de mídia e de composição (contexto).

Para definir objetos de mídia, devemos inserir o elemento <media> dentro do elemento <body>, onde cada objeto de mídia tem alguns atributos principais além do atributo id: o atributo *src* que define um URI (Uniform Resource Identifier) do conteúdo do objeto, o atributo *type*, que define o tipo do objeto, e o atributo *descriptor*, identificador do descritor que controla a apresentação do objeto de mídia. Um exemplo de nó de mídia é o seguinte (listagem 1.5):

```
<body>
      <media id="video1" descriptor="descVideo1"
      type="vídeo/mpeg" src="media/vídeo1.mpg"/>
</body>
```

Listagem 1.5. Definição de um nó de mídia

Vimos que neste exemplo foi criado um nó de mídia fazendo referência, através do atributo *src*, a um arquivo chamando *video1.mpg* que se encontra no diretório media. Segundo a ABNT [NBR 15606-2 2007], os URIs devem estar de acordo com a tabela 1.1.

Tabela 1.1. URIs permitidos

Esquema	Parte especifica do esquema	Uso
file:	//file_path/#fragment_identifier	Para arquivos locais
http:	//server_identifier/file_path/#fragment_identifier	Para arquivos remotos buscados pelo canal de interatividade usando o protocolo http.
rstp:	//server_identifier/file_path/#fragment_identifier	Para fluxos (streams) obtidos pelo canal de interatividade usando o protocolo

		rstp.
rtp:	//server_identifier/file_path/#fragment_identifier	Para fluxos (streams) obtidos pelo canal de interatividade usando o protocolo rtp.
sbtvd-ts:	//program_id	Para fluxos elementares recebidos pelo fluxo de transporte (TS).

Como vimos anteriormente, contextos são usados para estrutura o nosso documento hipermídia. Podemos definir contextos utilizando o elemento <context> atribuindo um *id* como identificador único dentro do documento (listagem 1.6).

```
<body>
      <context>
            <media id="video1" descriptor="descVideo1"
            type="vídeo/mpeg" src="media/vídeo1.mpg"/>
            <media id="video2" descriptor="descVideo2"
            type="vídeo/mpeg" src="media/vídeo2.mpg"/>
      </context>
</body>
```

Listagem 1.6. Definição de um contexto

1.6.4. Âncoras

Um nó de mídia ou de composição também pode ter âncoras associadas. Através de âncoras é possível definir segmentos ou propriedades de um nó de mídia. Existem dois tipos de âncoras: âncoras de conteúdo e âncoras de propriedade. Definido com o elemento <area> dentro do elemento <media>, as âncoras de conteúdo definem um segmento da mídia, como um intervalo de tempo, que poderá ser utilizado como um ponto de ativação de elos. Como ocorre normalmente, o elemento <area> deve ter um identificador único para o atributo *id.*

Abaixo, na listagem 1.7, são definidas três ancoras de conteúdo para um nó de vídeo. O atributo *begin* define o momento no qual a âncora será ativada. Já o atributo *end* determina o momento em que a âncora será encerrada.

```
<media type="vídeo/mpeg" id="video1" src="media/video1.mpg"
descriptor="descVideo1">
      <area id="aVideoLegenda01" begin="10s" end="14s"/>
      <area id="aVideoLegenda02" begin="15s" end="19s"/>
      <area id="aVideoLegenda03" begin="20s" end="30s"/>
</media>
```

Listagem 1.7. Definição de âncoras de conteúdo.

As âncoras de propriedade definem propriedades de uma mídia e permitem que elas sejam manipuladas. Uma propriedade é definida pelo elemento <property> dentro do elemento <media>. O elemento <property> contem apenas os atributos *name* e *value*. Exemplos de propriedades são: volume de áudio de um nó de áudio ou vídeo, coordenadas das dimensões de uma imagem, dentre outras. A listagem 1.8 exemplifica a criação de quatro propriedades, relativas ao posicionamento (*top* e *left*), altura (*height*) e largura (*width*) de um nó de mídia.

```
<media type="vídeo/mpeg" id="video2" src="media/video2.mpg"
descriptor="descVideo2">
      <property id="top"/>
      <property id="left"/>
      <property id="height"/>
      <property id="width"/>
</media>
```

Listagem 1.8. Definição de âncoras de propriedade.

No NCM, todo nó (mídia ou contexto) deve obrigatoriamente possuir uma âncora com uma região representando o conteúdo total do nó. Essa âncora é chamada de âncora de conteúdo total e é declarada por omissão (default) em NCL. Cada vez que um componente NCL é referenciado sem especificar uma de suas ancoras, deve-se obrigatoriamente assumir a âncora de conteúdo total.

1.6.5. Sincronizando com elos e conectores.

Até agora aprendemos a definir quais elementos de mídia serão apresentados, onde e como. Agora teremos que definir quando esses elementos serão apresentados. E a primeira coisa a fazer é definir qual o primeiro nó do documento a ser apresentado. Então, deve-se criar uma porta no contexto *body* para esse nó.

A porta, representada pelo elemento <port>, é um ponto de interface de um contexto. Em outras palavras, para termos acesso a um determinado nó interno a um contexto, este contexto deve possuir uma porta que leve ao nó interno desejado. Um exemplo de definição de porta segue abaixo onde o atributo *id* define o nome da porta, único, e o atributo *component* define que nó será acessado em um determinado contexto (listagem 1.9).

```
<port id="portaInicio" component="video1"/>
```

Listagem 1.9. Definição de porta.

Em uma aplicação hipermídia mais elaborada requer que seja possível para o autor especificar relacionamentos entre mídias. A NCL utiliza de relacionamentos do tipo: quando começar a exibição da mídia X deve terminar a exibição da mídia Y. Esse relacionamento entre nós é escrito em NCL por meio de elos (<*link*>). O comportamento desses elos é dado por conectores, onde são definidas restrições ou sentenças causais, onde uma determinada ação é realizada quando uma condição é satisfeita.

No NCM e na NCL, o sincronismo é feito por mecanismos de casualidade e restrição definidos nos conectores (*connectors*). O conector define os papeis (roles) que os nós de origem e destino exercem nos elos que utilizam o conector, e ainda define condições (*condition*) sob as quais o elo pode ser ativado, e as ações (*action*) que serão realizadas quando o elo for ativado. Conceitualmente, conectores podem representar qualquer tipo de relação hipermídia, tal como relações de referencia, relações de sincronização, relações semânticas, relações de derivação etc.

```
<head>
      <connectorBase>
            <causalConnector id="onEndStart">
                  <simpleCondition role="onEnd"/>
                  <simpleAction role="start"/>
            </causalConnector>
      </connectorBase>
</head>
```

Listagem 1.10. Definição de conector.

Acima, na listagem 1.10, vimos que os conectores são especificados no elemento <connectorBase> dentro do cabeçalho (<*head*>). Um conector causal deve possuir pelo menos uma condição e uma ação, além de um identificador único para o atributo *id.* O conector *onEndStart* indica a condição sob a qual o elo será ativado (*onEnd*) e qual ação (*start*) será efetuada quando da ativação do elo.

Usualmente, os conectores são criados apenas uma vez, em um arquivo separado, para depois serem importados e reutilizados por diversas aplicações NCL. Ao importar um conector, especificamos um “apelido” (*alias*) à base de conectores, para que seja referenciado quando da criação de elos, e definimos onde o documento NCL deve procurar pela base de conectores através do atributo *baseURI*. Para importar uma base de conectores, podemos fazer o seguinte (listagem 1.11):

```
<head>
      <connectorBase>
            <importBase alias="conn" baseURI="connectorBase.ncl"/>
      </connectorBase>
</head>
```

Listagem 1.11. Importando base de conectores.

Um elo é uma entidade NCM que possui duas propriedades adicionais: um conector e um conjunto de associações a este conector [Soares e Rodrigues 2007]. Os elos (elemento <link>) associam os objetos de mídia (nós) através de conectores (*connectors*), que definem a semântica da associação entre os nós. Um elo é criado utilizando-se conectores e aplicando nós de mídia ou contexto a papeis (*roles*) estabelecidos pelo conector.

```
<body>
      <link id="endVideo1StartVideo2" xconnector="conn#onEndStart">
            <bind component="video1" role="onEnd" />
            <bind component="video2" role="start" />
</body>
      </link>
</body>
```

Listagem 1.12. Definição de elo utilizando um conector importado.

A listagem 1.12, acima, descreve a criação de um elo. A NCL define *id* e *xconnector* como atributos de elos, onde o primeiro é o identificador do elo, único no documento, e o segundo faz referência ao conector, que neste caso foi importado. O elemento *bind* indica um elemento envolvido no elo, indicando seu papel (*role*) no elo, conforme a semântica do conector. Um elo pode conter diversos elementos *bind*, e deve conter pelo menos um *bind* para papel definido no conector.

1.7. Lua

A linguagem Lua (Ierusalimschy, 2006) foi criada na PUC-RIO em 1993, por Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes. O objetivo era criar uma linguagem portável, simples, pequena e eficiente. Lua é linguagem de programação de extensão, isto é, o código Lua deveria ser embutido em um programa hospedeiro para funcionar. A linguagem foi projetada para dar suporte à programação procedural, mas oferece mecanismos para programação orientada a objetos, programação funcional e programação orientada a dados.

Lua evoluiu e passou a ser uma linguagem auto-suficiente. Hoje a linguagem é utilizada em vários tipos de aplicações, tais como: aplicações para TV Digital, aplicações web, aplicações desktop, em jogos, em analisador de protocolos, em rastreador de redes de segurança etc.

1.7.1. Convenções e itens léxicos

Os nomes ou identificadores na linguagem Lua podem ser qualquer cadeia de caracteres, dígitos e sublinhados, porém não podem começar com um dígito. Esses identificadores são utilizados para nomear variáveis, funções e campos de tabelas. Algumas palavras não podem ser utilizadas como identificadores, por serem reservadas para uso da gramática da linguagem (tabela 1.2).

Tabela 1.2. Palavras reservadas.

and	Return	true
end	Do	elseif
in	Nil	if
repeat	then	or
break	Else	until
false	function	while

local	Not	

A linguagem também define itens léxicos para definir operadores aritméticos, relacionais, de concatenação e de tamanho (tabela 1.3).

Tabela 1.3. Itens léxicos.

+	-	*	/	%	^	#
==	~=	<=	>=	<	>	=
(	)	{	}	[	]	
;	:	,	.	..	...	

Lua é uma linguagem “case sensitive", ou seja, diferencia identificadores maiúsculos e identificadores minúsculos. A definição do identificador ERCEMAPI vai ser interpretada de uma forma distinta de ercemapi.

Um comentário curto é definido com um hífen duplo (--), desde que o hífen duplo seja definido fora de uma cadeia de letras e que não seja precedido de colchete longo. O comentário longo é definido com um hífen duplo, precedido de colchete longo e se estende até o fechamento do colchete longo (listagem 1.13).

```
-- Comentário de uma linha
--[[ Esse é um comentário de
     múltiplas linhas]]
```

Listagem 1.13 Comentário curto e longo.

1.7.2. Variáveis

Variáveis são lugares utilizados para armazenar determinados valores. Em Lua, por padrão as variáveis são globais. Para declarar uma variável local, deve-se utilizar a palavra reservada *local* antes do identificador (listagem 1.14). Campos de tabelas também são tipos de variáveis.

```
name = “ERCEMAPI” –- variável global
print(name) –- ERCEMAPI

local name = “ERCEMAPI” –- variável local
print(name) –- ERCEMAPI
```

Listagem 1.14 variável global e local.

1.7.3. Atribuição

Lua aceita atribuições múltiplas, mas não trata atribuição como uma expressão. Se houver mais valores do que variáveis, os valores excedentes são descartados e se houver mais variáveis do que valores, as variáveis excedentes recebem *nil* (figura). Utiliza-se o operador de atribuição =, para atribuir um valor a uma variável (listagem 1.15).

```
x = 3 –- x é igual a 3
x, y = 3, 2 -- x é igual a 3 e y é igual a 2

x, y, z = 3, 2 -- x é igual a 3, y é igual a 2 e z é nil
x, y = 3, 2, 1 -- x é igual a 3, y é igual a 2 e o valor 1 é descartado
```

Listagem 1.15 atribuição.

1.7.4. Coerção

Lua é fracamente *tipada*. Em operações aritméticas entre *string* e *number*, a linguagem tenta fazer coerção para *number*. Quando um *number* é utilizado onde uma *string* é esperada, o *number* é convertido para uma *string* em formato razoável (listagem 1.16).

```
local x = “x é igual a”
local y = “y”
local z = x..y
print(z) –- x é igual a y

local value1 = “1”
local value2 = 2
local value3 = value1 + value2
print(value3) –- 3
```

Listagem 1.16 Coerção.

1.7.5. Valores

Lua é uma linguagem dinamicamente tipada, ou seja, os valores possuem os tipos e não as variáveis. Todos os valores são de primeira classe, logo os valores podem ser passados como argumentos, retornados como resultado, inclusos em uma estrutura de dados e armazenados em variáveis.

1.7.6. Tipos

A linguagem Lua conta com apenas 8 tipos: *nil, boolean, number, string, function, userdata, thread e table.* Os tipos *userdata* e *thread* não serão abordados no minicurso. A função *type* retorna o tipo do argumento passado e a função *print* imprime os argumentos passados.

1.7.6.1. Nil

Representa a ausência de um valor útil. Toda variável não inicializada tem o valor *nil*.

```
local name
print(type(name)) -- nil
```

Listagem 1.17 Tipo nil.

1.7.6.3. Number

Equivale a um double de precisão dupla por padrão, mas também representa números inteiros. É possível construir interpretadores Lua que usem outra representação interna para números.

```
local value = 1
print(type(value)) -- number
```

Listagem 1.19 Tipo number.

1.7.6.4. String

Representa cadeias de caracteres literais. Podem ser delimitadas através do uso de aspas simples (‘ ’), aspas duplas (“ ”) e formato longo ([[ ]]) para escrever cadeias de caracteres que se estendem por varias linhas. As strings também podem conter sequências de escape no estilo da linguagem C.

```
local name = “ERCEMAPI”
print(type(name)) –- string

local name = ‘ERCEMAPI’
print(type(name)) -- string

local name = [[ERCEMAPI]]
print(type(name)) -- string
```

Listagem 1.20 Tipo string.

1.7.6.5. Function

Representam as funções. Em Lua, as funções são valores de primeira classe e podem ter nenhum ou vários parâmetros e retornar nenhum ou vários valores.

```
function foo(x)
  print “function”
end
ou
foo = function(x)
  print “function”
end
print(type(foo)) – function
```

Listagem 1.21 Tipo function.

1.7.6.6. Table

Através do tipo *table* é possível representar em Lua arrays comuns, registro, grafos, árvores, tabelas de símbolos, classes, objetos etc. As tabelas podem conter todos os tipos de valores (exceto *nil*) (listagem 1.22).

```
local t = {} -- tabela vazia


local t = {1, "ERCEMAPI", true}
-- [[tabela com 3 valores. t[1] é igual a 1, t[2] é a string
ERCEMAPI e t[3] é o boolean true]]


local t = {name = "ERCEMAPI"}
-- t["name"] imprime ERCEMAPI e t.name imprime ERCEMAPI
```

Listagem 1.22 criação de tabelas.

1.7.7. Operadores

A linguagem Lua oferece um conjunto de operadores para realizar operações aritméticas (tabela 1.4), relacionais (tabela 1.5), lógicas (tabela 1.6) e de concatenação (tabela 1.7). O operador de atribuição é o sinal de =.

Tabela 1.4. Operadores aritméticos.

Operadores	Significado	Exemplo
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b
%	Módulo	a % b
^	Exponenciação	a ^ b
-	Negação	- a

Tabela 1.5. Operadores relacionais.

Operadores	Significado	Exemplo
==	Igualdade	a == b
~=	Diferente	a ~= b
<	Menor que	a < b
>	Maior que	a > b
<=	Menor ou igual a	a <= b
>=	Maior ou igual a	a >= b

Tabela 1.6. Operadores lógicos.

Operadores	Significado	Exemplo
and	E	a and b
or	Ou	a or b
not	Não	not a

Tabela 1.7. Operador de concatenação.

Operadores	Significado	Exemplo
..	Concatenação	a..b

1.7.8. Estruturas de controle

Em determinada circunstâncias é necessário executar algumas instruções baseadas em determinadas condições. Essas instruções serão executadas se uma condição for satisfeita ou repetir varias vezes a mesma estrutura até satisfazer uma condição. Lua prevê algumas estruturas para controle do fluxo de execução.

1.7.8.1. If then else

O comando *if .. then* representa uma tomada de decisão e permite a seleção de dois ou vários caminhos distintos em uma execução, dependendo do retorno da expressão analisada. Se a expressão avaliada for verdadeira a parte *then* é executada, senão a estrutura do *else* é executada.

```
local name = “ERCEMAPI”
if name == “ERCEMAPI” then
  print(name)
else
  print(“name não é igual a ERCEMAPI”)
```

Listagem 1.23 estrutura de controle if.

1.7.8.2. While

O *while* realiza um teste na expressão lógica na entrada da estrutura. Enquanto o teste for verdadeiro o bloco é executado.

```
t = {1, 2, 3}
local i = 1
while t[i] do
  print(t[i]) -- bloco a ser executado se o teste passar
  i = i + 1
```

Listagem 1.24 estrutura de controle while.

1.7.8.3. Repeat

A estrutura *repeat-until* realiza um teste na expressão lógica em *until*. O bloco só termina depois da condição de *until* for satisfeita.

```
t = {1, 2, 3}
local i = 1
repeat
  print(t[i])
  i = i + 1
until t[i] == nil
```

Listagem 1.25 estrutura de controle repeat.

1.7.8.4. For

Em Lua, o comando for pode ser numérico ou genérico. O for numérico a interação inicia com valor de *exp1*, prosseguindo até *exp2* com passos do tamanho *exp3*. Se *exp3* for omitida, o passo é 1.

```
local exp2 = 6
local exp3 = 3
for exp1 = 1, exp2, exp3 do
  print(exp1)
end
```

Listagem 1.26 for numérico.

O for genérico utiliza funções iteradoras, como pairs e ipairs. A cada iteração, essas funções produzem um valor.

```
t = {name = “ERCEMAPI”}
for key, value in pairs(t) do
  print(key..” => ”..value) -- imprime name => ERCEMAPI
end
```

Listagem 1.27 for genérico.

1.8. Integração NCL e Lua (NCLua)

Um documento NCL reconhece um script NCLua como uma mídia (listagem 1.18), tal como uma imagem, vídeo etc. Quando o script é iniciado, um objeto é instanciado e atribuído a variável global *canvas*. Esse objeto tem as mesmas dimensões definidas na região da mídia NClua (listagem 1.28) e é através desse objeto que as funções do módulo *canvas* podem ser utilizadas. O módulo *canvas* permite a possibilidade de fazer operações gráficas, tais como: desenhar círculos, linhas, retângulos, polígonos etc (listagem 1.20).

```
<media id="lua" src="frame/lua.lua" descriptor="dLua"/>
```

Listagem 1.28 Definindo uma mídia NCLua.

A linguagem Lua tem bibliotecas padrão para manipulação de strings, tabelas, entrada e saída de dados, funções matemáticas, pacotes, sistema operacional, depuração e uma biblioteca com funções básicas para a linguagem. A forma de integração entre os objetos NCLua e os documentos NCL é através do paradigma orientado a eventos, por isso a linguagem precisou ser estendida para tal integração. Além das bibliotecas padrão, foram adicionados os seguintes módulos para serem utilizados nos scripts NCLua:

- Módulo Canvas: oferece uma API que permite a alteração e manipulação de primitivas gráficas e imagens.
- Módulo Event: permite que aplicações NCLua tornem-se orientadas a eventos.
- Módulo Settings: permite a manipulação de variáveis definidas em um documento NCL e em variáveis reservadas de um objeto do tipo “aplication/x-ginga-settings”.
- Módulo Persistent: permite que variáveis persistentes sejam acessadas e manipuladas por objetos imperativos.

Um script NCLua pode interagir com diversas entidades, tais como: eventos do controle remoto, eventos do canal de interatividade, documentos NCL etc. Para que essa comunicação ocorra pelo menos uma função tratadora deve ser registrada no script NCLua através da função event.register (listagem 1.29).

```
function handler(evt)     -- função que trata os eventos
  ...
end
event.register(handler)   -- registro da função tratadora
```

Listagem 1.29 Registro de uma função tratadora de eventos.

Um evento é definido através de uma tabela Lua, onde *class* representa a classe do evento. Na linstagem 1.29, a tabela do evento indica que o botão vermelho foi pressionado, onde *type* pode ser *press* ou *release*, *key* é o valor da tecla do controle.

```
evt = { class ='key', type = 'press', key = 'RED'}
```

Listagem 1.29 Registro de uma função tratadora de eventos.

O módulo *event* define as seguintes classes de eventos:

- Classe ‘ncl’: representa o pressionamento dos botões do controle remoto pelo usuário.
- Classe ‘user’: Através dessa classe, aplicações podem estender suas funcionalidades criando seus próprios eventos.
- Classe tcp: Permite acesso ao canal de interatividade por meio do protocolo tcp.

- Classe sms: Usada para envio e recebimento de mensagens SMS em dispositivos móveis.
- Classe edit: Permite que os comandos de edição ao vivo sejam disparados a partir de scripts NCLua.
- Classe si: Provê acesso a um conjunto de informações multiplexadas em um fluxo de transporte e transmitidas periodicamente por difusão.

Na norma ABNT NBR 15606-2:2007 [ABNT 2007], pode-se encontrar todas as funções do módulo *canvas* e mais informações sobre os módulos que ainda não foram implementados.

1.10. Referências

ABNT NBR 15606-2 (2007) – Associação Brasileira de Normas Técnicas, "Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

Barbosa, S. D. J., Soares, L. F. G., "TV digital interativa no Brasil se faz com Ginga: fundamentos, padrões, autoria declarativa e usabilidade".

Ierusalimschy, R. (2006). "Programming in Lua", Segunda Edição. http://lua.org.

NCL - Nested Context Language. http://www.ncl.org.br/, 2010

Sant'Anna, F. F. G., Soares Neto, C. S., Azevedo, R. G. A. e Barbosa, S. D. J. "Desenvolvimento de Aplicações Declarativas para TV Digital no Middleware Ginga com objetos imperativos NCLua".

Sant'Anna, F. F. G., Soares, L. F. G. e Cerqueira, R. F. G., Nested Conext Language 3.0 Part 10 – Imperative Objects in NCL: The NCLua Objects, 2008.

Soares Neto, C. S. et al. Construindo Programas Audiovisuais Interativos utilizando a NCL 3.0 e a ferramenta Composer. 2007

Soares, L. F. G., Rodrigues R. F. Nested Context Model 3.0 Part 1 – NCM Core, 2005.

Soares, L. F. G. e Barbosa, S. D. J. (2009) Programando em NCL 3.0. São Paulo/SP. 1.ª ed. 2009.

Capítulo 2

Utilização do CUDA e OpenCL em Algoritmos de Visão Computacional

Thiago S. Santos, Kalyf A. B. Lima, Kelson R. T. Aires

Abstract

The Computer Vision can be viewed as an entity of automation and integration of a wide range of processes and representations used in perception, including techniques such as image processing and pattern classification. The high computational cost, from the techniques of data processing, requires a high processing power. In order to circumvent this problem, one solution is to use GPU (Graphics Processing Units). This paper presents an approach to the use of CUDA and OpenCL in Computer Vision. Experiments with edge detection algorithms and patterns will be undertaken in order to demonstrate the effectiveness of this method.

Resumo

A Visão Computacional pode ser vista como uma entidade de automação e integração de uma larga extensão de processos e representações usados na percepção, incluindo técnicas como processamento de imagens e classificação de padrões. O alto custo computacional, proveniente das técnicas de processamento de dados, requer um poder de processamento elevado. Para contornar esse problema, uma solução é a utilização de unidades de processamento gráfico (GPU - Graphics Processing Units). Este trabalho apresenta uma abordagem do uso do CUDA e OpenCL em Visão Computacional. Experimentos com algoritmos de detecção de borda e padrões serão realizados, a fim de demonstrar a eficácia deste método.

2.1. Introdução

Processamento para setores de visão computacional sempre requereram alto desempenho, os algoritmos utilizados no campo possuem excessivas repetições de cálculos para os elementos do conjunto a ser trabalhado. Apesar de avanços no campo de processamento, o mesmo não é ilimitado devido a leis físicas.

A programação paralela já se mostrou eficiente para a execução de algoritmos com cargas de dados como os já mencionados, porém, a tecnologia se tornou mais viável para o meio científico devido ao seu custo. Seu valor se justificava pela forma de como era realizado, geralmente um computador principal que divide a execução entre outros.

Placas gráficas são, em maioria, utilizadas como aceleradores de vídeo e têm seu verdadeiro potencial limitado. A tecnologia multi-nuclear das mesmas, motivo da evolução no desempenho, tem princípios mais sólidos e não estão restritos a processamento de imagens. As *threads* podem executar, de forma independente, linhas de código, tal que, funcionam como super computadores, característica ideal para a computação de alto desempenho.

A nVidia, fabricante de núcleos gráficos para computação, desenvolveu a tecnologia CUDA (*Compute Unified Device Architecture*), uma arquitetura para computação paralela. Esta permite utilização dos núcleos da GPU de forma simultânea e independente para a execução de blocos de códigos, o que a levou a ser foco e ferramenta de trabalho. A tecnologia diminuiu o custo de *hardwares* para estudo do paralelismo e desenvolvimento de *softwares* profissionais e científicos no setor de alto desempenho.

As aplicações para a tecnologia são diversas, simulações físicas, financeiras, além de químicas pela dinâmica de moléculas para descoberta de novos remédios através do AMBER. Ainda assim, a que ainda continua atraindo atenção, é o processamento gráfico utilizado principalmente para o setor robótico. Estima-se que todas as aplicações brevemente utilizem a tecnologia, acelerando suas execuções.

Indústrias em geral estudaram os princípios da tecnologia CUDA e incorporaram-nos em seus produtos, entre elas a rival AMD em sua tecnologia ATI Stream para as placas gráficas ATI Radeon. Percebendo o potencial do processamento paralelo através de núcleos gráficos e o aumento de vendas de computadores para fins científicos que necessitavam de qualidades e desempenho superiores. A Apple propõem o OpenCL como uma forma de padronizar a programação das GPUs e fez de seu sistema o primeiro a aderir ao *framework*.

O OpenCL (*Open Computing Language*) nasce como a ferramenta ideal para programação paralela, compatível com sistemas operacionais e dispositivos diversos, criando um modelo para desenvolvedores que desejam ao máximo a portabilidade de seus códigos.

2.2. CUDA

NVIDIA® CUDATM é uma proposta de arquitetura de computação paralela desenvolvida pela nVidia. A arquitetura inclui o CUDA *Instruction Set Architecture* (ISA). Um ISA inclui uma especificação do conjunto de *opcodes* (linguagem de máquina), os comandos nativos implementados por um determinado processador, e o mecanismo de computação paralela na Unidade de Processamento Gráfico (*Graphic Processor Unit* – GPU). Para programar para a arquitetura CUDA, os desenvolvedores podem optar por C, uma linguagem de alto nível, capaz de executar com grande desempenho em um processador com suporte CUDA.

A arquitetura CUDA foi desenvolvida com vários objetivos em mente, entre eles:

- Fornecer um pequeno conjunto de extensões para linguagens de programação padrão que permitem uma aplicação direta de algoritmos paralelos. Com CUDA os programadores podem focalizar na tarefa de paralelização dos algoritmos ao invés de gastar tempo na sua implementação [1];

- Ter suporte a computação heterogênea onde aplicações utilizam ambos, CPU e GPU. Partes seriais das aplicações são executadas na CPU, e as parcelas paralelas são descarregadas para a GPU. A CPU e a GPU são tratadas como dispositivos separados que têm seus próprios espaços de memória.

As GPUs com suporte CUDA têm centenas de núcleos que podem executar coletivamente milhares de *threads*. Cada núcleo tem recursos compartilhados, incluindo registradores e memória. A memória compartilhada nos *chips* permite a execução de tarefas paralelas sobre estes núcleos para compartilhar dados sem enviá-los através do barramento de memória do sistema. Impulsionado pela demanda de mercado por alta definição e processamento em tempo-real de gráfico 3D, as unidades de processamento gráfico têm evoluído para um processador de vários núcleos com alto poder computacional, altamente paralelo e com uma larga banda de memória, como ilustrado na Figura 2.1 [1].

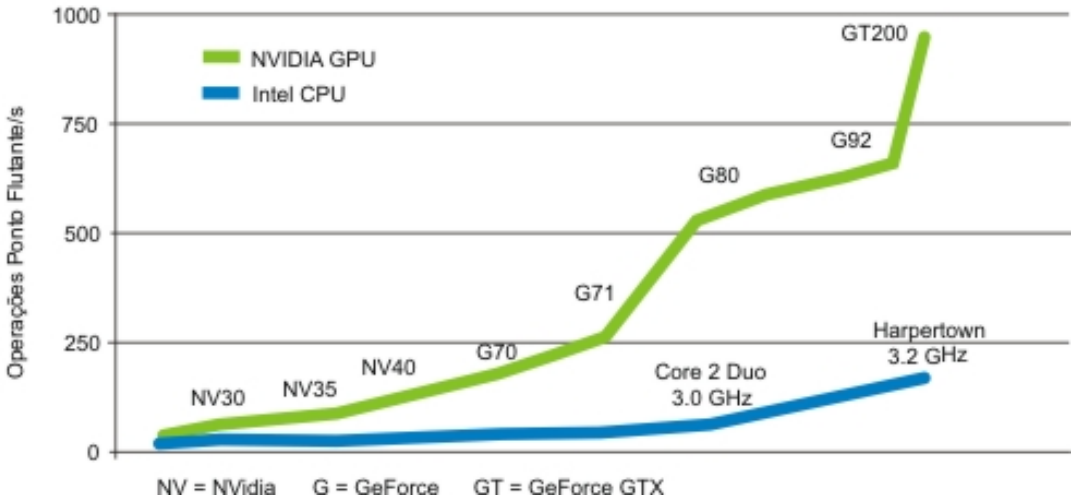


Figura 2.1. Operações Ponto Flutuante por Segundo.

A razão por trás da alta diferença em termos de capacidade de ponto flutuante entre a CPU e a GPU é o fato da última ser especializada em computação intensiva e computação paralela. Especialmente acerca da renderização de gráficos, portanto, possui transistores dedicados ao tratamento de dados ao invés de armazenar dados ou de controle de fluxo, esquematicamente ilustrado pela Figura 2.2.

Outra especialidade da GPU é consiste no fato de ela ser especialmente adequada para resolver problemas os quais podem ser expressos por cálculos paralelos de dados com alta intensidade de aritmética. Como o mesmo algoritmo é executado para cada elemento de dado, não há tanta exigência de controle de fluxo sofisticado. A latência de

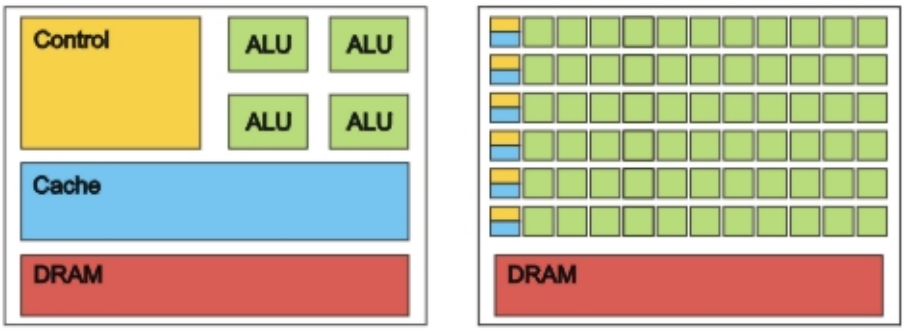


Figura 2.2. A GPU possui mais transistores para processamento de dados.

acesso à memória pode ser escondida com os cálculos em vez de grandes caches de dados, visto que possui intensiva aritmética.

No processo de programação paralela, os elementos de dados são mapeados para trilhas de processamento (*threads*). Muitas aplicações que processam grandes quantidades de dados podem utilizar um modelo de programação paralela para acelerar os cálculos. Na renderização 3D grandes conjuntos de *pixels* e vértices são mapeados para *threads* paralelas. Da mesma forma, aplicativos, mídias e processamentos de imagens, tais como pós-processamentos de imagens renderizadas, codificação e decodificação de vídeo, ampliação de imagem, visão estéreo, e reconhecimento de padrões podem mapear blocos de imagens e *pixels* para *threads* paralelas. De fato, muitos algoritmos fora do campo de processamento de imagens são acelerados por processamento de dados paralelo, desde processamento de sinais gerais ou simulações físicas até finança computacional ou biologia computacional.

A tecnologia CUDA é uma proposta geral de arquitetura de computação paralela da nVidia. Em novembro de 2006, a nVidia introduziu CUDA como um novo modelo de programação paralela e ISA (*Instrution Set Architecture*) [1]. CUDA aproveita o mecanismo de computação paralela em GPUs nVidia para resolver muitos problemas computacionais complexos de maneira mais eficiente do que em uma CPU. CUDA disponibiliza um *software* que permite aos desenvolvedores usar o C como uma linguagem de programação de alto nível, outras linguagens ou interfaces de programação de aplicativos são suportadas, tais como FORTRAN CUDA, OpenCL, Phyton e computação direta. Tal integração é ilustrada na Figura 2.3.

Um grande desafio da computação paralela é desenvolver um aplicativo escalável ao aumento do número de núcleos, da mesma forma que aplicativos gráficos em 3D que possuem transparência a grandes variações no número de núcleos. O CUDA é projetado para superar este desafio, mantendo uma baixa curva de aprendizado para programadores familiarizados com linguagens padrões como C. Na essência do CUDA existem basicamente três principais hierarquias, Grupos de *threads*, Memória Compartilhada e Sincronização, expostas como um conjunto mínimo de extensões da linguagem, como ilustrado na Figura 2.4.

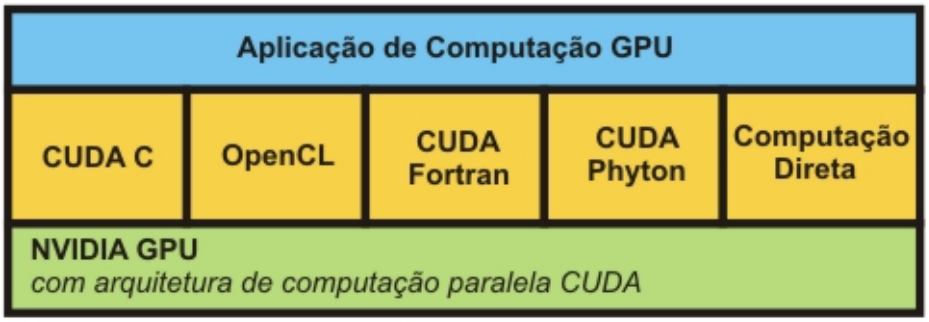


Figura 2.3. CUDA é desenvolvido para ter suporte a varias linguagens.

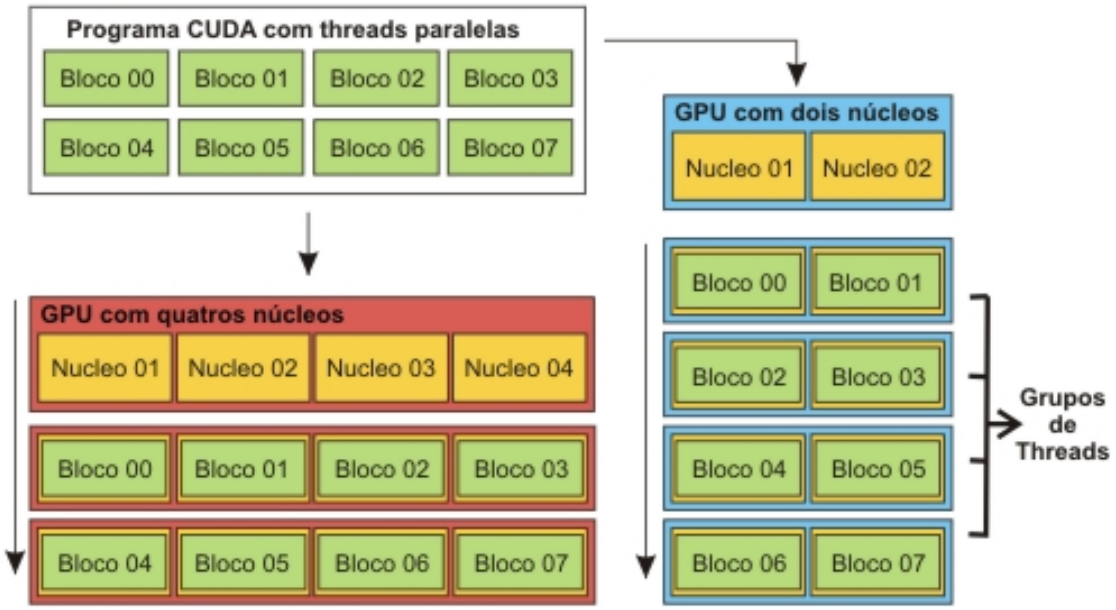


Figura 2.4. Escalabilidade Automática.

2.2.1. Kernels

CUDA C estende C permitindo ao programador definir funções C, chamadas *kernels*. Essas funções quando chamadas são executadas N vezes em paralelo por N *threads* CUDA. Os *kernels* são definidos utilizando-se o modificador de acesso _ **global**_ no cabeçalho da função e o número de *threads* CUDA é definido na chamada do *kernel*. Cada *thread* que executa o *kernel* possui um ID único, esse ID é acessível no *kernel* através da variável *threadIdx* [1].

As *threads* são organizados em hierarquias. Por definição a variável *threadIdx* é um vetor de três componentes e o resultado da normalização destas componentes é um valor único o qual a *thread* está relacionada. O número de *threads* por bloco é limitado pelo numero de núcleos do processador, contudo o *kernel* pode ser executado por múltiplos blocos de forma semelhante. Sendo assim o numero de *threads* executadas será o número de blocos vezes o número de *threads* por bloco.

Blocos de *threads* podem ser organizados dentro de uma Grade (*Grid*) unidimen-

sional ou bidimensional. O número de blocos geralmente é ditado pelo tamanho dos dados processados ou o número de processadores do sistema. As *threads* dentro de blocos podem cooperar entre si compartilhando dados através da memória compartilhada (*shared memory*), sincronizando a execução, e coordenando o acesso à memória.

2.2.2. Hierarquia de Memória

As *threads* CUDA podem acessar múltiplos espaços de dados na memória durante a execução, conforme ilustrado na figura 2-5. Cada *thread* possui sua própria memória local privada. Cada bloco de *threads* tem uma memória compartilha visível por todas as *threads* do bloco e com o tempo de vida igual ao do bloco. Todas as *threads* possuem acesso a memória global.

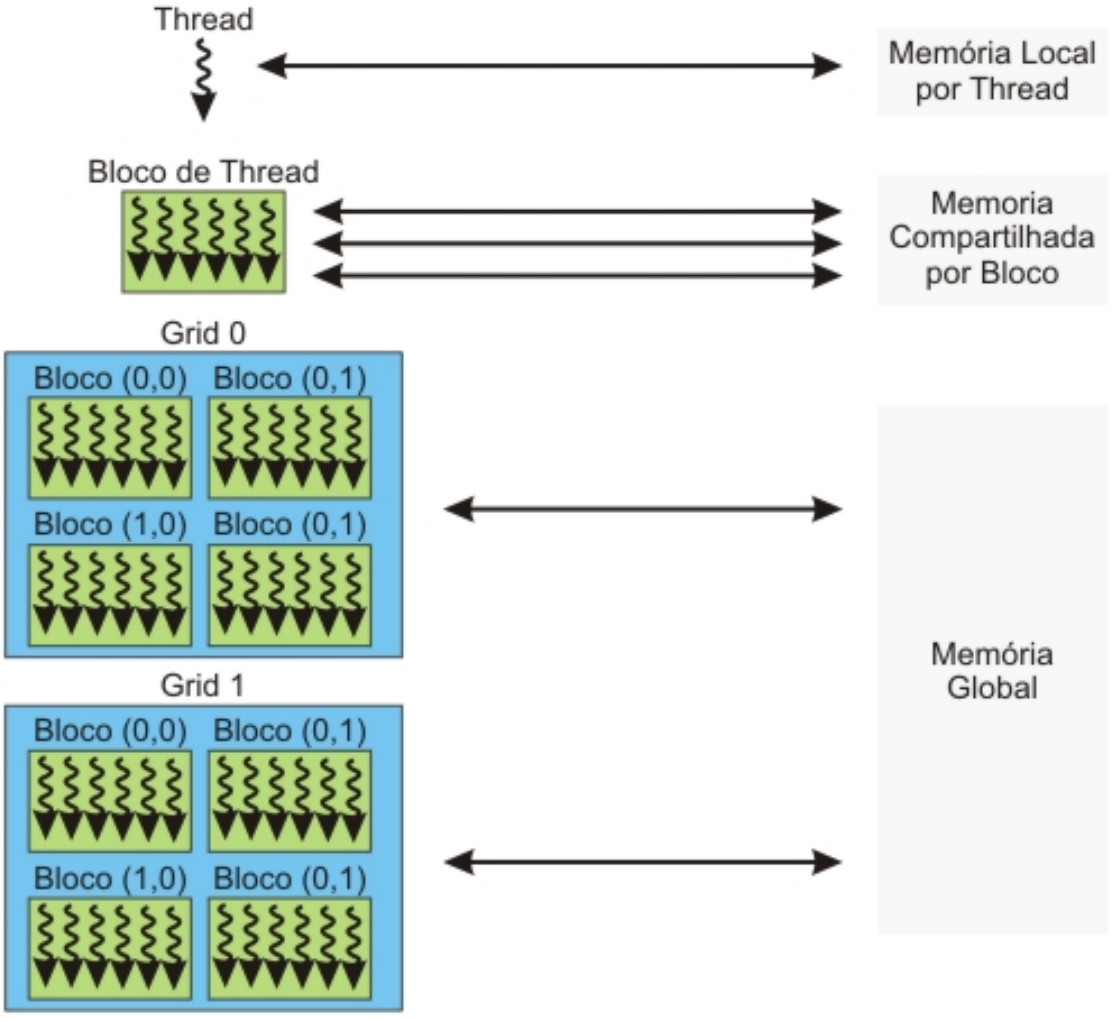


Figura 2.5. Hierarquia de Memória.

Há também dois espaços adicionais de memória do tipo "somente leitura", acessíveis a todos as *threads*: espaços de memória constante e textura. Os espaços de memória global, constante e textura são otimizados para diferentes usos. Memória de textura oferece diferentes modos de endereçamento, bem como filtragem para específicos tipos de

dados. Memória global, constante e textura são persistentes entre as execuções de *kernels* na mesma aplicação.

2.2.3. Programação Heterogênea

O modelo de programação CUDA assume que as *threads* executam em um dispositivo fisicamente separado, como pode ser observado na Figura 2.6. Dispositivo esse que funciona como um co-processador para o *host* que executa o programa em C. Este é o caso, por exemplo, quando os *kernels* executam em uma GPU e o resto do programa C é executado em uma CPU.

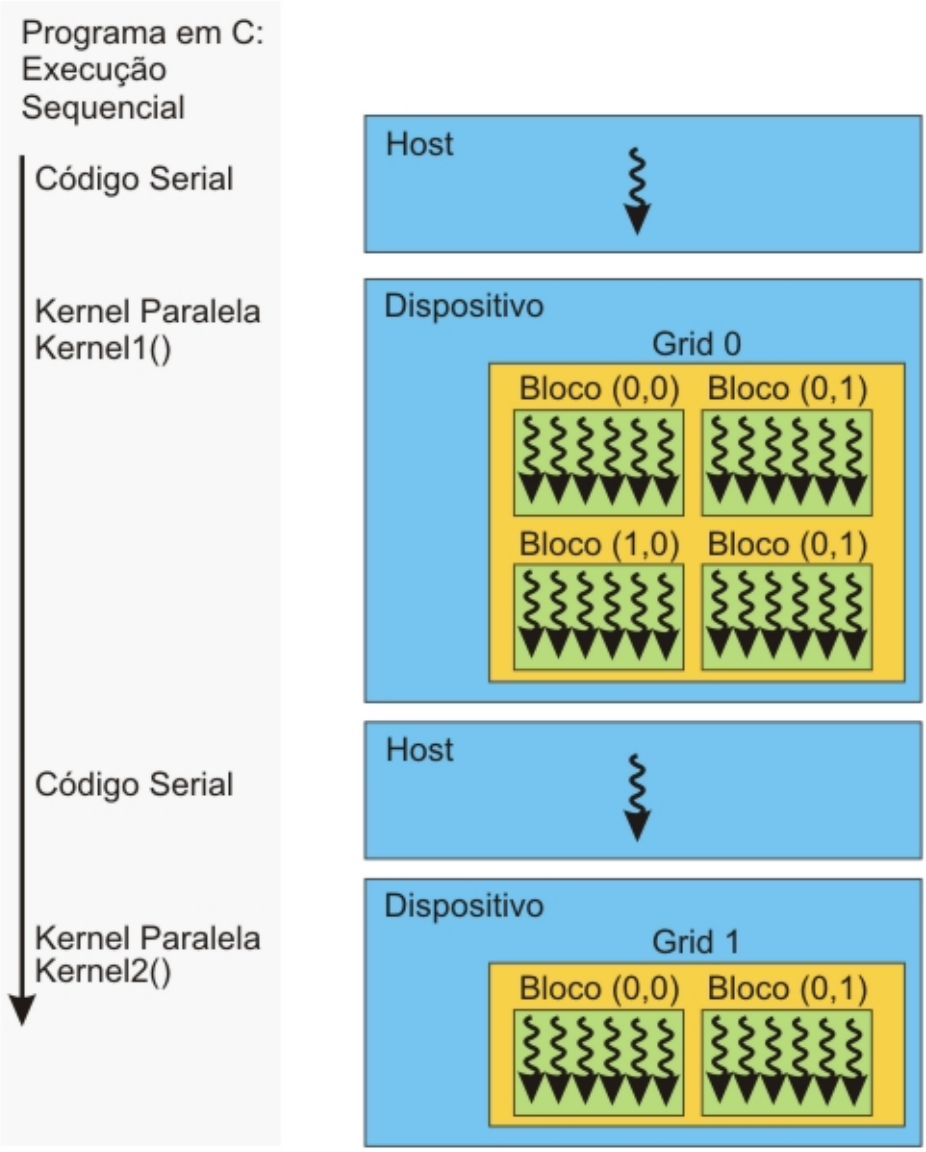


Figura 2.6. Programação Heterogênea.

O modelo de programação CUDA pressupõe que o *host* e o dispositivo mantém em separado seus próprios espaços de memória na DRAM, referenciados como memória

do *host* e memória do dispositivo, respectivamente. Portanto, um programa administra espaços de memória global, constantes e texturas visíveis para *kernels* através de chamadas CUDA. Este dispositivo inclui alocação e desalocação de memória, bem como a transferência de dados entre a memória do *host* e a do dispositivo.

2.3. OpenCL

Por limitações físicas, desenvolvedores deixaram a corrida pelo aumento de frequência nos processadores para incremento de núcleos que, por teoria, ampliaria o desempenho proporcionalmente. Um determinado código pode assim, ser repartido entre os núcleos existentes para que não ocorra uma sobrecarga e demora na execução.

A nova geração de GPUs, que são responsáveis pelo avanço da qualidade gráfica, elevou a realidade de ambientes tridimensionais, agora capazes de processar altas cargas de dados para renderizar imagens de modo que se assemelham a supercomputadores, por utilizar tecnologias com o princípio *multi-core*. Tal poder computacional pode ser utilizado não só para processamento gráfico, mas para uma nova geração de *softwares* de alto desempenho [2, 3].

O desenvolvimento de arquiteturas diversas para o paralelismo impossibilitou a portabilidade entre *softwares* de desenvolvimento, visto que, cada entidade implementava suas próprias rotinas de manipulação de dados e execução. Programadores que por ventura fossem trabalhar com dispositivo de fabricantes diferentes teriam que reescrever o código-fonte, o que se tornou inviável já que projetos complexos teriam que ser revistos elevando seu custo e tempo para lançamento.

O OpenCL surge do trabalho conjunto da Khronos Group, iniciado pela Apple com as líderes de mercado do setor, entre elas AMD, nVidia e Intel, como um novo *framework* padrão para programação paralela com rotinas padrões [4, 5]. O OpenCL permite que códigos possam ser executados em sistemas heterogêneos compatíveis sem a necessidade de modificação dos arquivos. Além disso, poupa recursos financeiros, tempo e permitindo que desenvolvedores possam criar softwares com melhores desempenhos, cujas possibilidades de aplicação vão desde games a softwares acadêmicos.

2.3.1. Rotinas

As rotinas iniciais de detecção do sistema operacional e dispositivos moldam o método de trabalho do OpenCL. Tais rotinas ajustam-no ao sistema tal que um mesmo *software* possa ter diferentes implementações para cada máquina que por ventura execute o programa, o contexto.

As etapas seguintes geram o código através de implementações das funções em OpenCL C, este pronto para compilação como observado na Figura 2.7. Devido à memória dos dispositivos não permitir acesso direto, são usados funções da ferramenta para alocação, escrita e leitura dos dados e após o uso ocorre à liberação dos mesmos. O uso das implementações do programador é indireto utilizando funções padrões para passagem de parâmetros e execução paralela.

Entre cada passagem das etapas são retornados erros, devido a cada etapa requerer a boa execução da anterior, há a necessidade de constante checagem de erros, liberação

Figura 2.7. Rotinas básicas de um programa em OpenCL.

de memórias alocadas e aborto da operação. A necessidade de colher informações recai sobre uma característica do *framework*, a forma de compilação.

2.3.2. Compilação

A partir dos dados obtidos nas rotinas, o programa é compilado em tempo de execução, adequando o código OpenCL C. Uma variante da linguagem C o que o torna amigável, facilitando seu estudo. Há o acréscimo de novos tipos em relação a ISO C99 como os específicos para processamento de imagem e o booleano incorporado pela linguagem C++ e vetores de tamanhos variados para acelerar o processo de transferência [4, 5].

Como ilustrado na Figura 2.8, na compilação para CPU é gerado um código binário x86 onde em tempo de execução é determinado o número *multi-core* do processador distribuindo para os mesmos automaticamente. Para GPUs é gerado um código incompleto, as bibliotecas do fabricante finalizam o trabalho e o executável para *kernel* é enviado para o dispositivo [2, 3].

2.3.3. Funcionamento

O paralelismo no *framework* se dá pela projeção de funções que só poderão ser invocadas pelo *kernel* do dispositivo e que serão executadas em cada item de trabalho, os quais são independentes entre si e aplicam uma mesma função. Desta forma, linhas de comando que estariam em *loops* de grandes repetições são executadas simultaneamente através de GPUs ao invés do tradicional sequenciamento.

No momento que a aplicação se mostrar pronta para execução do *kernel*, os dados são movidos para VRAM da GPU e trabalhados pelos núcleos de cada unidade de forma síncrona ou assíncrona [3] de acordo com a necessidade do programa, como esquematizado na Figura 2.9.

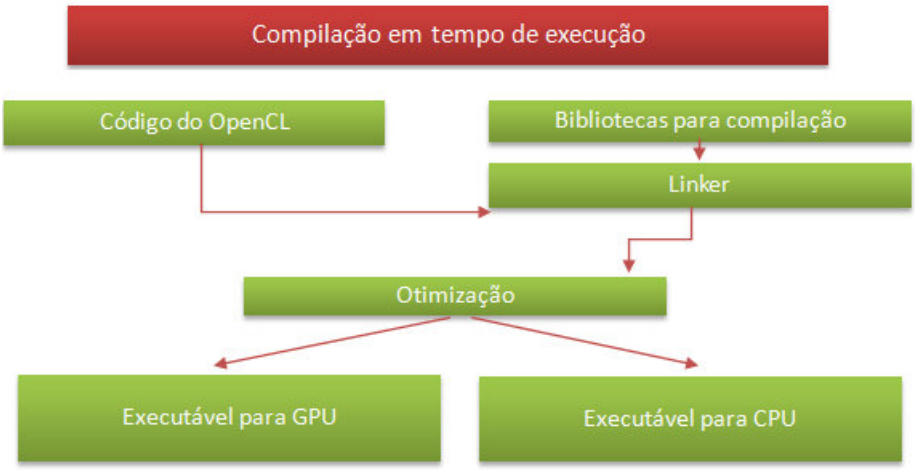


Figura 2.8. Esquema do processo de compilação do código OpenCL.

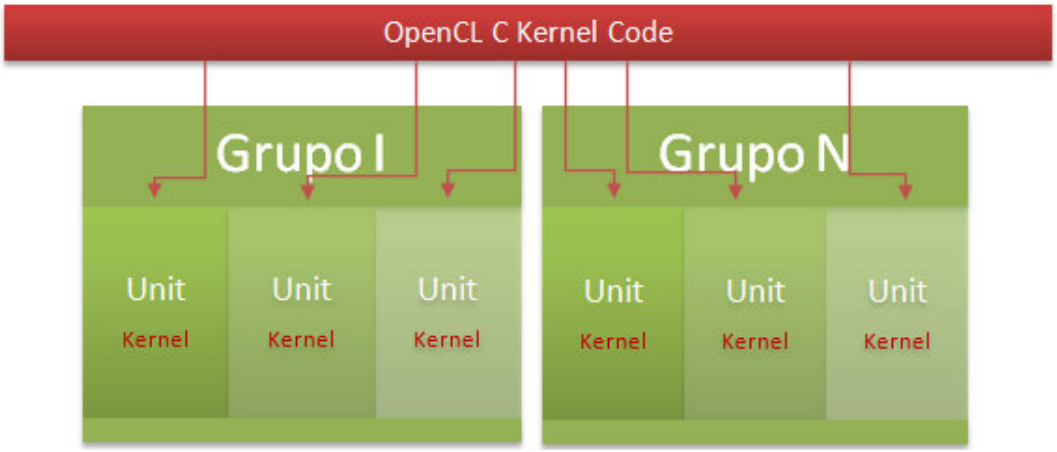


Figura 2.9. Modelo de transferência de código *kernel* para *units*.

2.3.4. Arquitetura da memória

A memória GPU se divide em memória global, constante, local e privada. A primeira visível para todas as units, podendo ser reescrita; a segunda com o mesmo nível da anterior, mas sem permissão de alteração; a terceira visível em um grupo; e a última somente na própria *thread* [3], como ilustrado pela Figura 2.10.

A separação da memória é lógica, a diferença está no escopo das mesmas, e fisicamente estão localizadas no mesmo espaço. Contudo, devido aos procedimentos de acesso, as de menor escopo têm maiores velocidades na comunicação com as *units*.

Devido ao fato dessa visibilidade, há um cuidado a ser tomado com a modificação do conteúdo de um endereço de memória por dois ou mais itens de trabalho. O fato de serem independentes implica na imprevisibilidade do resultado de múltiplas gravações simultâneas em um mesmo ponto, pois inicialmente não há um controle de precedência.

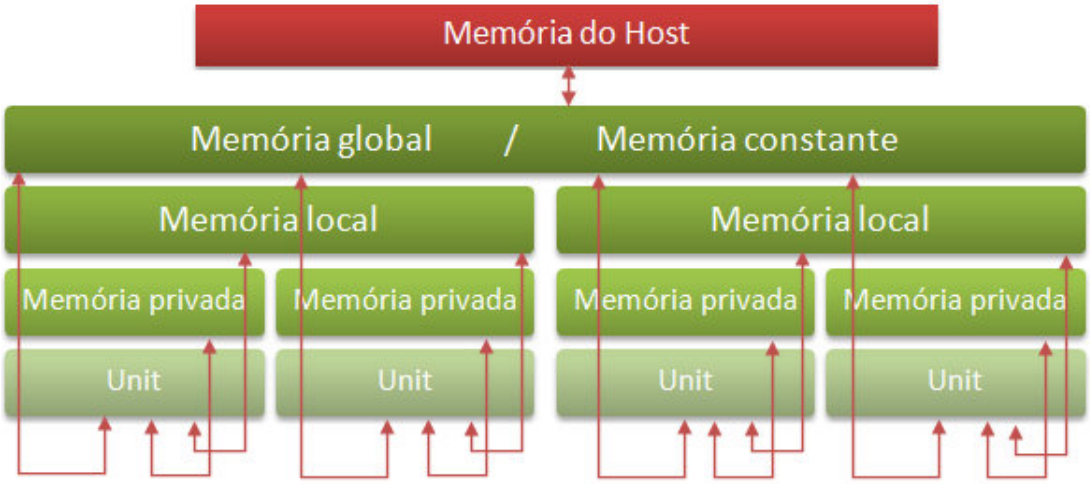


Figura 2.10. Esquema da arquitetura da memória e acesso.

O OpenCL em uma de suas extensões oferece como solução o uso de funções atômicas, ao acessar um endereço de memória o mesmo é bloqueado temporariamente para garantir a integridade de seu conteúdo, alterando-o para posteriormente liberar seu acesso para a manipulação por qualquer outra unidade elementar. A Khronos Corp. disponibilizou algumas funções básicas entre elas incremento, soma e troca que são as mais usuais. Através das mesmas é possível gerar um sinalizador com um vetor cujos índices representam determinados locais de memória indicando seu estado atual de ocupação. Tal técnica pode gerar uma pequena perda de desempenho por usar a memória global na maioria dos casos.

2.3.5. Sincronismo

Por permitir o uso de muitos dispositivos em conjunto em um mesmo programa, útil para filas de comandos distintas, contudo, interligadas e dependentes dos resultados uma da outra. O OpenCL dispõem de formas para sincronização por eventos e por barreiras para serem usadas entre filas e itens de trabalho pertencentes a um mesmo grupo [6].

As sincronizações por barreira são mais usuais entre itens de trabalho, como utilizam o princípio de que a próxima etapa só será executada após o término de todas as atuais, a utilização deste método entre filas de comando se assemelharia ao uso de uma única fila de comandos.

Sincronizações por eventos são mais relevantes em algoritmos complexos e robustos por prepararem a próxima execução antes do pedido de execução o que diminui o tempo para inicialização aumentando, portanto, o desempenho como mostrado na Figura 2.11.

2.3.6. Limitações

As transferências de dados entre as memórias do *host* e dispositivo requerem certo tempo. De tal forma, o OpenCL pode não ser a melhor opção para tarefas triviais, sendo recomendado apenas para algoritmos de cálculos complexos que necessitam de enorme poder para

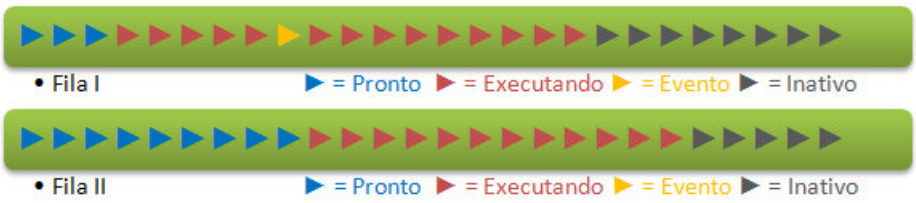


Figura 2.11. Estado de filas de comando sincronizadas por evento.

serem executados. Devido à limitação na velocidade das passagens pela memória, dados não devidamente utilizados podem gerar perda computacional.

Os compiladores atualmente estão separados, há um especificamente para GPUs e outro para CPUs, de modo que, ainda não é possível usufruir de todos os benefícios que o *framework* oferece de portabilidade entre tipos de dispositivos que executarão as *kernels* em OpenCL C.

Por ser uma tecnologia recente, a bibliografia existente é pouca, boa parte da mesma restringindo-se as fabricantes e desenvolvedores de tecnologia que a usam como AMD responsável pela ATI Stream, nVidia desenvolvedora da CUDA, ambas mostrando ótimos resultados, contudo, a nVidia sendo pioneira no setor e possui maior suporte como a questão do recurso de atomicidade de funções, restrita à grupo nas placas habilitadas a tecnologia ATI Stream.

2.4. Estado da Arte

O CUDA e o OpenCL oferecem duas interfaces para programar GPUs. O OpenCL é uma interface padrão aberta que permite programar CPUs, GPUs e outros dispositivos de diferentes vendedores, enquanto o CUDA é especifico das GPUs nVidia. Entretanto OpenCL promete uma portabilidade de linguagem para programar em GPU. Essa generalidade implica em uma perda de desempenho. As duas interfaces de programação possuem funcionalidades similares, no entanto o desempenho do *kernel* do OpenCL é entre 13% e 64% mais lento que CUDA [7].

No entanto em áreas como segurança computacional, há uma preocupação quanto à utilização do poder computacional das GPUs. Os autores de *malwares* buscam constantemente por novos métodos para evitar a detecção por antivírus. Já existem duas técnicas de código de blindagem que representam desafios significativos para sistemas de análise e detecção de código malicioso, são eles a descompactação e o polimorfismo em tempo de execução [8].

Na Biologia Molecular, as unidades de processamento gráfico, possibilitou a implementação do algoritmo de *Smith-Waterman* que esteve disponível há mais de 25 anos explorando o imenso poder computacional das GPUs [9].

2.4.1. Aglomeração em classes: K-Means

Muitos dos algoritmos de reconhecimento se baseiam em padrões de *pixels*, contudo, uma imagem do mundo real não possui um único valor para cada tom de cor. O espectro

vermelho conta com muitas combinações. Alguns mais escuros que outros, tendendo a outras tonalidades, mas que ainda assim, são agrupados no conjunto que chamamos de vermelho.

Devido a essas variações, é necessário aglomerar as mesmas em uma respectiva classe que as englobem em função dos *pixels* encontrados na imagem. De tal modo, algoritmos em geral serão mais produtivos em seus resultados, excluindo dados que por ventura pudessem servir como obstáculos na implementação.

O K-Means é um algoritmo de aglomeração amplamente aceito pela comunidade científica. Constitui em uma poderosa ferramenta para tratamento prévio de imagens que serão processadas com a finalidade de obtenção de informações específicas como orientação, angulações, detecções em geral, localização [10].

O código se baseia seguindo em etapas como podemos ver na Figura 2.12. A inicialização de classes com *pixels* centrais, utilizando os primeiros elementos da imagem ou definindo-os aleatoriamente. Todos os pontos da imagem são testados verificando sua proximidade de cada classe através da distância euclidiana em cubo de cores tridimensional. O *pixel* é agrupado e o algoritmo se repete até que os *pixels* não mudem de classes ou após um número de *loops* pré-determinado. Ao final os *pixels* da imagem são substituídos pelos centros de seus agrupamentos.

Por utilizar *loops* com altos números de repetições, o mesmo requer alta capacidade processamento, o que o faz ser alvo de processamento paralelo, obtendo os resultados esperados com maior velocidade, garantindo uma boa execução no setor da robótica, ambiente ideal para desenvolvimento do OpenCL.

O algoritmo K-Means foi implementado e as análises foram realizadas utilizando como dispositivo GPU uma nVidia GeForce 8400 GS, como *host* um Intel Quad Core 2 Quad Q8400 e a imagem original, exibida na Figura 2.13, possuindo 256 *pixels* de largura e 192 *pixels* de altura.

A Figura 2.14 abaixo mostra os resultados obtidos em um algoritmo para o *framework* realizado em uma imagem totalizando 49152 *pixels*, em número de *loops* da implementação e segundos para executá-la, desconsiderando os 4 segundos iniciais para compilar as funções no início do *software*.

A inicialização das classes no algoritmo foi gerada aleatoriamente, devido a isso, até certa quantidade de *loops*, os resultados obtidos poderiam ser diferentes. Em média de 100 repetições o algoritmo já se mostrou apto a definir classes íntegras como pode se observar na Figura 2.13. O desempenho do teste mostrou a eficácia da utilização do OpenCL para o desenvolvimento de sistemas de visão computacional ou partes de seus algoritmos.

2.4.2. Detecção de Borda: Filtro Sobel

Uma borda em uma imagem possui uma característica relevante, como uma mudança no nível de intensidades dos *pixels*. Mecanismos detectores de bordas são elaborados a fim de encontrar *pixels* onde ocorre esse tipo de variação. Dessa forma quando esses *pixels* estão próximos podem ser destacados formando uma borda ou contorno [12].

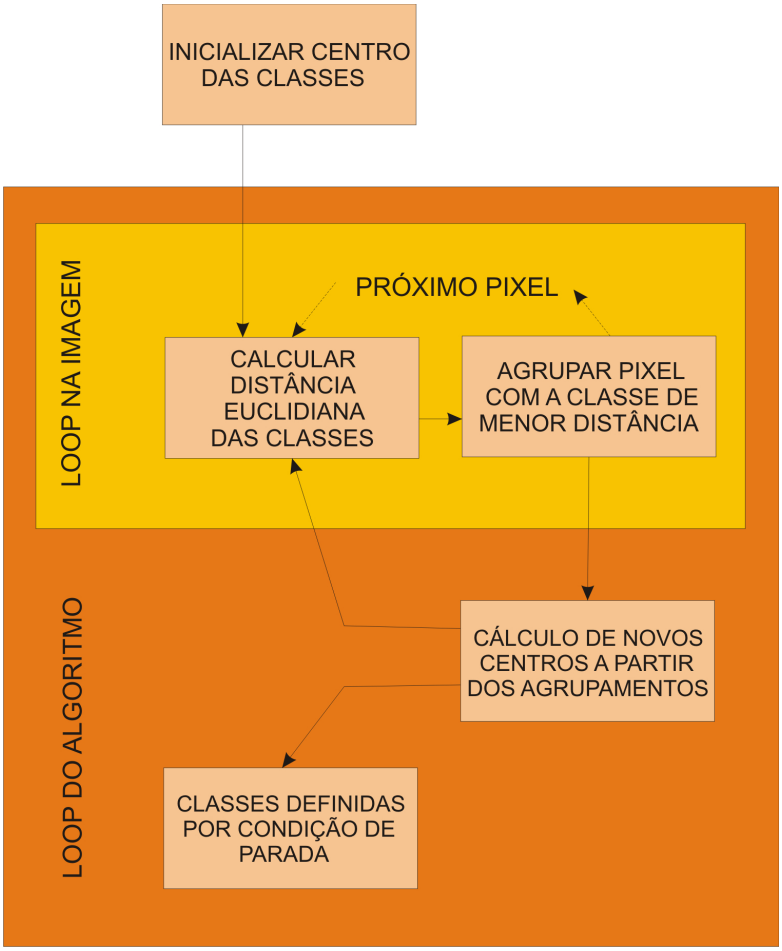
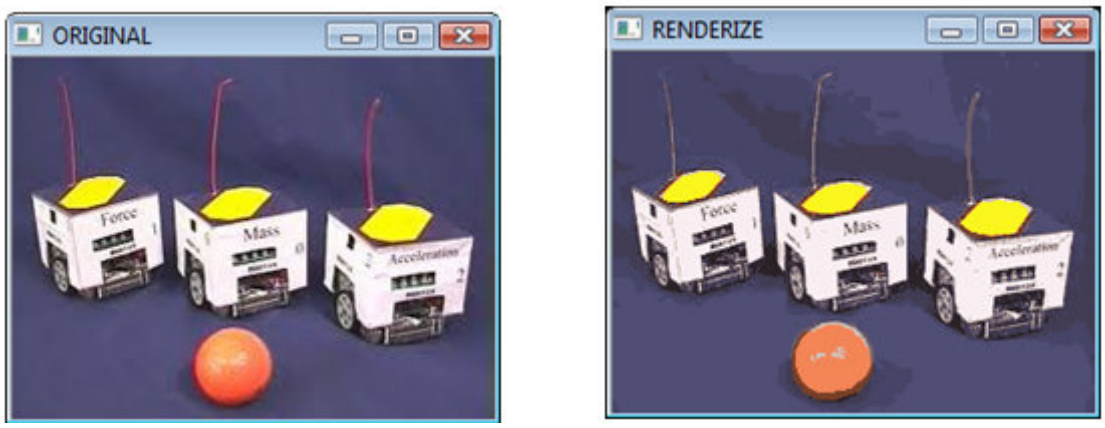


Figura 2.12. Algoritmo K-Means.

Alguns detectores de borda utilizam o cálculo da primeira derivada e/ou segunda derivada para encontrar essas mudanças na intensidade dos *pixels*. Esse cálculo consiste em se definir máscaras de que caracterizam essa variação e em seguida fazer a covolução da imagem pela máscara.

Filtrar uma imagem por convolução é um processo que depende de uma vizinhança de influência do *pixel* que está sendo considerado. Esta operação possui a seguinte idéia: é necessário uma máscara com dimensões MxN, onde cada posição da máscara possui seu valor. Em seguida, a máscara é centrada sobre a imagem na posição (x, y), e cada posição da máscara é multiplicado pelo *pixel* correspondente. Para finalizar calcula-se o somatório dos valores obtidos na multiplicação e substitui na imagem o valor na posição (x, y) por este resultado. Esta operação é representada na Figura 2.15

Os operadores mais comuns baseadas no cálculo da derivada primeira de uma imagem são os operadores de Prewitt e de Sobel [12], a Figura 2.16 mostra as máscaras relacionadas ao operador para detecção de bordas Sobel na horizontal e na vertical e a expressão matemática que o representa, Equação 1. Para determinar bordas nas diagonais basta rotacionar a máscara do ângulo desejado.



(a) Imagem original (b) Imagem resultante

Figura 2.13. Resultados para o algoritmo K-Means.

$$
\begin{aligned}
g(x) &= \frac{\partial f(x,y)}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \\
g(y) &= \frac{\partial f(x,y)}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)
\end{aligned}
\tag{1}
$$

Em uma imagem com resoluções de 800x640 *pixels*, omitindo-se os *pixels* das bordas, para cada *pixel* serão feitas as multiplicações e o somatório. Calcular Sobel para cada um desses *pixels* requer um custo computacional elevado. Em imagens com resoluções maiores, o custo aumenta. É possível também detectar bordas em vídeos, onde é necessário aplicar o operador de Sobel para cada frame do vídeo.

Ao filtrar-se uma imagem com Sobel, em cada *pixel*, os cálculos de multiplicação e adição são independentes de cálculos anteriores. Paralelizar este algoritmo é bem viável, visto que é possível repassar para cada núcleo da GPU a responsabilidade pelo cálculo da primeira derivada de um *pixel* na imagem. Com isso serão computados de vários *pixels* simultaneamente, como ilustrado na Figura 2.17.

2.4.3. Detecção de Padrão: Descritor SIFT

O descritor SIFT (*Scale Invariant Feature Transform*) consiste de um histograma baseado na magnitude e orientação do gradiente de todos os *pixels* em uma região ao redor do ponto de interesse [11]. Desta maneira, tal descritor é capaz de estabelecer uma medida de correlação. Descritores são vetores de características de porções da imagem que podem ser utilizados para comparar porções em imagens diferentes de uma mesma cena. O descritor SIFT é um descritor local altamente distinto, e invariante a mudanças na iluminação e ponto de vista 3D.

O primeiro passo no cálculo do descritor é computar a orientação e magnitude do gradiente da imagem de todos os pontos ao redor do ponto de interesse. Com o objetivo de obter um descritor invariante a orientação, as coordenadas do descritor e as orientações dos gradientes são rotacionadas em relação à orientação do ponto de interesse, ou seja, a orientação do ponto de interesse deve ser a orientação de referência igual a 0^o.

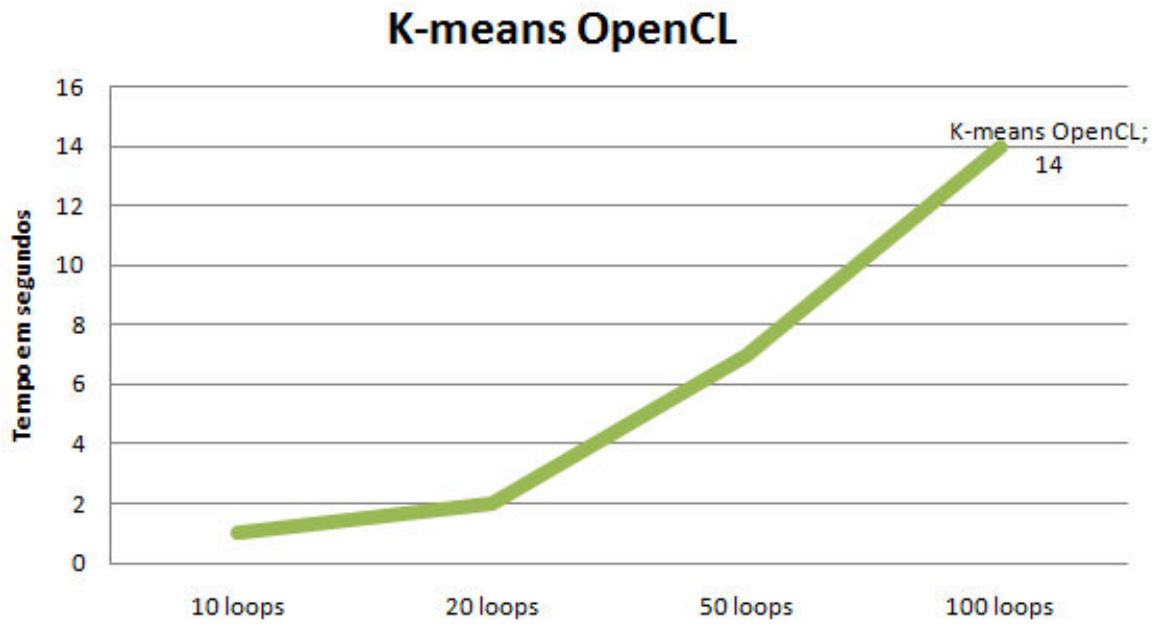


Figura 2.14. Desempenho em segundos para quantidades de *loops*.

Em seguida, cada magnitude calculada em torno do ponto de interesse é atenuada segundo uma janela gaussiana com variância s, igual à metade da largura da região ao redor do ponto de interesse. O próximo passo consiste em definir n x n sub-regiões de k x k *pixels* cada. A partir daí, as magnitudes atenuadas são acumuladas em um histograma de orientações para cada sub-região. A magnitude acumulada em cada entrada do histograma de orientações corresponde à soma das magnitudes dos gradientes próximos à direção daquela entrada. O procedimento de construção do descritor é ilustrado na figura 2.18.

O descritor é representado por um vetor de 32 elementos (2x2x8). O comprimento de cada seta na figura do lado direito corresponde à soma das magnitudes dos gradientes próximos àquela direção dentro de cada sub-região.

O descritor é então representado por todos os histogramas das sub-regiões. Esta representação é na forma de um vetor, onde cada elemento do vetor corresponde a cada uma das direções de cada um dos histogramas. Por fim, para que o descritor seja invariante à iluminação uma normalização é aplicada de tal forma que o valor máximo do vetor de características seja igual a 1.

2.5. Conclusão

As GPUs possuem ampla utilização, no entanto para utilizá-la era necessário muito conhecimento sobre o assunto. O que dificultava o aproveitamento das GPU em algoritmos computacionais. Com o surgimento de ferramentas de programação, como a arquitetura CUDA e o *framework* OpenCL, tornou mais fácil o uso das unidades de processamento gráfico. Visto que unificaram as técnicas utilizadas e no caso do OpenCL, unificou também as arquiteruas. O CUDA e o OpenCL são ferramentas poderosas e ainda possuem um grande potencial a ser desenvolvido.

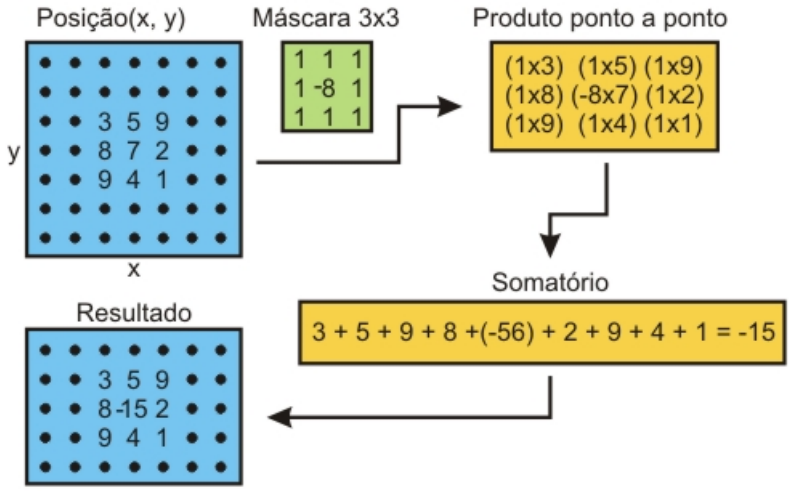


Figura 2.15. Esquema da operação de correlação. No topo à esquerda uma imagem, ao centro uma máscara. Abaixo o resultado do somatório das multiplicações dos *pixels* pela máscara.

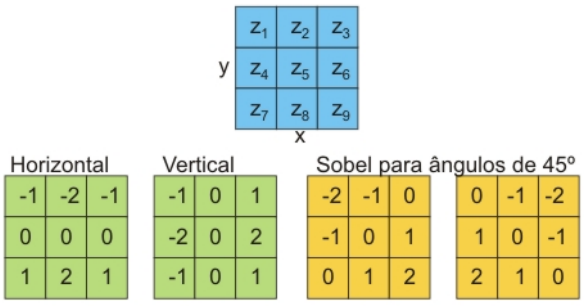


Figura 2.16. Em cima uma máscara genérica com as posições dos valores utilizados nas equações de Sobel. Abaixo máscaras do operador de Sobel.

Essas ferramentas desenvolvem-se a cada dia. A nVidia investe bastante no CUDA aumentando cada vez mais o seu potencial, tornando a araquitetura mais eficiente e segura. Ao mesmo tempo que a nVidia juntamente com a AMD, a Intel e a Apple, dão um forte suporte ao OpenCL. Juntas contribuem para tornar o *framework* melhor e mais robusto. No momento, OpenCL, por ser genérico, é menos eficiente que o CUDA. Contudo, o OpenCL é portátil quanto a arquitetura utilizada, sem a necessidade de reescrever os códigos fontes ao mudar de plataforma. Com o desenvolvimento do *framework*, o OpenCL pode se tornar tão eficiente quanto CUDA.

Na visão computacional, o uso das GPUs se mostrou muito eficiente. Processamento de imagens requer alto poder computacional, principalmente quando há necessidade de processar os dados em tempo-real. Em imagens de alta resolução, a execução paralela de cálculos de derivadas e K-Means em GPUs possibilita a obtenção de resultados em tempo-real. Isso se explica pelo fato destes algoritmos serem amplamente paralelos, ou seja, cada cálculo pouco depende dos anteriores. O processamento em tempo-real pode ser utilizado aplicações onde informações são retiradas de um *stream* de video, e neste caso é necessário computar os frames do video. Se o processo for lento, haverá

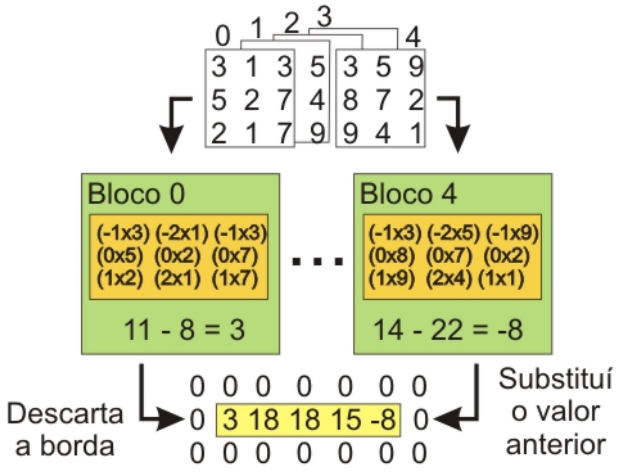


Figura 2.17. Cálculo da primeira derivada utilizando GPU.

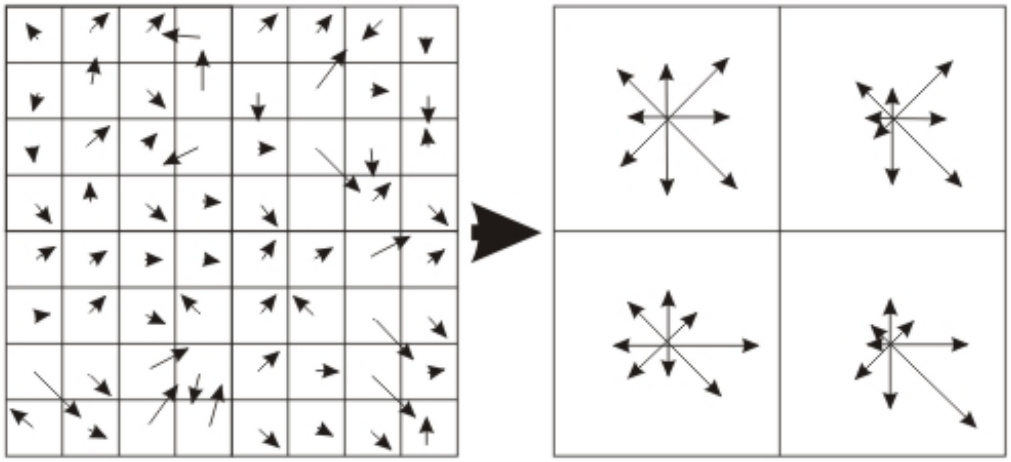


Figura 2.18. Descritor SIFT calculado a partir de um conjunto de 8x8 *pixels* divididos em 2x2 sub-regiões. Cada histograma do descritor possui 8 intervalos de orientação.

perda ou incosistência da informação.

Neste trabalho abordamos algumas técnicas utilizadas na visão computacional e suas aplicações. Também foi abordado, duas fortes ferramentas para progamação paralela em GPUs, o CUDA e OpenCL tentando explicar o funcionamento delas. Buscamos demonstrar a contribuição das GPUs na visão computacional, bem como seu desempenho devido à paralelização dos algoritmos utilizados. E assim contribuir para a continuidade e o desenvolvimento dos algoritmos aqui expostos, divulgando a programção paralela em GPUs e o seu uso na visão computacional.

Em fato, existem ainda outros algoritmos a serem implementados e analisados, dando continuidade ao trabalho aqui exposto.

Referências

[1] NVIDIA, OpenCL Programming Guide for the CUDA Architecture Version 2.3. 2010, NVIDIA Corporation: Santa Clara, Califórnia.

[2] http://gemma.apple.com/technologies/mac/snowleopard/opencl.html

[3] APPLE, Taking the graphics processor beyond graphics. June 2009, Apple Inc.

[4] http://www.khronos.org/opencl/.

[5] KHRONOS, Introduction and Overview. June 2010, Khronos Group.

[6] AMD, ATI Stream SDK OpenCL Programming Guide. June 2010. Advanced Micro Devices Inc.

[7] KARIMI, Kamran; DICKSON, Neil G.; HAMZE, Firas. A Performance Comparison of CUDA e OpenCL 2010, D-Wave Systems Inc.: Canada.

[8] VASILIADIS, Giorgos; POLYCHRONAKIS, Michalis; IOANNIDIS Sotiris; GPU Assisted Malware With CUDA and OpenCL.

[9] Manavski, S.A.; VALLE, Giorgio; CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. Março 2008.

[10] AIRES, Kelson R.T.; ALSINA, Pablo J.; MEDEIROS, Adelardo A.D. A Global Vision System for Mobile Mini-Robots. Maio, 2001.

[11] AIRES, Kelson R.T.; Segmentação de Planos Baseada em Homografia Afim, Fluxo Óptico e Reconstrução Métrica. Outubro, 2009. Natal-RN.

[12] MARENGONI, Maurício; STRINGHINI, Denise. Introdução à Visão Computacional usando OpenCV. RITA 2009.

Capítulo 3

Joomla 1.6

Alexandre Marques Albano da Silveira,
Andréa Carneiro Linhares,
Clemilson Costa Santos

Abstract

Publicity is very important for the recognition of any work done. With the expansion of the Internet, the use of websites has become a powerful ally in the products and brands promotion. It is in this context that emerges Joomla. Joomla is a free website devellopment tool widely used and secure. You do not need programming skills to use it and maintain it. In fact, Joomla is no more complicated than a text editor. It's flexible and customizable. Thus, it can be used to develop both small sites for scholarly works and large portals for document management.

Resumo

Propaganda é de fundamental importância para o reconhecimento de qualquer trabalho realizado. Com o avanço da Internet, a utilização de websites tornou-se um aliado poderoso na divulgação de produtos e marcas. É nesse cenário que surge o Joomla. O Joomla é uma ferramenta livre para criação de websites, muito utilizada e segura. Não é necessário conhecimento em programação para usá-lo e mantê-lo. Na verdade, O Joomla não é mais complicado que um editor de texto. É totalmente flexível e customizável. Desse modo, ele pode ser usado tanto no desenvolvimento de pequenos sites para trabalhos acadêmicos, quanto em grandes portais para gerenciamento de documentos.

Introdução

Propaganda é, como dizem, "a alma do negócio", sendo de fundamental importância para o reconhecimento de qualquer trabalho realizado. Existem várias formas de propaganda, como: boca a boca, em que clientes satisfeitos espalham que seu serviço ou produto é bom; panfletos, mas esse está em declínio e é mal visto devido à conscientização ecológica; divulgação em meios de comunicação, como rádio, televisão e Internet.

A Internet é indiscutivelmente o maior e mais democrático meio de comunicação. Nela se pode postar de tudo, desde conteúdos imorais a histórias de assombração. As empresas já se deram conta disso e a usam como o principal meio de propaganda, através de hotsites, banner em blogs, anúncios em redes sociais, spam e qualquer outra forma em que apareça o produto ou a marca. Mas, é indispensável ter-se um site bem estruturado, organizado, com conteúdo de qualidade e informação clara.

Um site deixa sua marca e produtos visíveis em tempo integral e em qualquer lugar do mundo, o que facilita bastante o acesso do consumidor e a descoberta do que sua empresa ou grupo tem a oferecer. Além disso, o endereço do site pode ser divulgado em outros meios, deixando que a curiosidade guie o internauta ao seu sítio. É nesse cenário que surge o Joomla. O Joomla é uma ferramenta capaz de criar sites de qualidade, de maneira rápida, permitindo ao usuário concentrar-se na elaboração do conteúdo sem preocupar-se com muitos detalhes técnicos.

Antes de aprendermos a usar o Joomla, é necessário apresentar alguns conceitos fundamentais sobre o mesmo. Para iniciarmos nosso aprendizado da ferramenta, precisamos saber o que ela é, para que serve, quais recursos estão disponíveis e em que casos usar. Desse modo, os seguintes tópicos serão abordados:

- O que é Joomla;
- Conceito de *Content Management System* (CMS);
- Explicação sucinta das ferramentas necessárias ao Joomla;
- Como instalar e configurar o Joomla;
- Algumas definições;
- Como criar menu, weblink, grupos de usuário e nível de acesso;
- Exemplo de criação de conteúdo e gerência de acesso;
- Novidades da versão 1.6;

3.1. O que é Joomla

O nome Joomla (pronuncia-se *djumla*), tem origem de uma palavra Swahili, *Jumla*, que significa "todos juntos". Joomla, software livre licenciado sobre a GNU/GPL, é um *Content Management System* (ver seção 4.2) derivado do Mambo, escrito em PHP e executado em servidor Apache. Usa banco de dados MySQL, sendo válido ressaltar que a interação com o banco já se encontra implementada. Outra característica importante é que o sistema de desenvolvimento do Joomla é baseado em componentes, onde determinado componente tem uma finalidade específica. Existem muitos componentes já prontos, tanto gratuitos como comerciais, que podem ser adaptados a diversas aplicações. Dentre esses componentes, podemos citar: galeria de fotos, galeria de vídeos, menus com efeitos diversos, interação com redes sociais, nuvem de palavras, controle de formulários, entre outros. Os componentes também podem ser comprados ou feitos por você mesmo, caso

saiba PHP. Tem-se ainda a possibilidade de modificar o código dos componentes existentes. Essa flexibilidade faz do Joomla uma boa ferramenta para desenvolvimento de diversos tipos de sites.

Figura 3.1. Logo do Joomla

3.2. O que é um *Content Management System* (CMS)

Content Management System significa Sistema de Gestão de Conteúdo ou Sistema de Gerenciamento de Conteúdo. Um CMS possui ferramentas necessárias para manter portais e websites em funcionamento sem a necessidade de um profissional qualificado na área (*webmaster*). Nesses sistemas, já encontram-se implementadas funções de incluir, editar, publicar e apagar textos. Além disso, ele possibilita organizar o site por tipo de conteúdo e grupo de permissões. Desse modo, é possível definir quais usuários terão acesso a determinado tipo de artigo e suas permissões (controle total, leitura ou sem acesso).

Um CMS possibilita que uma empresa ou grupo de trabalho possua um portal onde os membros poderão postar seus trabalhos diretamente, sem a necessidade de enviá-los, primeiramente, para um setor específico que dará sequência à sua publicação. Isso implica numa redução de custos e corrobora no aumento da velocidade das postagens, já que não vai depender de um *webmaster* para isso.

Um CMS pode ser usado para a criação de *blogs*, *websites*, ambientes de ensino a distância e portais. Como características destacam-se a disponibilização prévia de uma estrutura do site (esqueleto) e a facilidade de gerenciar o que vai ser publicado, não sendo mais difícil de operar que um editor de texto ou uma planilha.

3.3. Vantagens em se usar Joomla

Algumas vantagens podem ser evidenciadas no uso de Joomla preferencialmente a uma outra ferramenta para desenvolvimento de sites:

- **Reduzido conhecimento técnico necessário**. O Joomla não elimina a necessidade de conhecimento técnico, mas ele é importante apenas nas etapas iniciais, como instalação e configuração. Além disso, o Joomla é de fácil aprendizado;
- **Não é preciso reinventar a roda.** No Joomla, as tarefas básicas de um site já estão prontas e não é preciso refazê-las, nem iniciar o site do ponto zero;

- **Comunidade atuante e com grande número de seguidores.** O Joomla é muito usado e sua comunidade vem crescendo rapidamente, o que é de grande reconforto aos usuários. O projeto tem amadurecido e agregado novas funcionalidades, provendo sua constante atualização em relação às tecnologias Web e CMS. Além disso, a existência dessa comunidade atuante facilita a obtenção de ajuda por parte dos usuários;

- **Grande disponibilidade de módulos e extensões**. O Joomla possui as funcionalidades básicas de um site ou portal, mas existem muitos módulos e extensões disponíveis voltados a aplicações mais específicas;

- **É software livre.** O Joomla é licenciado pela GNU/GPL. Não é preciso pagar nada e nem piratear o software para usar o Joomla como base do seu website. Além disso, o código fonte está aberto para ser alterado de acordo com suas exigências;

- **Desprendimento com o desenvolvedor.** Um site feito em Joomla é fácil de manter e evoluir, e por ser um padrão, qualquer profissional que trabalha com a ferramenta não terá dificuldades em gerenciar o site desenvolvido por outra pessoa;

- **Segurança.** O Joomla é implementado seguindo padrões de segurança inclusos na ferramenta. Além dessa segurança padrão, existem *plugins* que podem se adicionados com a finalidade de deixar o site ainda mais seguro.

- ***Templates*.** Para mudar o *layout* do site, basta modificar ou editar o *template* do mesmo, o qual pode ser baixado, editado ou comprado. O Joomla segue o padrão MVC (*Model View Controller*), que divide seu programa em camadas e torna mais fácil a inserção de mudanças no mesmo;

- ***Frontend* traduzida em vários idiomas (ver seção 4.6).** Seu site fornece suporte a diversos idiomas. Assim, é possível que usuários de diferentes lugares do mundo acessem o mesmo site no seu respectivo idioma pátrio;

- **Adequado a dispositivos móveis.** O Joomla pode ser acessado de dispositivos móveis, tornando seu site portável e mais interativo.

3.4. O que é preciso para usar Joomla

De modo geral, as seguintes ferramentas são necessárias para se usar o Joomla: PHP, MySQL e Apache. Cada uma delas será descrita a seguir:

- **PHP** (um acrônimo recursivo para PHP: *Hypertext Preprocessor*). O PHP é uma linguagem de *script open source*, de uso geral, muito utilizada e especialmente guarnecida para o desenvolvimento de aplicações Web embutíveis dentro do HTML. É preciso dispor do interpretador do PHP para utilizar o Joomla;

- **MySQL**. O programa MySQL (R) é um servidor robusto de bancos de dados SQL (*Structured Query Language* - Linguagem Estruturada para Pesquisas), muito rápido, multi-tarefa e multi-usuário. O Servidor MySQL pode ser usado em sistemas de produção com alta carga e missão crítica, bem como pode ser embutido em

programas de uso em massa. MySQL é uma marca registrada da MySQL AB. O programa MySQL é de licença dupla. Os usuários podem escolher entre usar o programa MySQL como um produto *Open Source/Free Software* sob os termos da GNU *General Public License* (http://www.fsf.org/licenses/) ou podem comprar uma licença comercial padrão da MySQL AB;

- **Apache**. O servidor Apache (ou Servidor HTTP Apache. Em inglês: *Apache HTTP Server*, ou simplesmente: Apache) é o mais bem sucedido servidor web livre. Foi criado em 1995 por Rob McCool, então funcionário do NCSA (*National Center for Supercomputing Applications*). Numa pesquisa realizada em dezembro de 2007, foi constatada que a utilização do Apache representa 47.20% dos servidores ativos no mundo. Em maio de 2010, o Apache serviu mais de 54,68% de todos os sites existentes e mais de 66% dos milhões de sites mais movimentados. É a principal tecnologia da *Apache Software Foundation*, responsável por mais de uma dezena de projetos envolvendo tecnologias de transmissão via web, processamento de dados e execução de aplicativos distribuídos.

3.5. Instalação do Joomla

O *download* do Joomla pode ser feito em: **http://www.joomla.org/download**. O ideal é sempre optar pela última versão estável. Além de usar o Joomla diretamente em uma hospedagem na Internet, podemos instalá-lo em servidores locais apenas assegurando os pré-requisitos citados no tópico 4.4. Para isso, deve-se descompactar os arquivos e colocá-los na pasta que o servidor cria automaticamente para tal fim. O envio dos arquivos do Joomla ao servidor deve ser realizado através do protocolo FTP (*File Transfer Protocol*, Protocolo de Transferência de Arquivos). Um cliente de ftp muito popular e livre é o filezilla, que pode ser obtido em: http://filezilla-project.org/. Quando os arquivos estiverem no local correto, a interface de instalação do Joomla poderá ser acessada pelo browser.

A instalação é dividida em sete passos:

1. Escolha do idioma;
2. Verificação da instalação;
3. Licença;
4. Banco de dados;
5. Configuração de FTP;
6. Configurações gerais;
7. Finalização;

O primeiro passo consiste na escolha do idioma. Caso não esteja disponível o idioma de sua preferência, a seção 4.8 irá descrever como adicionar pacotes de idiomas e modificá-lo. No segundo passo o Joomla verifica se está tudo correto e mostra uma lista de verificação. É importante analisar atentamente as configurações realizadas e assim

evitar erros comuns, como por exemplo, não configurar adequadamente as permissões necessárias à pasta em que se encontra o Joomla. Será destacado em vermelho qualquer problema identificado pelo Joomla e após corrigi-los, deve-se clicar em *Check Again* e verificar se o problema foi realmente sanado.

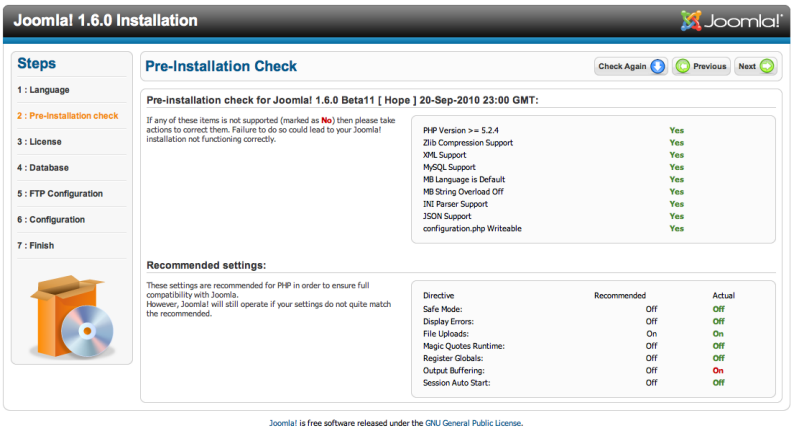


Figura 3.2. Verificação da instalação

No passo seguinte, terceiro, aparecerá uma tela explicando os termos da licença de uso do Joomla, a *GNU General Public License*. Feita a leitura, pode-se clicar em *Next*. O passo quatro consiste na configuração do acesso ao banco de dados. Devem ser fornecidos os dados que a hospedagem do site forneceu ou os dados obtidos a partir da instalação do Apache. Alguns servidores permitem que se tenha mais de um banco de dados configurado. Nesse caso, é necessário criá-los e fornecer os dados ao Joomla. Em caso de servidores locais, o Joomla geralmente tem permissão para criar o banco. Assim, resta fornecer um nome que o próprio Joomla poderá criá-lo.

O passo cinco corresponde à configuração do acesso por ftp, com as informações fornecidas pela hospedagem do site (mesmas configurações usadas para montar o site no servidor). No passo seis, deve-se entrar com o nome do site e dados pessoais. Deve-se ter especial atenção com a senha, pois a mesma será de administrador. É válido ressaltar que na versão 1.5 não era usado o login do administrador. Entrava-se com a senha e o usuário criado era denominado **admin**. Isso gerava um inconveniente, pois era preciso mudar o nome do usuário para garantir a segurança do site. Existe uma opção disponível para instalar exemplo de conteúdo. Essa opção cria um exemplo de site com conteúdo, categorias, artigos, menus, etc. Pessoalmente, prefiro não marcar essa opção, pois será necessário apagar tudo depois. Contudo, para quem está querendo aprender a usar a ferramenta, selecionar essa opção pode vir a ser interessante, pois ela deixará tudo previamente configurado.

No último passo, é preciso remover o diretório *installation*. Esse diretório encontra-se na mesma pasta onde foram copiados os arquivos do Joomla. Após removê-lo, a instalação está concluída e será possível visualizar o site criado. A área administrativa é acessada incluindo "/administrator"no final do endereço http. Ex: Seja **www.nome-do-site.com** o nome do site. Para se ter acesso à área de administração, digita-se **www.nome-do-site.com/administrator**.

3.6. Frontend/Backend

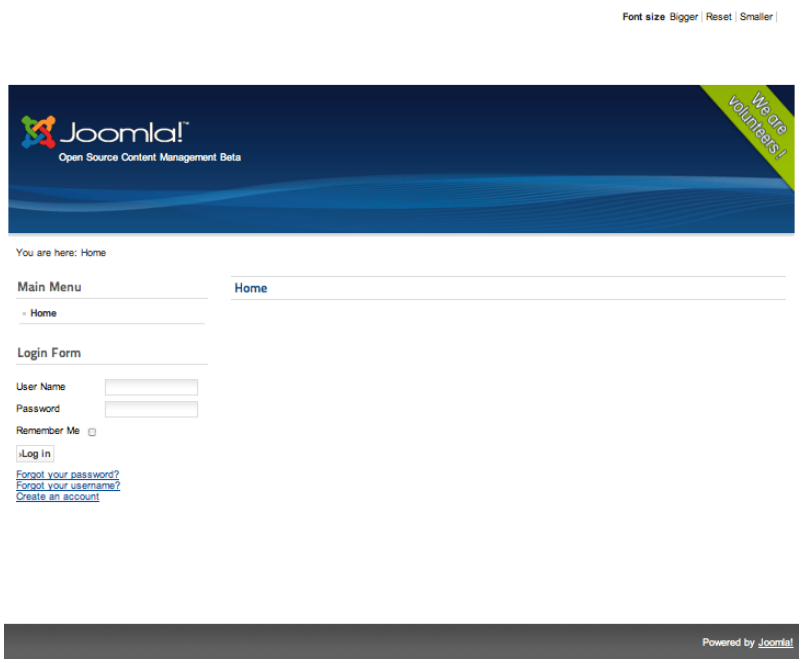


Figura 3.3. *Frontend* do Joomla

O Joomla tem duas faces/áreas: uma para os usuários padrão, chamada *frontend* (figura 4.3) e outra administrativa, denominada de *backend* (figura 4.4). A *frontend* é onde aparece o site em si. Nela é possível limitar o acesso aos dados do site configurando grupos e níveis de acesso, de modo a disponibilizar conteúdos diferentes no mesmo site, com base no perfil de um dado usuário. Na *backend* é preciso, obrigatoriamente, fornecer *User Name* e *Password* para se ter acesso à mesma. Nela gerenciamos tudo do site: artigos, usuários, acesso ao banco de dados e configuração de ftp. Esta também poderá ter acesso diferenciado - o grupo de um usuário define seu nível de acesso.

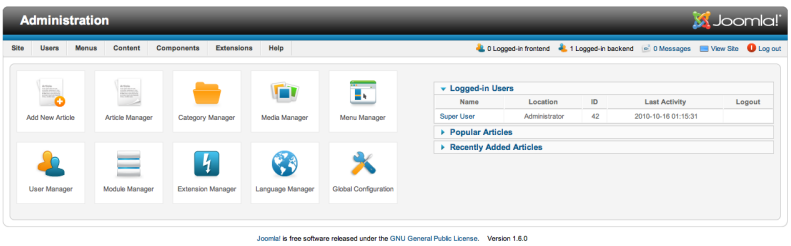


Figura 3.4. *Backend* do Joomla

3.7. *Templates*

O *template* é a aparência do Joomla. Tanto a *frontend* como a *backend* possuem *templates* que podem ser alterados. Esses são independentes do conteúdo do site. Na figura 4.3, podemos visualizar o *template* da *frontend* do Joomla 1.6 (Beez) sem nenhum conteúdo. No canto superior direito do template temos uma opção para aumentar, resetar ou diminuir

o tamanho do site. Ainda na mesma figura, tem-se uma imagem seguida por "onde você está", que indica a hierarquia do diretório onde o usuário se encontra. Na parte esquerda da figura encontramos os menus. Nesse exemplo específico só temos o home, um menu padrão para a página principal do site e logo abaixo o login *form*. Mais à direita (ainda na mesma figura), temos a chamada "área principal", onde aparecem os artigos marcados para serem exibidos na *home*. Na parte inferior da tela temos um rodapé.

As alterações no *template* do *backend* também podem ser efetivadas no mesmo local que as do *frontend*. O *template* padrão da instalação, figura 4.4, é o *bluestork*.

No Joomla tem-se um "Gerenciador de Temas", que pode ser acessado em Extensões->Gerenciador de Temas. Nele podemos visualizar os *templates* instalados, bem como selecionar o padrão e editá-los. Para escolher o *template* padrão, seleciona-se o *template* desejado e clica-se em padrão, ou marca-se o template de sua preferência na coluna padrão. Podem ser observadas duas estrelinhas, cada qual indicando o *template* padrão da *backend* e o da *frontend*. Esse processo é feito com a aba estilos selecionada.

Podemos editar algumas opções básicas do *template* selecionado que se deseja alterar. Clica-se em editar. As opções que aparecem variam em função do *template* utilizado.Também pode-se alterar o código dos arquivos do *template*. Faz-se isso escolhendo a aba "temas"e clicando no nome do template a ser editado. Feito isso, aparecerão os arquivos que o compõe, podendo-se editá-los, entre outros.

Ainda no "Gerenciador de Temas", é possível visualizar as posições programadas para adicionar os módulos na aba "temas". Para que a opção apareça, clica-se em opções, marcando-se "ativado"em Pré-visualização de posições de módulos, seguido de salvar. Feito isso, aparecerá a opção Pré-visualizar abaixo do nome do menu, podendo-se visualizar a distribuição das posições do *template* (figura 4.5).

Nos *templates* são personalizados o estilo do menu, ícones, imagens e os locais dos módulos, como cabeçalho, esquerda, direita, rodapé, dentre outros. Existem muitos *templates* disponíveis e gratuitos, enquanto outros estão disponíveis para venda (o comprador pode inclusive "encomendar"um template). Existem ainda os clubes de *templates*, onde os associados tem direito de baixar o conteúdo produzido pelo clube.

3.8. Joomla no idioma de sua preferência

O Joomla tem suporte a pacote de idiomas, permitindo tradução do conteúdo do site simplesmente através da adição de arquivos do idioma desejado. Esse suporte possibilita que operadores de nacionalidades diferentes possam trabalhar no mesmo site através da escolha do idioma de sua preferência. Na *frontend* esse suporte é bastante relevante, pois quem vai visitar seu site quer encontrá-lo no seu idioma ou pelo menos ter a opção de configurá-lo. Os arquivos de tradução do Joomla são encontrados facilmente na Internet, sendo necessário apenas estar atento à versão escolhida.

Por exemplo, para configurar o Joomla para o português, o usuário deve clicar na opção Extensions e escolher Extension Manager (figura 4.6). Em seguida, o usuário deve clicar em selecionar arquivo, escolher o arquivo de tradução e confirmar a opção de Upload & Install. Aparecerá um aviso em azul indicando que o idioma foi instalado e logo abaixo um quadro com o termo de licença de uso. Alguns componentes requerem

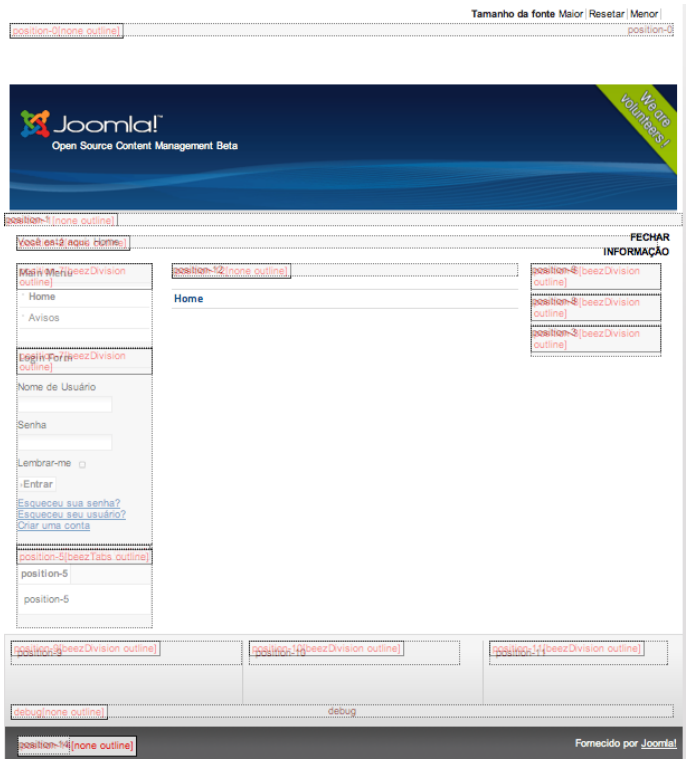


Figura 3.5. Posições no template

uma pequena configuração.

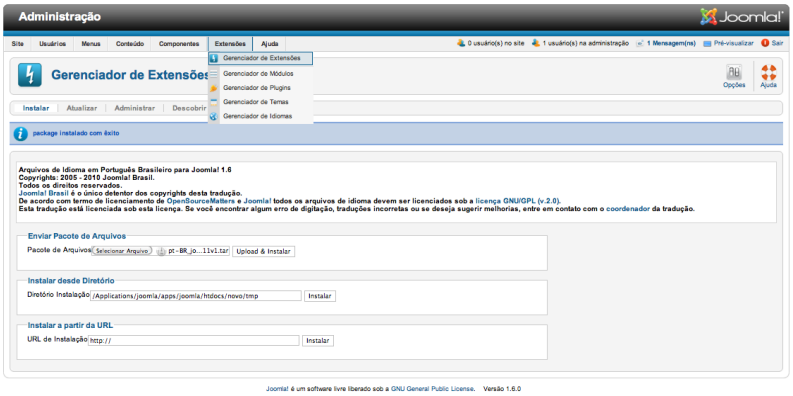


Figura 3.6. Gerenciador de Extensões /Instalar

Também é possível instalar o pacote de idiomas colocando a pasta com os arquivos requeridos dentro do diretório *language*, para adicionar a opção à *frontend*, e dentro do diretório *administrator/language*, para adicionar essa opção à *backend*. Depois de instalado, tem-se que ativar o idioma, definindo-o como padrão para o site. O usuário deve clicar novamente em Extensions e escolha a opção Language Manager. No campo Default deve-se indicar o idioma desejado.A estrelinha amarela indica o idioma padrão. A sequência de passos indicada anteriormente permite traduzir a *frontend*. Para traduzir a *backend*, o usuário deve selecionar Administrator em Filter Location, que fica no lado

direito da janela, em cima das opções, e indicar o idioma desejado da mesma forma que na *frontend*.

Não é necessário que o usuário fique modificando seu idioma "'padrão"' a cada acesso. O usuário escolhe seu idioma preferencial e quando efetuar login, o joomla exibirá automaticamente a opção previamente escolhida.

3.9. Organização do conteúdo

O Joomla 1.6 apresenta apenas categorias, diferente da versão 1.5 que tinha sessão e categoria. Categorias são as divisões do conteúdo onde os artigos são vinculados. Pode-se ter uma categoria dentro de outra, o que implica num maior número de opções de níveis de conteúdo. Essa inovação facilita a organização dos artigos, que são os conteúdos em si, onde ficam os textos e as imagens. Todo artigo deve estar obrigatoriamente vinculado a uma categoria. A versão 1.6 inclui uma categoria para que os artigos possam ser adicionados. Se ela for apagada, será necessário criar outra categoria para poder adicionar artigos.

3.10. Grupos de Usuários

Uma das maiores novidades do Joomla 1.6 é a nova camada de ACL (*Acess Control List* ou Lista de Controde de Acesso), que permite atribuir grupos aos usuários e determinar o que cada grupo pode fazer. Além disso, existe a opção de adicionar ou editar níveis de acesso. Na versão 1.5 existia apenas três níveis de acesso:

- Público: formado pelos usuários que não estavam logados;
- Registrado: usuários que estavam logados no site, mas tinham acesso apenas à *frontend* e não podiam publicar ou editar;
- Especial: formado pelos usuários que eram classificados como autores, editores, *publishers* ou administradores.

Os níveis acima citados ainda existem, mas é possível adicionar mais níveis, editá-los e relacioná-los com os grupos desejados. Fazer um controle de acesso requer planejamento para que tudo saia como esperado. A seguir um exemplo de como controlar a visualização da publicação de conteúdos por cursos. Considere os alunos do curso de Engenharia da Computação. Os mesmos possuem acesso ao conteúdo de "cálculo"e "programação", mas não precisam visualizar artigos de outro curso, como os da Odontologia, por exemplo.

3.10.1. Exemplo de gerência de acesso

Nesse exemplo serão criados dois grupos, "Engenharia da computação"e "Economia", e os níveis de acesso "Cálculo", "Programação", "História"e "Tratamentos dentários". Vincularemos a esses níveis os grupos anteriormente citados de acordo com a afinidade de cada grupo. Desta forma, os alunos podem ser criados como usuários e passam a pertencer ao grupo "Engenharia da Computação"ou "Economia".

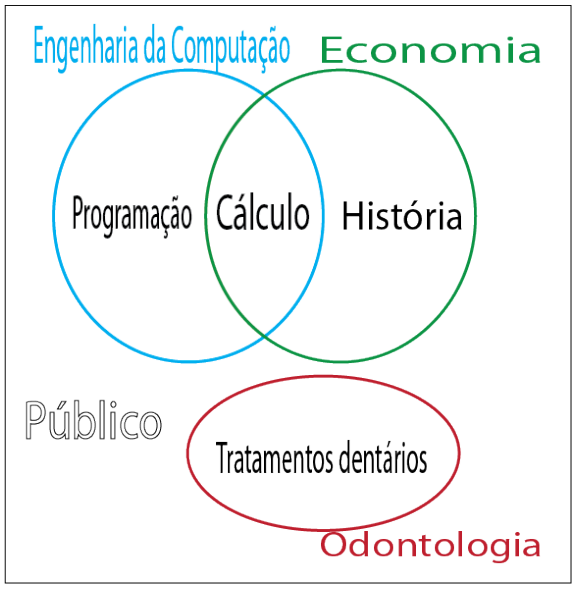


Figura 3.7. Exemplo de grupos de acesso

Para se criar um grupo, deve-se clicar no menu usuários, escolher Gerenciador de Grupos e em seguida a opção Novo. Em título, é necessário preencher o nome do grupo (por exemplo, Engenharia da Computação) e o grupo pai como *registered*. Isso restringirá os alunos apenas a leitura dos artigos. Após preencher os campos, deve-se clicar em Salvar e Fechar.

A criação do nível de acesso é obtido por meio da sequência de opções: Usuários->Gerenciador de Níveis de Acesso e Novo. Em seguida, se fornece um título e no caso do exemplo considerado marca-se Engenharia da Computação para Cálculo e Programação. Finaliza-se o processo com a opção SALVAR. Nos grupos Cálculo e História deve ser selecionado o item economia. Tratamentos dentários não será selecionado, por não ter sido criado um grupo Odontologia.

A Figura 4.7 apresenta a visualização de cada grupo. Integrantes do grupo Engenharia da Computação poderão ver conteúdos postados nos níveis de acesso "Programação"e "Cálculo". Economia visualiza "Cálculo"e "História". "Tratamentos dentários"foi vinculado ao grupo Odontologia, que poderá visualizar esse nível de acesso. Lembrando que todos dos grupos podem visualizar o que está com nível público.

3.10.2. Criando usuários

Para criar usuários pode-se selecionar o atalho "Gerenciador de Usuários"no Painel de Controle ou, de qualquer outro diretório, clicar no menu "Usuários"e selecionar "Gerenciador de Usuários". Em seguida, clicar em "Novo"para adicionar o usuário. Os seguintes campos devem ser preenchidos:

- Nome: nome verdadeiro do Usuário;
- Nome de Usuário: nome usado para fazer login;
- Senha: senha de acesso;

- Confirmar senha;
- E-mail: e-mail do usuário. É importante seu preenchimento, pois o Joomla manda um e-mail de ativação da conta. Contudo, a conta pode ser ativada diretamente pelo administrador;
- Grupos Atribuídos: marcar os grupos aos quais o usuário pertence.

O exemplo de criação de usuários para o caso "aluno de engenharia da computação", neste deve-se fornecer os dados do mesmo e na opção "grupos atribuídos", marcar "Engenharia da Computação". Considera-se ainda a situação em que se deseja criar um usuário "professor de Engenharia da Computação". Este novo usuário além de ler o conteúdo marcado como "Engenharia da Computação, poderá alterá-lo e adicionar novos artigos. Para isso, devemos marcar também o grupo "Editor".

3.11. Criando categorias e artigos com acessos diferenciados

Sabendo que todo artigo deve pertencer a uma categoria, cria-se primeiro as categorias e depois os artigos. Inicialmente, será criada a categoria através do menu de "Conteúdo", acessando o sub-menu "Gerenciando Categorias", e selecionando "Nova Categoria". Considerando o exemplo de uma categoria de Programas em C e outra para programas em Python, cujo nível de acesso será programação. Pode-se ainda tomar a criação de uma categoria programação como categoria pai, a Figura 4.8 apresenta a hierarquia. Para se obter essa estrutura, primeiro cria-se a categoria programação e depois as demais, obedecendo os níveis parentais. Abaixo uma exposição dos campos a serem preenchidos.

- Título: título da categoria;
- Codinome: nome que será usado para indexar a categoria (não deve conter caracteres especiais). Esse campo pode ser deixado em branco, passando a ser preenchido automaticamente;
- Pai: usado quando se deseja que a categoria esteja vinculada à uma pai, assim pertencerá a categoria pai, ou seja, estará contida na pai;
- Estado: pode ser publicado, despublicado, arquivado ou lixeira;
- Nível de acesso: definir um nível de acesso;
- Idioma: qual o idioma da categoria;
- Descrição: nesse campo é descrito quais os tipos de artigos contidos na categoria, ou caso seja utilizada como menu, conter uma descrição no nome;

A Figura 1.8 apresenta nas opções à direita (ver figura 4.9) as permissões disponíveis na Categoria. Entretanto, mesmo escolhendo o nível de acesso anteriormente configurado, vinculando os grupos, pode-se ainda alterar neste momento o acesso à categoria, definindo os usuários que podem criar, remover e editar, tornando-o assim, ainda

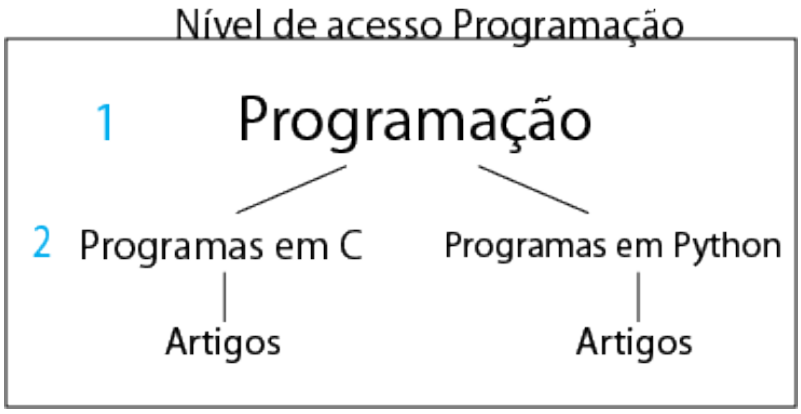


Figura 3.8. Hierarquia de categorias do exemplo

mais flexível o controle das categorias. Esta opção é uma característica nova encontrada na versão 1.6.

Retomando o exemplo, cria-se uma categoria com nome "Programação", sem a categoria pai, estado publicado e nível de acesso Programação(Primeiro nível da árvore da Figura 4.8. Clicando em Salvar e Novo é criado neste momento "programas em C"e "programas em Python", a essas serão adicionadas a categoria "programação"como categoria pai.

A partir das categorias criadas, pode-se gerar artigos. A sequência de passos inicia-se em Conteúdo, seguido por Gerenciador de Artigos->Novo Artigo. Observa-se que os artigos podem ser configurados de forma independente, bastando escolher o nível de acesso e as permissões adequadas. Na criação do artigo é necessário escolher a qual categoria será vinculado. A tela de criação do artigo é bem semelhante a da sessão, contendo de novidade uma aba de configuração denominada "Opções de Artigo", o que permite configurar particularidades tais como: o nome do artigo como link, uma opção para salvá-lo como pdf, ou imprimir, existem várias outras, que por serem bem intuitivas não integra-se aos objetivos gerais deste curso.

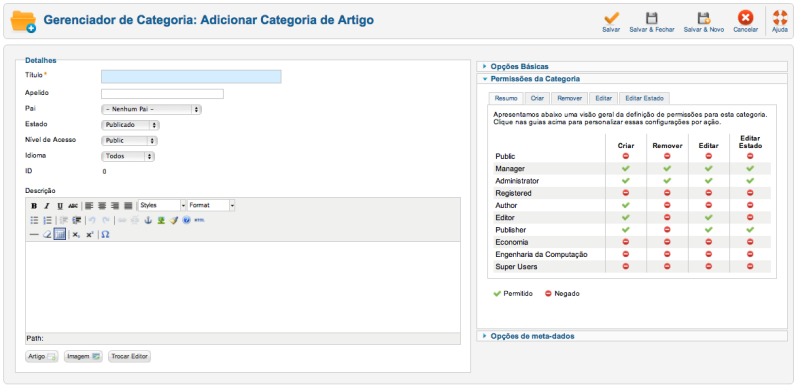


Figura 3.9. Tela de criação de categoria

Pode-se notar que esse controle individual de permissão de artigo e sessão possibilita construir acessos diferenciados para alguns grupos, até os que estejam vinculados

ao mesmo nível de acesso. Um controle de acesso com vários níveis, pode atingir certa complexidade, dificuldade que poderá ser amenizada com uma dedicação especial em um pré-projeto de controle de acesso.

3.11.1. Estados dos Artigos

Existem quatro estados para os artigos, bem como para os *weblinks*, *banner* ou outro conteúdo. Os artigos podem está publicados, despublicados, arquivados ou na lixeira. Os publicados aparecem normalmente na *frontend*, os despublicados, não aparecem. Os arquivados não aparecem normalmente na *frontend*, mas podem ter menus especiais que os exibem. Os artigos apagados vão para lixeixa, podendo posteriormente ser restaurado ou apagado definitivamente. Outro atributo de um artigo é poder está ou não em destaque, os sites em destaque aparecem na página principal, sendo mais facilmente visualizados, e existem módulos para listar artigos em destaque.

3.12. Enviando imagens

As imagens, a serem adicionadas ao site, devem ser enviadas ao servidor. O ambiente Joomla possui implementado um gerenciador de mídias, que executa um controle completo, exibindo os arquivos presentes no diretório, definindo quais arquivos poderão ser upados e o tamanho máximo de cada arquivo.

O Gerenciador tem um atalho no painel de controle que também pode ser encontrado através da sequência: Conteúdo->Gerenciador de Mídia. Os campos selecionados na Figura 4.10 indicam a sequência necessária para a criação de uma pasta onde deverá ser organizada a mídia utilizada. Deve-se inserir um nome no campo "Criar Pasta"e confirmar clicando. Posteriormente, a pasta é selecionada, para que o arquivo enviado fique embutido nela. Clicando em "selecionar arquivo"é possível fazer a escolha da imagem a ser exposta no site. Após as medidas anteriores deve-se clicar em Iniciar Envio. Realizada a sequência de passos a imagem desejada aparecerá dentro da pasta corrente.

3.13. Weblinks

O ambiente Joomla possui um gerenciamento padrão para weblinks, tratando-os de forma similar aos artigos, com opções de categorias dotadas de hierarquia e gerenciamento de nível de acesso, podendo ainda, ser personalizado na própria classe. Criar uma categoria para Weblink é muito semelhante a criação de categorias para artigos. A criação de uma categoria pode ser feita a partir da seguinte sequência de passos: Clicar em Componentes->weblinks->categorias e por fim, novo. O processo de criação do weblink também é semelhante ao do artigo e segue a mesma sequência de passos anteriormente descrita.

3.14. Menu

Nas sessões 4.11 e 4.13 foram apresentadas a criação de categorias, artigos e weblinks, porém esses não aparecem no exemplo do site. Certamente, nem tudo que é criado na *backend* deve ser exibido na *frontend*, sendo ainda necessário organizar a exibição de acordo com o *layout* do site.

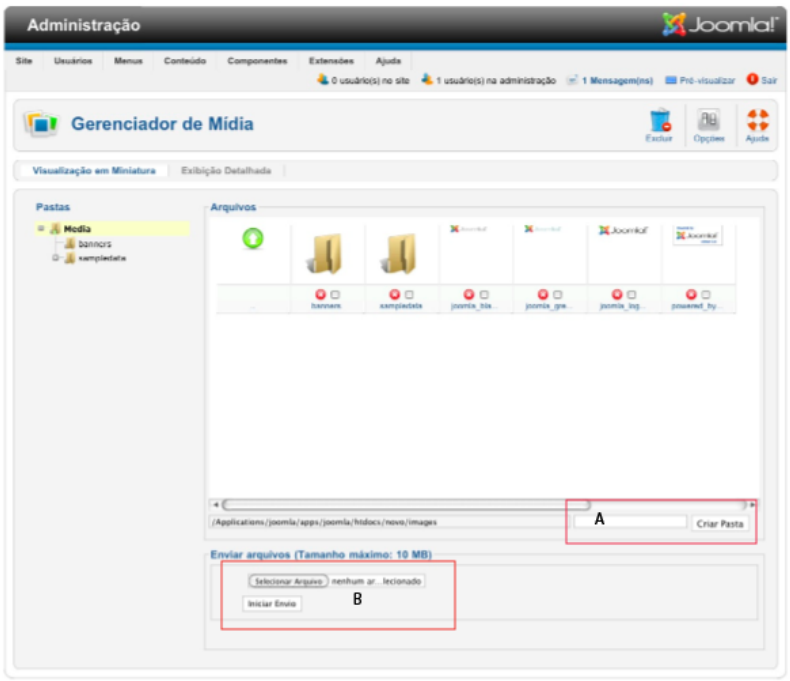


Figura 3.10. Gerenciador de Mídia

Para que os usuários possam visualizar o conteúdo na *frontend*, cria-se menus associados à categoria, artigo, weblink e outros. Na versão 1.6 esse processo foi melhorado ficando mais organizado e com mais opções. Na criação de menus existem duas opções:

- **Criar Menu;** Menu-> Gerenciador de Menus-> Criar novo menu; Nesta opção cria-se menus associados a módulos (ver sessão 4.15) e um menu raiz que não é associado a nenhum conteúdo. Na associação com o módulo, pode-se escolher onde este e suas opções ficarão.

- **Criar item de menu;** Menu -> (Nome do Menu) Em seguida, seleciona-se a opção Adicionar item de menu. Nessa opção pode ser criado um item de menu, que faz parte de um menu raiz. Depois de criado o Menu, cria-se os itens de menu. O joomla, por padrão tem o *"Main Menu"*, um menu principal já criado para facilitar o trabalho de gerenciamento. Em sites com apenas um menu não é necessário criar o módulo e associá-lo, apenas cria-se os itens de menus no *"Main Menu"*.

Um menu é criado em Menu->Gerenciador de Menus->Criar novo menu, preenchendo com o título do menu e seu nome para o sistema. O nome será usado para vincular à um módulo de menu, e opcionalmente a uma descrição. Após preenchido clica-se em Salvar->Fechar. A Criação dos itens de menu, é feita acessando Menu->(Nome do Menu), seguido pela opção Adicionar item de menu. Por fim, deve-se preencher os campos ilustrados na figura 4.11. Alguns deles são explicados a seguir.

- **Tipo de item de Menu.** Clicando em selecionar aparecerá as opções (ilustração na Figura 4.12), essas opções mudam de acordo com os módulos instalados. Não

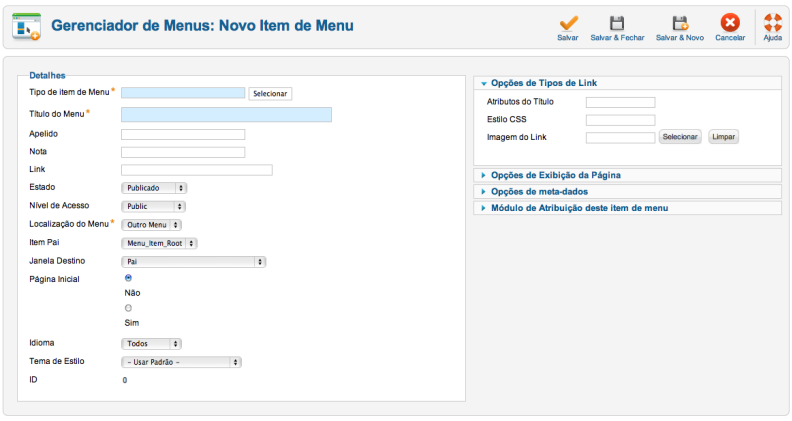


Figura 3.11. Criação de Item de Menu

caberá neste momento a descrição de cada uma das opções, por serem muitas e multiplicarem-se com a adição de novos módulos. Entretanto, todas são bem intuitivas e quando selecionada uma opção, surgirá outras mais, para o ajuste das configurações particulares ao menu escolhido.

- **Título do Menu.** Nome que aparecerá na *frontend*.
- **Nível de acesso.** Define a qual nível de acesso está vinculado esse menu (ver sessão 4.7).
- **Localização do Menu.** A qual menu raiz este pertencerá.
- **Item Pai.** Marca-se quando o menu a ser criado pertencerá ao item selecionado.

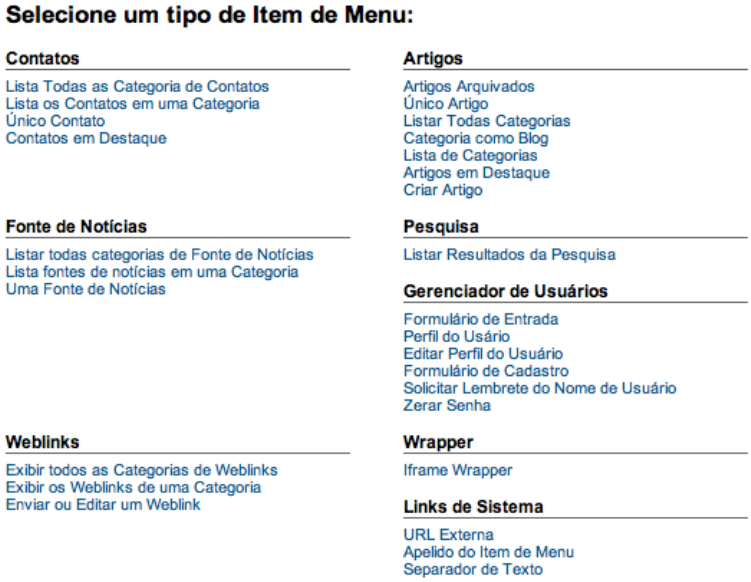


Figura 3.12. Escolha do tipo de menu

Exemplificando, deverá ser criado o menu "Material para alunos"e um item de menu para cada categoria de disciplina. O nível de acesso é o mesmo que o tratado na sessão 4.11, fazendo assim o acesso diferenciado para os alunos de cada curso. Na opção "Tipo de item de menu"clica-se em selecionar e escolha, na aba artigo e na opção "Categoria como blog", aparecerá á direita algumas opções de configuração, em Configurações obrigatórias seleciona-se a categoria desejada. Nota-se que pode ser adicionado um menu relacionado a apenas um artigo e um usuário, dentre várias outras opções listadas na Figura 4.12 podendo serem adicionadas mais.

Figura 3.13. Escolha de módulos

3.15. Módulos

O módulo mostra na *frontend* os conteúdos e opções. Tem-se módulos para menu, login de usuário, relógio, busca, dentre outros. O módulo é a parte mais interativa do usuário com o site. Estes trazem opções de lugares previamente definidos nos *templates*. Isto será exemplificado com a criação de um módulo para menu: clicar em "Extensões"e escolher "Gerenciador de Módulos". Clica-se em "novo"para criar um novo módulo. Em seguida, surgirão as opções ilustradas na figura 4.13. É válido relembrar que podem ser instalados módulos adicionais através do Gerenciador de Extensões, como no exemplo descrito na sessão 4.8. Neste exemplo, o pacote de idiomas foi instalado, mas qualquer outro módulo poderia ter sido instalado seguindo o mesmo procedimento.

Conforme o tipo de módulo temos configurações diferentes. Mesmo assim, não foge a simplicidade das configurações que o ambiente Joomla oferece, sendo bastante intuitivo até mesmo em módulos disponíveis para download na internet. No caso de surgirem problemas em algumas configurações, deve-se procurar no site, onde foi realizado o download, o arquivo correspondente ao manual de ajuda do pacote utilizado.

Seguindo com o exemplo, será criado um módulo para menu. Deve ser preenchido, além das informações básicas, informações relativas a localização do módulo (essas posições são predefinidas e poderão serem identificadas na pré-visualização do *template* (ver sessão 4.7)), e no caso do menu, selecionar a qual será associado, esta opção

encontra-se a direita da tela em "Opções Básicas".

3.16. Considerações Finais

Neste capítulo foi abordado o uso da ferramenta, Joomla, para criação de *websites*. Visto como instalar/configurar, alguns conceitos relevantes, criação de categorias, artigos, web-links e menus. O conteúdo abordado possibilita a criação de um *website*, mas deve-se procurar outras fontes para obtenção de maiores funcionalidades (as referências podem ajudar). Ao encontrar desafios ou dificuldade, os grupos Web ajudam bastante, podendo disponibilizar a resolução de problemas e novas funcionalidades. Os usuários que optem por usar profissionalmente a ferramenta, é aconselhável a associação em clubes de *templates*, para o desenvolvimento de *designs* diferenciados em cada site trabalhado.

Referências

[1] Webb, Cory. (2009). Beginning Joomla! Web Site Development. Wiley Publishing, Inc.

[2] Graff, Hagen. (2006). Building Websites with Joomla!. Packt Publishing

[3] Rahmel, Dan. (2007). Beginning Joomla!: From Novice to Professional. Apress.

[4] Rahmel, Dan. (2007). Professional Joomla!: From Novice to Professional. Apress.

[5] Dawson, Brandon, Canavan, Tom. Money-making weapons for your Joomla! website. Packt Publishing.

[6] Harwani, B. M. (2009). Foundation Joomla!. Friendsof.

[7] Torres, João Vítor.(2006). Gestão de Conteúdos com Joomla.

[8] Shreves, Ric. (2010). Joomla! Bible. Wiley Publishing, Inc.

[9] Holzner, Steven, Conner, Nancy. (2009). Joomla! For Dummies. Wiley Publishing, Inc.

[10] North, Barrie. (2007). Joomla! Um Guia do Usuário - Construindo um Website Poderoso com o Joomla!. Prentice Hall Professional.

[11] Chapman, Daniel. (2009). Joomla! 1.5x Customization - Make Your Site Adapt to Your Needs. Packt Publishing.

[12] Severdia, Ron. Crowder, Kenneth. (2010). Using Joomla - Building Powerful and Efficient Websites. O?Reilly Media.

[13] Soares, Hugo.Joomla! Visão geral Para Programadores (Componentes). Creative Commons.

[14] http://www.joomlabrasilia.org/.

[15] http://www.joomla.org/.

[16] http://www.joomla.com.br/.

[17] http://www.joomlaclube.com.br/.

[18] http://www.joomlapt.com/.

Capítulo 4

Desenvolvimento Dirigido por Testes de Aceitação

Pedro de Alcântara Santos Neto, Antonio Gabriel Di Atlanta Valente e Ítalo Linhares de Araújo

Abstract

The Acceptance Test Driven Development prioritizes the client, by the elevation of the relevance of acceptance testing. These tests are used to guide the development process. The use o ATDD is increasing and compliance tools are being developed, as Fitnesse. In this chapter are presented the concepts e principles related to ATDD, as well as an overview of the Fitnesse.

Resumo

O Desenvolvimento Dirigido por Testes de Aceitação (ATDD) prioriza o cliente, a partir da elevação da importância dos testes de aceitação. Esses testes são utilizados para guiar o processo de desenvolvimento. O uso do ATDD está crescendo e ferramentas de apoio estão sendo desenvolvidas, como por exemplo a FitNesse. Nesse capítulo são apresentados os conceitos e princípios relacionados ao ATDD, bem como uma visão geral da ferramenta Fitnesse.

4.1. Introdução

Nas próximas décadas a demanda por software irá aumentar exponencialmente [Broy 2006] [Boehm 2006] por consequência do intenso uso em larga escala, como por exemplo, na integração de negócios e em novas plataformas *web* e móveis. Ou seja, nos tornamos cada vez mais dependentes desses sistemas de software, por esse motivo necessitamos de mecanismos de garantia de qualidade dos próprios.

Vários fatores de qualidade, dentre eles disponibilidade, confiabilidade, segurança e robustez são garantidos por mecanismos como testes de software, que constituem elementos críticos na garantia de qualidade, representando a revisão final da especificação, projeto e geração de código [Pressman 2006]. O teste pode ser definido

como sendo verificação dinâmica do funcionamento de um programa, em um conjunto finito de casos de teste, cuidadosamente selecionado dentro de um domínio infinito de entradas, contra seu funcionamento esperado [Abran et al. 2004].

Apesar de melhorar substancialmente qualidade ao software, existe um grande empecilho na realização de testes: seu custo é considerável em um projeto de desenvolvimento. Dependendo das características do software a ser desenvolvido, esse custo pode significar mais de 50% do projeto [Pressman 2006]. Um relatório publicado pelo NIST [NIST 2002] mostra uma ideia dos custos envolvidos com teste, sendo relatado que o valor relativo ao custo de falhas em softwares desenvolvidos nos Estados Unidos, em 2002, foi de U$ 59.500.000.000,00. Esse mesmo relatório ainda estima que mais de 37% desse custo (U$ 22.200.000.000,00) poderia ter sido eliminado se a infra-estrutura para teste fosse melhorada.

Uma vez que os custos relacionados a falhas podem ser elevados, o investimento em garantia de qualidade torna-se essencial para diminuição desses custos. Para aferir essa qualidade em um software necessita-se da conformidade aos requisitos, sendo justamente nesse aspecto que a execução dos testes atua. Logo, uma forma de reduzir os custos das atividades relacionadas aos testes é automatizando essas execuções utilizando ferramentas apropriadas e desenvolvidas com esse foco. Há muitos trabalhos sendo desenvolvidos nessa linha visando a diminuição de custos e aumento de produtividade.

Como citado anteriormente, há vários tipos de testes, cada qual com seu foco. Entre eles temos: testes funcionais para verificar o comportamento do software; testes de aceitação, com o mesmo objetivo, porém executadas pelo usuário final para decidir sobre a aceitação do produto desenvolvido; testes de desempenho para verificar se o funcionamento do software atende a certos parâmetros de especificados; testes de estresse para verificar se o sistema funciona sob condições anormais de demanda; testes de usabilidade para verificar se o software é adequado ao uso; testes de regressão para mostrar que o software continua funcionando após alguma alteração; teste de instalação para verificar se os procedimentos de instalação são corretos e bem documentados; testes de segurança para verificar o nível de segurança do sistema; teste alfa para verificar seu funcionamento com pequenos grupos de usuários trabalhando em ambiente controlado; e teste beta em um ambiente não controlado [Neto et al. 2007]. Este capítulo é focado no teste de aceitação.

A realização de alguns tipos de testes, como os testes funcionais e testes de aceitação, normalmente é feito após o desenvolvimento da aplicação, todavia, com o surgimento do paradigma de Desenvolvimento Dirigido por Testes (TDD), os testes são desenvolvidos antecipadamente ao código, sendo utilizados para auxiliar o levantamento de requisitos, guiar o desenvolvimento, automatizar a criação de outros testes, além de auxiliar a geração da documentação do sistema e facilitar sua manutenção. Essa técnica é simples e eficaz para desenvolvimento de software de alta qualidade, a partir da flexibilidade de um projeto continuamente sendo melhorado [Koskela 2008].

O TDD foi estendido e deu origem ao Desenvolvimento Dirigido por Testes de Aceitação (ATDD), com intuito de guiar o desenvolvimento a partir da automação dos testes criados pelos usuários finais da aplicação, feitos nos estágios iniciais do

desenvolvimento de software [Azdic 2009] [Crispin e Gregory 2009]. Essa prática com testes de aceitação ainda é recente e não está totalmente difundida, no entanto, existem ferramentas para essa técnica, que facilitam a automação desse tipo de teste, a partir de mecanismos simples de definição de testes e padrões para facilitar sua ligação a uma implementação executável. São exemplos de ferramentas para o ATDD o FIT [FIT 2010], FitNesse [FITNESSE 2010] e Concordion [CONCORDION 2010].

Neste capítulo será descrito o ATDD, iniciando com um detalhamento do TDD, para em seguida apresentarmos o ATDD, juntamente com alguns exemplos do uso da ferramenta Fitnesse..

4.2. Desenvolvimento Dirigido por Testes

O Test Driven Development (TDD) é uma metodologia de desenvolvimento que tem como objetivo desenvolver o teste antes do código [Koskela 2008]. Inicialmente, cria-se um teste já sabendo que o teste irá falhar e depois se cria o código do *software* para que o teste passe.

O TDD, no desenvolvimento de software, pode ser considerado como um conjunto de regras e guias que levam a um determinado lugar e no tempo certo. Para melhor exemplificar, seria como chegar a uma estrada onde tivessem duas possibilidades de seguir em frente. Sem a orientação do TDD, poderia seguir por um caminho e após certo tempo descobrir que não era o caminho correto e depois voltar pela outra estrada. Porém com a orientação, o caminho a ser seguido já seria o correto estando no lugar correto no tempo esperado [Koskela 2008].

Para que a prática do TDD tenha sucesso é necessário seguir alguns passos. O primeiro deles é a decomposição dos requisitos. Eles precisam ser quebrados em tarefas que devem ser convertidas em testes e que irão guiar o desenvolvimento do software [Koskela 2008].

Não é necessário que os testes compilem ou executem para serem criados. Os primeiros códigos dos testes não vão compilar e isto é o correto porque não existe, ou não era pra existirem, classes no projeto do *software*. Como já foi dito, o código do programa, incluindo as “menores” classes, só deve ser criado após a criação dos testes. Um fato importante de ser ressaltado é que não tem como definir quando um teste pode ser considerado bom. Por isso, não existem regras para a criação de classes de teste. Alguns autores consideram que um bom teste é aquele atômico e isolado. Um teste pode ser considerado isolado quando não depende do resultado de outro teste para ser executado. Para ser atômico ele necessita testar cada parte separadamente.

4.2.1. Exemplo de desenvolvimento guiado por testes utilizando JUnit

Para melhor mostrar o uso do TDD será criado um novo projeto para implementar um programa que classifique um triângulo, baseado nas medidas dos seus lados.

Para iniciar o projeto uma IDE para desenvolvimento na linguagem JAVA deve ser escolhido, com posterior criação de um novo projeto. As classes podem ser criadas no pacote default, mas com o objetivo de separar e melhor gerenciar as classes do projeto, elas serão separadas em pacotes neste exemplo. Para as classes que servirão para guiar o desenvolvimento e testar o código será necessário criar um pacote chamado *teste*. No exemplo será criado uma nova classe chamada TesteTriangulo.java.

Como foi dito anteriormente, o código só pode ser escrito após os testes terem sido criados. A classe de teste foi criada, porém os testes não foram implementados. Será necessário importar, inicialmente, dois pacotes sendo um deles o org.junit.Test e o segundo, que deve ser importado estaticamente, é o org.junit.Assert.*. Esses pacotes que foram importados servem, respectivamente, para identificar um método que será interpretado como um caso de teste e para poder invocar métodos de asserção, provenientes da ferramenta de teste denominada JUnit [Gamma e Beck 2010]. Esses métodos serão invocados para a comparação dos resultados esperados com os obtidos no teste.

Após importar esses pacotes pode-se iniciar a codificação dos testes. Para identificar um método que testará alguma função da aplicação é necessário colocar antes dele a expressão @Test. O método a ser criado deve ser público, não deve retornar nada (void) e o seu nome deve identificar de forma clara o que se está testando. Por exemplo, a classe TesteTriangulo.java terá um método public void trianguloEquilatero(), onde será testado se um determinado triângulo é eqüilátero. Isso pode ser verificado na Figura 4.1.

```
1  package teste;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5
6  public class TesteTriangulo {
7
8      @Test
9      public void trianguloEEquilatero(){
10         Triangulo triangulo = new Triangulo(3,3,3);
11         assertEquals("equilatero",triangulo.tipo());
12     }
13
14 }
15
```

Figura 4.1. Classe de Teste do Triângulo.

Como pode ser visto na Figura 4.1, na linha 10 existe um erro que seria a não existência da classe Triangulo.java no projeto. Para corrigir esse erro é criada a classe Triangulo.java. Essa classe também pode ser criada no pacote default, mas pelo mesmo motivo anterior, criaremos em outro pacote que será chamado triângulo. Após a criação da classe, dois erros serão acrescentados: a inexistência do construtor com três parâmetros e a ausência do método tipo() que também não foi implementado. Para corrigir esses problemas o construtor e o método devem ser implementados. A classe Triangulo.java deve ficar conforme a Figura 4.2:

```java
package triângulo;

public class Triangulo {

    int ladoUm;
    int ladoDois;
    int ladoTres;

    public Triangulo(int ladoUm, int ladoDois, int ladoTres) {
        this.ladoUm = ladoUm;
        this.ladoDois = ladoDois;
        this.ladoTres = ladoTres;
    }

    public String tipo() {
        return null;
    }

}
```

Figura 4.2. Classe do Triângulo.

O retorno do método tipo() deve ser nulo até o momento que o teste falhar. Após a execução do teste pode ser verificado que o teste falhou e com isso alterar o código. A Figura 4.3 mostra como ficará a classe TesteTriangulo.java quando o teste for executado.

```java
package teste;

import static org.junit.Assert.*;
import org.junit.Test;

import triângulo.Triangulo;

public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

}
```

Figura 4.3. Classe de teste após falha no método tipo().

A parte do código que está em vermelho identifica que aquele método foi executado, mas encontrou algum erro. Nesse caso a falha encontrada foi causada pela diferença entre a resposta esperada, que deveria ser “eqüilátero”, e o retorno do método, que foi nulo. Para corrigir esse erro o retorno do método é alterado para a resposta desejada. A Figura 4.4 mostra como deve ficar o método após a correção. Para verificar se a alteração teve efeito a classe de teste deve ser executada de novo. Se o método ficar com a cor verde pode-se concluir que o teste passou.

```java
public String tipo() {
    return "equilatero";
}
```

Figura 4.4. Método alterado após teste falhar.

```
public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

}
```

Figura 4.5. Classe de teste após a nova execução

No código exibido na Figura 4.5 foi criado o teste para verificar se a classificação de um triângulo eqüilátero é feito com sucesso. É necessário criar outros métodos de teste, visando verificar as outras possibilidades existentes na classificação de triângulos. No entanto, isso é feito aos poucos, para que o código evolua em um ritmo adequado, mantendo sempre testes para cada parte a ser desenvolvida.

A Figura 4.6 mostra mais um método adicionado na classe de teste. Nesse caso o teste foi criado para verificar se um triângulo isósceles é corretamente classificado. Novamente o teste deve falhar, até que o código da classe Triangulo seja corretamente alterado, refletindo o que se pretende adicionar de comportamento.

```
public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

    @Test
    public void trianguloEIsosceles(){
        Triangulo triangulo = new Triangulo (3,3,4);
        assertEquals("isosceles",triangulo.tipo());
    }

}
```

Figura 4.6. Método de teste para verificar um triângulo isósceles.

```
public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

    @Test
    public void trianguloEIsosceles(){
        Triangulo triangulo = new Triangulo (3,3,4);
        assertEquals("isosceles",triangulo.tipo());
    }

}
```

Figura 4.7. Falha no método de teste para verificar um triângulo isósceles.

Após mais uma execução dos testes pode-se verificar que o teste falhou (Figura 4.7) por que é necessário alterar o código da classe Triângulo para que o teste com os novos casos criados sejam atendidos. Nesse caso, a alteração deve ser mais do que alterar o retorno, pois só assim não seria possível passar nos dois testes, demonstrado na Figura 4.8.

```
public String tipo() {
    if (ladoUm == ladoDois && ladoDois == ladoTres)
        return "equilatero";
    else
        return "isosceles";
}
```

Figura 4.8. Método tipo() alterado para retornar isosceles

Agora deve ser acrescentado mais um método de verificação. No caso será adicionado o método para testar a classificação de um triangulo escaleno. Como nos outros exemplos, o teste irá falhar. Em seguida será adicionada a verificação no método da classe triângulo. As figuras 4.9, 4.10 e 4.11 mostram os passos necessários para a adição do tipo escaleno aos tipos de triângulo.

```
public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

    @Test
    public void trianguloEIsosceles(){
        Triangulo triangulo = new Triangulo (3,3,4);
        assertEquals("isosceles",triangulo.tipo());
    }

    @Test
    public void trianguloEEscaleno(){
        Triangulo triangulo = new Triangulo (3,4,5);
        assertEquals("escaleno",triangulo.tipo());
    }

}
```

Figura 4.9. Classe de teste com verificação para triângulo escaleno.

```
public class TesteTriangulo {

    @Test
    public void trianguloEEquilatero(){
        Triangulo triangulo = new Triangulo(3,3,3);
        assertEquals("equilatero",triangulo.tipo());
    }

    @Test
    public void trianguloEIsosceles(){
        Triangulo triangulo = new Triangulo (3,3,4);
        assertEquals("isosceles",triangulo.tipo());
    }

    @Test
    public void trianguloEEscaleno(){
        Triangulo triangulo = new Triangulo (3,4,5);
        assertEquals("escaleno",triangulo.tipo());
    }

}
```

Figura 4.10. Classe de teste com método de teste do escaleno falho.

```
public String tipo() {
    if (ladoUm == ladoDois && ladoDois == ladoTres)
        return "equilatero";
    else if ((ladoUm == ladoDois && ladoUm != ladoTres)
            || (ladoUm == ladoTres && ladoUm != ladoDois)
            || (ladoDois == ladoTres && ladoDois != ladoUm)) {
        return "isosceles";
    } else {
        return "escaleno";
    }
}
```

Figura 4.11. Método tipo() alterado

Existe ainda a possibilidade de se ter valores inválidos para a criação de um triângulo. Para esses casos deve haver a verificação se um lado é menor do que a soma dos demais e maior do que a soma dos outros dois. As duas imagens abaixo mostram como ficariam os métodos na classe de teste e na classe principal com a devida adição dos testes e do novo código à classe.

```
@Test
public void trianguloEInvalido(){
    Triangulo triangulo = new Triangulo (1,2,3);
    assertEquals("invalido",triangulo.tipo());
}
```

Figura 4.12. Método de teste de triângulo inválido.

```
public String tipo() {
    if (((ladoUm < ladoDois + ladoTres) && (ladoUm > ladoDois - ladoTres))
            && ((ladoDois < ladoUm + ladoTres) && (ladoDois > ladoUm
                    - ladoTres))
            && ((ladoTres < ladoUm + ladoDois) && (ladoTres > ladoUm
                    - ladoDois))) {
        if (ladoUm == ladoDois && ladoDois == ladoTres)
            return "equilatero";
        else if ((ladoUm == ladoDois && ladoUm != ladoTres)
                || (ladoUm == ladoTres && ladoUm != ladoDois)
                || (ladoDois == ladoTres && ladoDois != ladoUm)) {
            return "isosceles";
        } else {
            return "escaleno";
        }
    } else {
        return "invalido";
    }
}
```

Figura 4.13. Método tipo() identificando os tipos de triângulo.

Como pode ser visto no exemplo acima, o código foi desenvolvido com base nos testes que foram criados. Também pode ser percebido que os testes eram atômicos e independentes. No caso, cada parte é testada independente do resultado dos outros testes e cada parte é a menor que pode ser separada.

Pode-se notar que o software foi desenvolvido de uma melhor forma, com mais cautela e sempre tentando atender aos testes. Com isso fica mais fácil desenvolver sem perder o objetivo do projeto. Além disso, o custo e o tempo de desenvolvimento tendem a ficar menores, por conta da eliminação do retrabalho e facilidade para se encontrar qualquer problema em eventuais necessidades de alteração.

4.2.2. Problemas do Desenvolvimento Dirigido por Testes

O TDD é uma técnica bastante utilizada pela comunidade atualmente. Ele possibilita uma série de características desejáveis, como melhora documentação, garante boa cobertura do teste, facilita manutenção, etc. Porém, eles são feitos pelos desenvolvedores, geralmente utilizando uma ferramenta de programação.

Embora isso não seja uma desvantagem, normalmente torna-se bastante complicado para um cliente ajudar no desenvolvimento via TDD. Via de regra, os clientes preferem ser figuras ativas no desenvolvimento, podendo criar suas situações para verificação sem ter que solicitar apoio ao pessoal de desenvolvimento.

O ATDD atua justamente nesse aspecto, a partir da disponibilização de ferramentas simples para que os clientes se tornem membros ativos do desenvolvimento, criando testes sempre que desejarem e podendo acompanhar o status do desenvolvimento, a partir da verificação da execução dos seus testes. A próxima seção é dedicada à explanação dessa nova técnica, que tende a colocar clientes e desenvolvedores de forma muito mais sinérgica em qualquer projeto de desenvolvimento de software.

4.3. Desenvolvimento Dirigido por Testes de Aceitação

Como vimos anteriormente, o TDD ajuda os desenvolvedores a produzirem códigos de alta qualidade, porém os clientes, normalmente, não estão interessados em como foram construídos esses códigos, e sim como o software se comporta de forma a tornarem mais produtivo suas atividades, dando os resultados esperados por eles. Ou seja, necessitam da funcionalidade conforme a regra de negócio especificada.

Nessa conjuntura, temos o Desenvolvimento Dirigido por Testes de Aceitação (ATDD). Enquanto o TDD garante a qualidade técnica do software, o ATDD fornece suporte para o atendimento às necessidades do negócio durante seu desenvolvimento, a partir dos testes de aceitação que são especificações de como deve se comportar determinada funcionalidade do sistema. Em suma, Desenvolvimento Dirigido por Testes de Aceitação ajuda a coordenar projetos de software de uma maneira que permite aos desenvolvedores entregar exatamente o que o cliente quer, quando ele quer [Koskela 2008].

A Figura 4.14 mostra a relação do Desenvolvimento Dirigido por Testes de Aceitação e do Desenvolvimento Dirigido por Testes [Koskela 2008]. O ciclo inicia pela coleta de requisitos (estórias) a partir de várias reuniões com os clientes, nas quais são descritas novas características do *software*. Todas as informações coletadas são especificadas para então serem escritos os testes de aceitação para os requisitos levantados, tentando-se obter a maior cobertura possível do cenário em questão. Os testes de aceitação deverão ser falhos por não possuir nenhuma ligação com o *software* a ser testado.

A partir do requisito especificado a equipe pode partir para o Desenvolvimento Dirigido por Testes, iniciando o processo de codificação do sistema. Nessa fase do desenvolvimento, o código é testado e caso não tenha resultado positivo é alterado para que passe nos testes (TDD), sendo refatorado para melhorar sua estrutura sempre que

possível. Esse ciclo continua até que o software passe em todos os testes criados pelos desenvolvedores e atinja uma estrutura aceitável.

Uma vez que o código está funcional, pelo menos sob a perspectiva dos desenvolvedores, é chegado o momento de submetê-lo aos testes criados pelos usuários finais da aplicação. Nesse momento o código é ligado aos testes de aceitação. Eles são executados, gerando refatorações e mudanças no código, até que ele esteja completamente operacional, atendendo a todos os requisitos e passando nos testes de aceitação. Se o *software* atender aos requisitos, poderá ser considerado pronto. Caso contrário, todos os passos são repetidos para modificar o programa para que possa atender todas as exigências e expectativas do cliente.

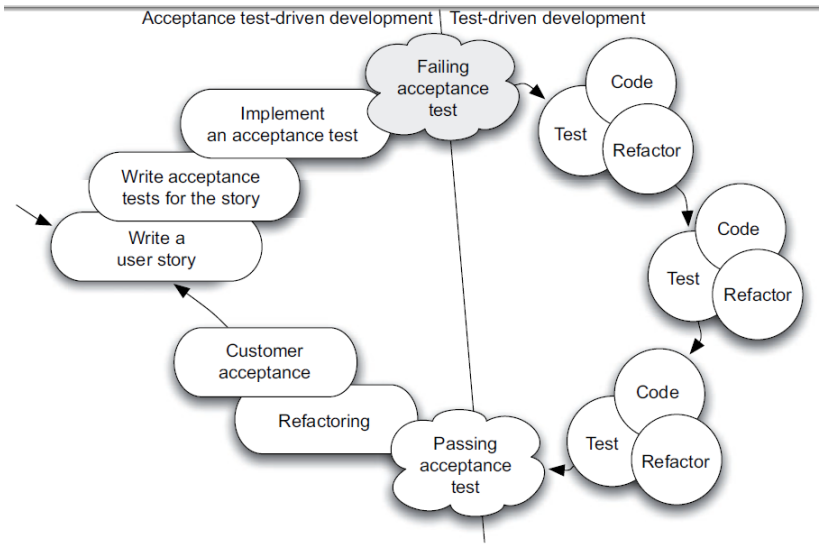


Figura 4.14. Ciclo de funcionamento do ATDD e TDD

4.3.1. Iniciando o uso do ATDD

Anteriormente vimos a aplicação de um exemplo utilizando o paradigma de Desenvolvimento Dirigido por Testes no contexto de uma aplicação para classificar um triângulo. Agora, iremos mostrar exemplo semelhante, porém sob a ótica do ATDD utilizando a ferramenta FitNesse como apoio para essa tarefa.

Na Figura 4.15 é exibida a página inicial da FitNesse. Para criar um teste, basta criar uma tabela simples. Para isso, é necessário ter um conhecimento básico de como funciona um *wiki*. Depois de instalado o FitNesse, o usuário estará habilitado a fazer qualquer criação e edição de página usando seu *Web Browser*. Qualquer teste deve ser criado a partir da página inicial da Fitnesse.

Figura 4.15. Página inicial da FitNesse.

4.3.1.1. Criação de uma página de teste

Para se criar uma página é necessária inseri-la em outra página já existente, a partir da sua edição. Isso permite que seja criada uma página para se especificar um novo teste. Nesse caso então, é necessário clicar no botão *Edit* situado no canto superior esquerdo da página *FrontPage* (única página existente no início do uso da FitNessse), exibida na Figura 4.15. O usuário será redirecionado para a página de edição, conforme é visto na Figura 4.16.

Figura 4.16. Exemplo de página em criação.

Todo teste na FitNesse é, na verdade, uma tabela contendo dados de entrada e saídas esperadas. Assim, para criar uma página que vai conter as tabelas utilizadas em um determinado teste devemos defini-la utilizando uma convenção de nomenclatura. Para nomes de tabelas, devemos utilizar o formato *CamelCase*, que prescreve que os nomes de uma tabela sejam formados por palavras e que cada palavra inicie com letra maiúscula, sem utilizar nenhum caractere especial para essa ligação. Um exemplo de nome de tabela seguindo essas prescrições seria *TestTableLogin*.

Na Figura 4.16 podemos ver que foi adicionado o termo "*>TesteTriangulo*". Ao salvarmos a página com essa alteração, podemos notar que um novo link será exibido na página original (Figura 4.17). Ao clicar nesse link, localizado ao lado do termo *TesteTriangulo*, seremos direcionados para a edição da nova página (Figura 4.18). Essa

será a página que utilizaremos para demonstrar o uso do ATDD para o exemplo da construção de um programa para classificar triângulos.

Figura 4.17. Exemplo de página em criação.

Figura 4.18. Exemplo de edição de página.

Clicando em *Save* no canto esquerdo inferior, mostrado na Figura 4.18, visualizaremos a página em branco, conforme a Figura 4.19.

Figura 4.19. Exemplo de página criada em branco.

As páginas do FitNesse tem alguns tipos. Esses tipos são definidos no botão *Properties* localizado no menu à esquerda, mostrado na Figura 4.19. Clicando nesse botão podemos definir algumas propriedades da página, como as visualizadas na Figura 4.20.

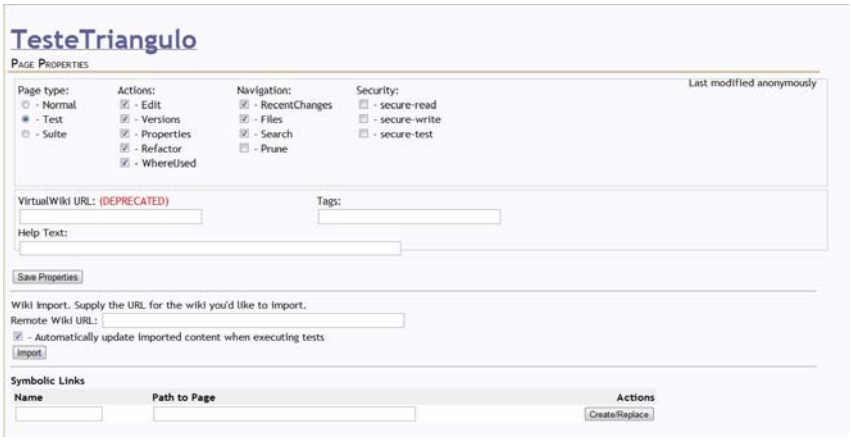


Figura 4.20. Exemplo de propriedades de página

Conforme exibido na Figura 4.20, é possível definir o tipo de página, sendo possível: *Normal*, *Test* e *Suite*. O tipo Normal define que a página será apenas *web* sem uma característica especial, o tipo *Test* define que será uma página de teste, podendo conter definições de testes para serem executados e o tipo *Suíte*, que será uma coleção de páginas de teste que podem ser executados em conjunto e que poderão ter seus resultados mostrados em uma única página. Para o exemplo de demonstração do uso do ATDD iremos utilizar o tipo *Test.*

Na página de propriedades exibidas na Figura 4.20 existem outras definições que não vamos alterar devido não precisarmos de nenhuma definição diferente das que foram exibidas.

4.3.1.2. Especificação do teste

O diferencial do Desenvolvimento Dirigido por Testes de Aceitação para o Desenvolvimento Dirigido por Testes é o fato de quem especifica os testes na ferramenta é o cliente, logo iremos simular como o cliente especificaria o teste do triângulo feito anteriormente no JUnit, na ferramenta FitNesse.

Temos a página criada e definida como do tipo *Test*, conforme a Figura 4.19. Clicamos no botão *Edit* para definirmos os dados da tabela do teste de aceitação a ser executado. O cliente definirá a tabela com os dados de entrada e as saídas esperadas do exemplo do triangulo, conforme a Figura 4.21, utilizando a sintaxe *wiki*, no qual ele terá que ter conhecido.

O cliente apenas colocará os dados de entrada como os valores dos lados, o ladoUm, ladoDois e ladoTres, e o tipo que o cliente quer que seja, no caso: eqüilátero, isósceles, escaleno e inválido.

Após clicar em *Save*, o cliente visualizará a tabela conforme a Figura 4.22. Essa é a especificação do teste de aceitação dos tipos de triângulo feita por um cliente com a ferramenta FitNesse.

Figura 4.21. Exemplo de edição de tabela definida pelo cliente

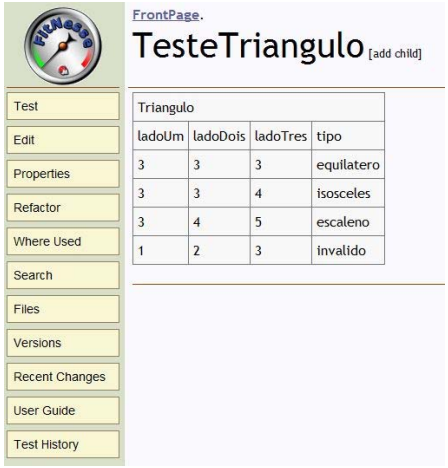


Figura 4.22. Exemplo de edição de tabela definida pelo cliente

Nesse momento entra em cena a figura do desenvolvedor, que será responsável por criar a ligação da tabela com a implementação a ser testada. Para isso o desenvolvedor deve modificar a tabela de teste, que será usada como exemplo para execução do teste de aceitação para o programa de classificação de triângulos. Esse passo é necessário devido a tabela necessitar de possuir algumas peculiaridades para ser possível criar a ligação com o código.

A página exibida na Figura 4.23 apresenta as modificações do desenvolvedor para incluir marcadores específicos da FitNesse. O primeiro marcador é o comando *!define*, utilizado para especificar o estilo de tabela que será utilizado, nesse caso, uma tabela SLIM (Simple List Invocation Method), ou seja, é uma lista simples de invocação de métodos.

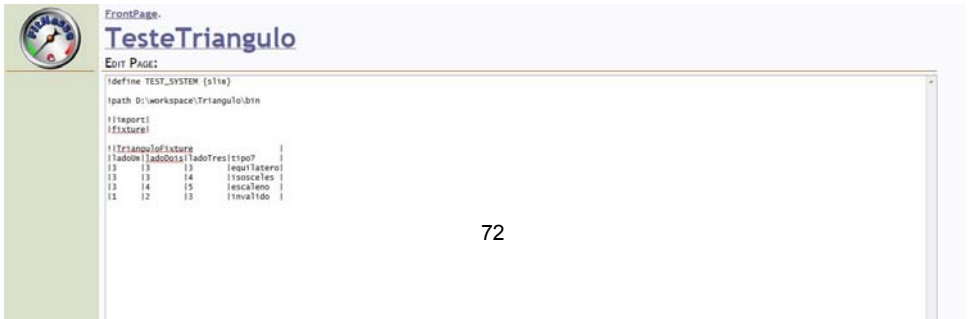


Figura 4.23. Exemplo de edição de tabela alterada pelo desenvolvedor

No marcador para o comando *!path* é especificado o caminho em que encontram-se todas as classes de suporte que serão necessárias para execução do teste. Logicamente, durante o início do ATDD, apenas os testes serão definidos, ficando essa indicação para um momento posterior, após a implementação do programa desejado.

Após essas definições, temos que especificar algumas outras tabelas, como por exemplo, a tabela de importação, especificando o pacote da classe *fixture*, responsável pela ligação dos teste e do código executável, além da tabela do teste de aceitação do triângulo. Os componentes existentes nessa parte são:

- *TrianguloFixture*: título da tabela, que vai representar o nome da classe da *fixture* que será instanciada;
- *ladoUm*: coluna com as entradas de valores inteiros do primeiro lado do triângulo, associado a um método *set* na classe fixture;
- ladoDois: coluna com as entradas de valores inteiros do segundo lado do triângulo, associado a um método *set* na classe fixture;
- ladoTres: coluna com as entradas de valores inteiros do terceiro lado do triângulo, associado a um método *set* na classe fixture;
- tipo*?:* coluna com as saídas esperadas sobre o tipo de triângulo, associado ao método *return* a partir do ponto de interrogação.

No preenchimento de uma tabela existem alguns botões de formatações, muito úteis às pessoas que especificam testes:

- *Spreadsheet to FitNesse*: facilita a cópia de parte de uma planilha para uma página da FitNesse, a partir da formatação dos dados copiados para um formato amigável na FitNesse.
- *FitNesse to Spreadsheet*: permite a cópia da tabela FitNesse para uma aplicação de planilha eletrônica, voltando ao formato não *wiki*.
- *Format*: comando utilizado para colocar os dados de uma página em um formato alinhado e legível.

Definido todas as informações necessárias à tabela, o usuário pode clicar em *Save* e a página deve ser mostrada conforme é exibido na Figura 4.24.

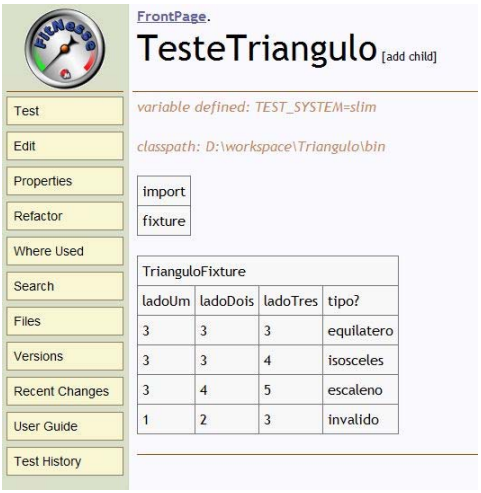


Figura 4.24. Exemplo de página com a tabela de teste.

4.3.1.3. Execução e criação da fixture

Para acessar o conteúdo de uma tabela e fazer sua ligação ao código executável, é necessário criar uma *fixture*, que é uma implementação em uma linguagem de programação da abstração da tabela contida na página *wiki* [Koskela 2008]. Essa linguagem deve ser suportada pela FitNesse.

Executaremos o teste a partir do botão *test* no canto superior esquerdo da página, mesmo sem existir nenhuma implementação de *fixture* para o teste. Logicamente, o resultado deve ser falho, como mostrado na Figura 4.25. Isso acontece por que a FitNesse não encontra nenhuma classe que faça a ligação com uma implementação, logo o teste não pode ser executado com sucesso.

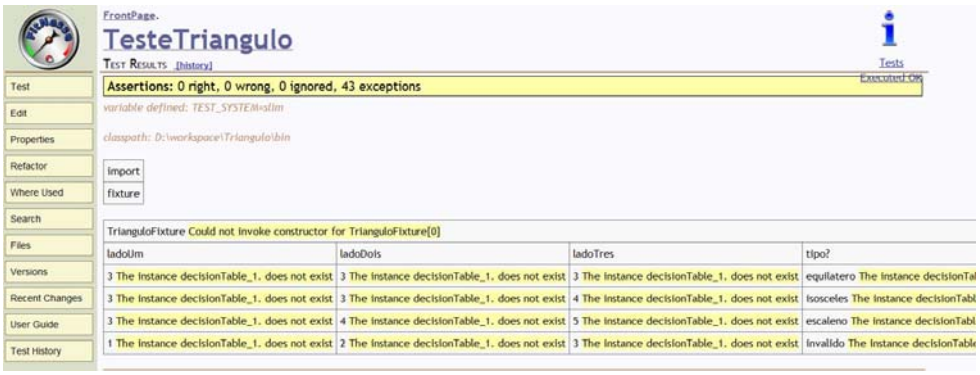


Figura 4.25. Exemplo de teste falho.

Para automação do teste definido na Figura 4.21 foi desenvolvida uma classe Java para funcionar como *fixture* para o teste. Ela deve ser chamada de TrianguloFixture (Figura 4.26), devido ao título utilizado na tabela do teste (Figura 4.24).

```
package fixture;

import entidade.Triangulo;

public class TrianguloFixture {
    private int ladoUm;
    private int ladoDois;
    private int ladoTres;

    public void setLadoUm(int ladoUm) {
        this.ladoUm = ladoUm;
    }

    public void setLadoDois(int ladoDois) {
        this.ladoDois = ladoDois;
    }

    public void setLadoTres(int ladoTres) {
        this.ladoTres = ladoTres;
    }

    public String tipo(){
        Triangulo triangulo = new Triangulo(ladoUm, ladoDois, ladoTres);
        return triangulo.tipo();
    }
}
```

Figura 4.26. Exemplo de código de fixture.

A Figura 4.26 exibe um exemplo de fixture para o teste contido na página TesteTriangulo (Figura 4.24). Quando é solicitada a execução dos testes contidos nessa página, a partir do acionamento do botão Test, uma instância da classe *TrianguloFixture* é criada. A FitNesse automaticamente cria essa classe e invoca os métodos de atribuição de valores existentes (*setters*), para que os dados de entrada sejam utilizados. Da mesma forma, ao encerrar a atribuição dos dados de entrada, é invocado o método de verificação do resultado. No caso da classe em questão, o método *tipo* é o responsável pelo retorno, uma vez que ele foi especificado utilizando um marcador (“?”) que indica justamente isso.

É justamente em algum método de verificação do resultado que devemos ligar o teste ao código do sistema sob teste. Pode-se observar que dentro do método *tipo* existe uma chamada ao método *tipo*() da classe Triângulo, mostrado na Figura 4.13. Esse método é o responsável pela avaliação das regras para análise de triângulos. A invocação desse método garante que o teste está usando as funcionalidades do sistema desenvolvido. Embora o exemplo seja simples, isso deixa claro como ligar o teste à implementação do sistema.

O resultado da execução do teste é exibido na própria página contendo a especificação do teste. Os campos associados aos resultados são coloridos de acordo com a resposta obtida, tornando-se verdes caso o resultado tenha sido bem sucedido, amarelos para exceções ou vermelho para casos de erro, como demonstrado na Figura 4.27.

Figura 4.27. Exemplo de resultado de teste bem sucedido.

A FitNesse mantém um histórico de execução de testes, acessado a partir do botão *Test History*, contido no canto inferior esquerdo de uma página de teste, como mostra a Figura 4.28. Acessando esse comando é possível visualizar todas as execuções dos testes de uma página.

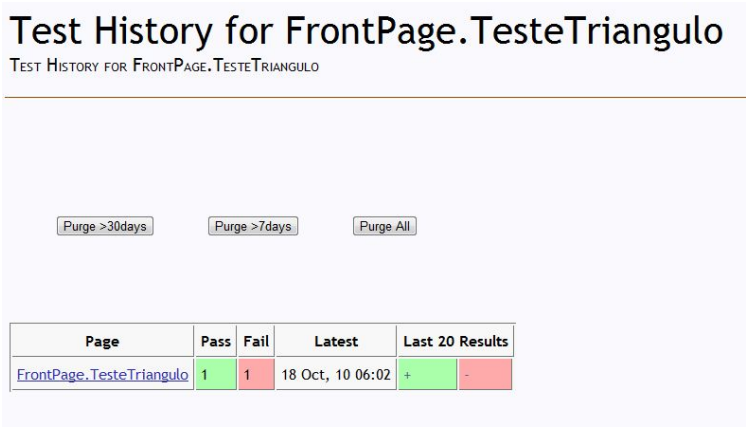


Figura 4.28. Exemplo de histórico de execuções de testes de uma página.

A partir do histórico de teste podemos ter algumas informações relevantes para o processo de desenvolvimento, tais como a quantidade de vezes que o teste foi bem sucedido e a quantidade de vezes que falhou, além de ter acesso a todos os testes executados. Isso pode ser visualizado a partir do acionamento do link *Last 20 Results*. Uma página de histórico normalmente possui três comandos: *Purge > 30 days*, que elimina do histórico os registros de execução com mais de 30 dias de execução, *Purge > 7 days* que elimina os registros de execução com mais de uma semana de execução e *Purge All*, que elimina todos os registros de execução.

4.3.2. Uso do ATDD em um Sistema de Informação

Na seção anterior exibimos o uso do ATDD em um exemplo simples, referente à classificação de um triângulo. Embora o exemplo seja válido para demonstrar como usar a ferramenta, ela não apresenta a conexão a um sistema real. Nesse exemplo, iremos executar um teste de aceitação para autenticação de login de um sistema chamado Capriovi, um *software* de controle zootécnico e genético de caprinos e ovinos. Será uma aplicação simples de ATDD, porém próximo da realidade.

Inicialmente, temos que definir alguns casos de testes juntamente com o cliente, para então procedermos com a especificação das tabelas FitNesse. Definimos três casos de testes, sendo eles: login válido (Figura 4.29), login com senha incorreta (Figura 4.30) e login com identificador inexistente (Figura 4.31).

Identificação	Login válido	
Itens a testar	Verificar se a tentativa de login utilizando os identificadores válidos retorna a mensagem apropriada.	
Entradas	**Campo**	**Valor**
	Login	admin
	Senha	admin
Saídas esperadas	**Campo**	**Valor**
	Mensagem	Entrada no sistema efetuada com sucesso.
Estado alcançado	Login efetuado.	
Ambiente	Capriovi	
Dependências	Não aplicável.	

Figura 4.29. Caso de teste de login válido.

Identificação	Login inválido com senha incorreta	
Itens a testar	Verificar se a tentativa de login utilizando o identificador de senha incorreto retorna a mensagem apropriada.	
Entradas	**Campo**	**Valor**
	Login	admin
	Senha	errada
Saídas esperadas	**Campo**	**Valor**
	Mensagem	Não foi possível realizar a entrada no sistema.
Estado alcançado	Sem usuário.	
Ambiente	Capriovi	
Dependências	Não aplicável.	

Figura 4.30. Caso de teste de login com senha incorreta.

Identificação	Login inválido com login incorreto	
Itens a testar	Verificar se a tentativa de login utilizando o identificador de login incorreto retoma a mensagem apropriada.	
Entradas	**Campo**	**Valor**
	Login	errada
	Senha	admin
Saídas esperadas	**Campo**	**Valor**
	Mensagem	Não foi possível realizar a entrada no sistema.
Estado alcançado	Sem usuário.	
Ambiente	Capriovi	
Dependências	Não aplicável.	

Figura 4.31. Caso de teste de login com identificador inexistente.

Seguiremos os mesmos passos iniciais demonstrados no exemplo anterior, com o cliente definindo a tabela inicial e o desenvolvedor a alterando para adicionar as peculiaridades (tipo de tabela de teste, *paths*, pacote, nome da classe *fixture* e método de retorno) necessárias para se criar a ligação com o sistema de informação Capriovi.

Figura 4.32. Edição da tabela FitNesse do teste de autenticação de login.

A tabela para especificação do teste de aceitação para a autenticação do sistema Capriovi é especificada na Figura 4.32. Nela foram definidos pelo desenvolvedor os *paths* onde se encontram as classes e as *libs* de apoio para execução dos testes. A tabela possui o nome de *LoginFixture*. Logicamente, esse deve ser o mesmo nome da classe *fixture* a ser criada. A tabela deve possuir três colunas e três linhas, para refletir os testes especificados anteriormente. As colunas devem representar os dados de entrada login e senha. Deve haver também uma coluna representando a saída esperada, no caso, a mensagem a ser obtida com a execução do teste.

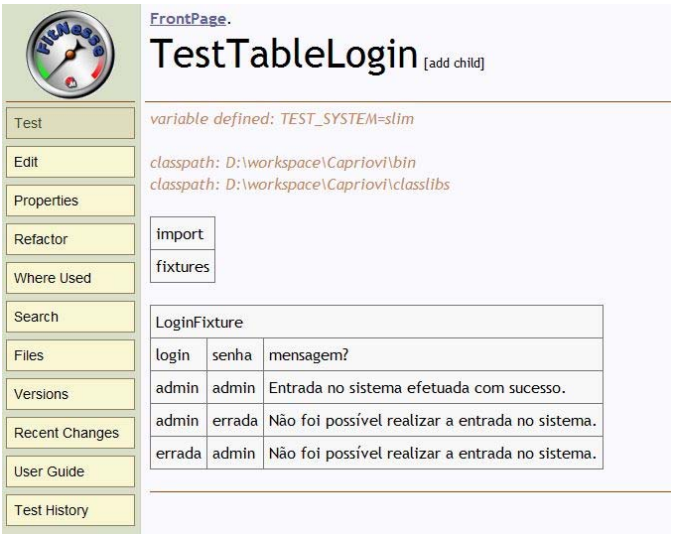


Figura 4.33. Tabela FitNesse do teste de autenticação de login.

Criada a tabela do teste, deve-se criar a *fixture*, para fazer a ligação com o sistema. Para a implementação do teste utilizaremos Java [JAVA 2010] e Selenium [SELENIUM 2010]. O Selenium é uma ferramenta para automação de testes funcionais de aplicações Web. A partir do seu uso é possível manipular páginas Web, entrando com valores, acionando comandos e obtendo os valores exibidos. Essa ferramenta é exatamente o que precisamos para fazer a ligação do teste de aceitação, especificado na FitNesse, com a aplicação a ser testada.

```
import com.thoughtworks.selenium.SeleneseTestCase;

public class LoginFixture extends SeleneseTestCase{

    private String login;
    private String senha;

    public LoginFixture() throws Exception{
        this.setUp();
    }

    public void setLogin(String login) {
        this.login = login;}

    public void setSenha(String senha) {
        this.senha = senha; }

    public String mensagem(){
        return this.efetuarLogin(login, senha);
    }

    public void setUp()throws Exception{
        setUp("https://sistemas.ufpi.br","*chrome");
    }

    public String efetuarLogin(String login, String senha){
        selenium.open("/capriovi");
        selenium.type("login", login);
        selenium.type("senha", senha);
        selenium.click("submit");
        selenium.waitForPageToLoad("30000");
        return selenium.getText("//div[@id='content']/div[2]/span");
    }
}
```

Figura 4.34. Fixture com código do Selenium

A Figura 4.34 exibe o código da *fixture* do sistema a ser testado juntamente com o código do Selenium. Essa classe, que vai fazer a ligação entre os testes e o *software*, deve ter o mesmo nome especificado na tabela *TestTableLogin*, no caso *LoginFixture*. Deve possuir também os métodos *setters* para atribuição de valores para as variáveis com os nomes das colunas de entrada de dados (*login* e senha) e um método (mensagem) que retorne uma string a ser comparada com o resultado esperado no teste.

Os métodos *setUp()* e *efetuarLogin()* são os métodos da ferramenta Selenium que irão auxiliar na captura das mensagens retornadas pelo *software*. Para que os testes possam ocorrer, o método *setUp()* deve ser chamado antes do comando *selenium.open("/capriovi")*, nesse caso a chamada foi feita no construtor, pois é necessário iniciar o *Web Browser* e a url do sistema. O método *efetuarLogin()* vai adicionar à url o *path* da página a ser testada, preencher os valores dos campos *login* e senha, clicar no botão para efetuar o login através do comando *submit* e por fim retornar a mensagem exibida pelo programa para que esta possa ser comparada com a mensagem especificada pelo cliente.

Quando solicitada a execução do teste contido na página *TestTableLogin* (Figura 4.33), uma instância da classe *LoginFixture* é criada, sendo invocados os métodos para atribuir os valores de entrada na referida instância. Em seguida, a FitNesse executa o método para verificar os resultados obtidos (método mensagem). É importante notar que a execução do teste na FitNesse dispara a execução do teste com Selenium.

O resultado da execução do teste é exibido na própria página contendo a especificação do teste. Um exemplo de execução pode ser visto na Figura 4.35. Os campos associados aos resultados são verdes, o que indica que o sistema Capriovi está correspondendo aos requisitos definidos pela especificação do teste de aceitação, ou seja, essa funcionalidade está funcionando conforme foi definida durante o ATDD.

Figura 4.35. Resultado da execução do teste.

4.4. Conclusões

O futuro do desenvolvimento de software está vinculado intimamente com a qualidade e, para obtê-la, é fundamental a utilização de testes. Partindo desse preceito, pressupõe-se que a nova proposição de Acceptance Test Driven Development terá cada vez mais aceitação no mercado e na academia, e como tal, pesquisas e ferramentas devem ser criadas para o aperfeiçoamento desse propósito.

A ferramenta FitNesse é um exemplo dessa iniciativa, ressaltando o ATDD, técnica ainda nova, mas em evolução. No entanto, analisando seu funcionamento foi possível observar que existem mecanismos para facilitar o trabalho dos desenvolvedores, facilitando também o uso da ferramenta e do próprio ATDD.

Neste trabalho apresentamos os princípios e conceitos do Desenvolvimento Dirigido por Testes, além de um exemplo utilizando-o em um desenvolvimento de uma simples aplicação. A partir dessa apresentação, mudamos o foco para o Desenvolvimento Dirigido por Testes de Aceitação, demonstrando o mesmo exemplo utilizado no TDD, no entanto, utilizando os preceitos de ATDD com a ferramenta FitNesse. Apresentamos também um exemplo do uso em um sistema de informação real, exemplo esse que exigiu o apoio da ferramenta Selenium. Os exemplos utilizados, embora simples, refletem os casos típicos do ATDD em um desenvolvimento de software.

O ATDD evidencia a importância do cliente, principal ator na construção das especificações do software. As ferramentas como a FitNesse tornam o envolvimento do cliente algo simples, por não exigir muito conhecimento para seu uso, ao mesmo tempo que permite a participação ativa no desenvolvimento, além de permitir um acompanhamento contínuo do trabalho. Acreditamos que o ATDD deve virar um padrão para o desenvolvimento de software em pouco tempo, dado suas características e benefícios associados ao seu uso.

Referências

Abran, A., Moore, J., Bourque, P., Dupuis, R., e Tripp, L. (2004) “Guide to the Software Engineering Body of Knowledge”, IEEE Computer Society.

Azdic, G. (2009) “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing”, Neuri Limited.

Boehm, B. W. (2006) “A View of 20th and 21st Century Software Engineering”, 28th International Conference on Software Engineering (ICSE), Shanghai, China.

Broy, M. (2006) “The ‘Grand Challeng’ in Informatics: Engineering Software-Intensive Systems, IEEE Computer, Vol. 39, No.10.

Crispin, L. e Gregory, J. (2009). “Agile Testing: A Practical Guide for Testers and Agile Teams”, Addison-Wesley Professional.

CONCORDION, “Concordion is an open source tool for writing automated acceptance tests in Java”, http://concordion.org/, último acesso em 15/10/2010.

FIT, “Fit: Framework for Integrated Test”, http://fit.c2.com/, último acesso em 06/09/2010.

FITNESSE, “The Fully Integrated Standalone Wiki, and Acceptance Testing Framework UserGuide”, http://fitnesse.org/, último acesso em 17/10/2010.

Gamma, E. and Beck, K., “Junit testing framework”, http://www.junit.org/, último acesso em 17/10/2010.

JAVA. http://java.com/, último acesso em 19/10/2010.

JUnit.org, http://www.junit.org/. último acesso em 10/10/2009.

Koskela, L. (2008) “Test Driven: Practical TDD and Acceptance TDD for Java Developers”, Manning Publications.

Pressman, R. (2006) “Engenharia de Software”, McGraw-Hill, 6th edition.

Neto, P.S., de Sousa, F.V., e Resende, R. (2007) “Automação de Teste de Software”, Escola Regional de Computação da SBC Ceará – Maranhão – Piauí, Fortaleza.

NIST (2002). “Planning Report 02-3”, National Institute of Standards and Technology, http://www.nist.gov/director/prog-ofc/report02-3.pdf, último acesso em setembro de 2009.

SELENIUM. *Web application testing system*. http://seleniumhq.org/, último acesso em 19/10/2010.

Capítulo

5

Tradução e interpretação de linguagens de domínio específico com a ferramenta ANTLR

Cairo Antonio Oliveira Rocha e Raimundo Santos Moura

Abstract

The ANTLR tools (ANother Tool for Language Recognition) is a parser generator that automates the construction of language recognizers. It uses has been growing throughout the computer community. ANTLR comes with complete source under the BSD license, that has been used to build translators and interpreters for domain-specific languages (DSLs). DSLs are languages that offer, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. DSLs are particularly important to software development because they represent a more natural, high-fidelity, robust, and maintainable means of encoding a problem than simply writing software in a general purpose language, like C, C++ or Java. The main objective of this work is present an overview about DSLs and a tutorial of ANTLR tools in the computer language context.

Keywords: Domain-Specific Languages (DSL), Language Recognition, ANTLR Tools

Resumo

A utilização da ferramenta ANTLR (ANother Tool for Language Recognition) na construção de tradutores, interpretadores e reconhecedores de linguagens vem crescendo em toda a comunidade da computação. O ANTLR é um software livre, lançado sobre a licença BSD, que vem sendo utilizado principalmente para construção de tradutores e interpretadores para Linguagens de domínio específico (do inglês: Domains Specific Languages – DSLs). DSLs são linguagens de alto nível projetadas sob medida para tarefas específicas. Elas são importantes para o desenvolvimento de software por representarem um meio mais natural, fiel, robusto e sustentável do que simplesmente escrever software em uma linguagem de propósito geral, como C, C++ ou Java. Esse trabalho tem como objetivo apresentar uma visão geral das DSLs e um tutorial da ferramenta ANTLR focado na construção de tradutores e interpretadores para linguagens de domínio específico.

Palavras-Chave: Linguagem de Domínio Específico, Reconhecedores de Linguagens, Ferramenta ANTLR

5.1. Introdução

O estudo das técnicas de compilação visa o conhecimento das estruturas e algoritmos usados na implementação de linguagens e destaca noções importantes sobre o uso de memória e eficiência no estilo de codificação dos profissionais da área de desenvolvimento de *software*. Essas técnicas são frequentemente aplicadas na solução de problemas que exigem alguma forma de tradução entre linguagens ou notações, na implementação de linguagens de domínio específico e na construção de geradores e analisadores de código. "***Projetar novas linguagens e construir tradutores para linguagens existentes, quando apropriado, é a marca registrada de um desenvolvedor profissional***" [Parr, 2007].

Com a criação dos computadores no século passado as linguagens de programação se tornaram o foco principal das pesquisas sobre linguagens. Noam Chomsky nos anos 60 propôs uma classificação para as linguagens que ficou conhecida como Hierarquia de Chomsky. Desde então diversas linguagens de programação têm sido criadas para resolver problemas em determinadas áreas, a saber: ***Cobol – programação comercial; Fortran – computação numérica e Lisp – processamento simbólico***. Porém, novas linguagens foram criadas (exemplo: Algol, Basic, Pascal, C) e, gradualmente, elas evoluíram para linguagens de propósito geral. Nos anos 90 surgiram as linguagens orientadas a objetos que admitem um estilo de programação no qual um programa consiste em uma coleção de objetos que interagem uns com os outros. São exemplos dessas linguagens: C++, Java e Ruby. No entanto, elas ainda são linguagens de propósito geral. A pergunta "Qual a melhor linguagem de programação?" esteve sempre presente nas discussões e eventos científicos da comunidade de computação e informática, porém nunca se chegou a uma resposta determinística e, a frase "depende da aplicação" passou a fazer parte das respostas da maioria dos pesquisadores. A Figura 1 apresenta a Hierarquia de Chomsky e suas inclusões próprias, com destaque para a classe das linguagens livre de contexto, vez que elas permitem uma representação simples da sintaxe, que é adequada tanto para a estruturação formal quanto para a análise computacional.

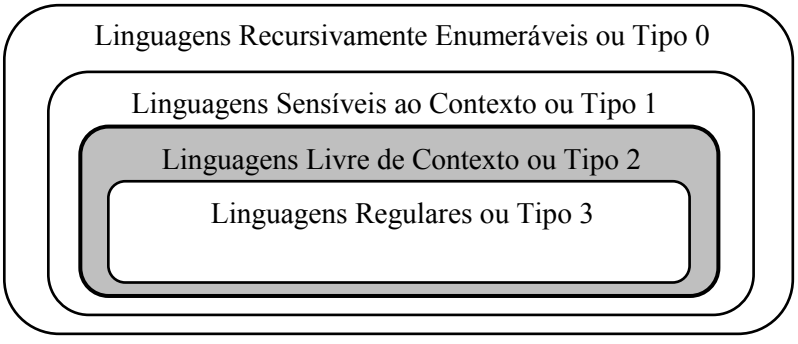


Figura 1. Hierarquia de Chomsky

Atualmente, em vez de buscar uma linguagem para ser a melhor na solução de qualquer tipo de problema computacional, as **linguagens de domínio específico (do inglês: *domain-specific languages – DSL*)** têm sido a solução para resolver uma classe específica de problemas e, dessa forma, elas estão se tornando mais acessíveis para o público geral do que as linguagens de programação tradicionais [Taha, 2008].

Este trabalho tem como objetivo discutir o uso de DSLs na solução de problemas computacionais e apresentar a ferramenta ANTLR e sua utilização na construção de tradutores e interpretadores de um modo geral. O restante desde trabalho está organizado da seguinte forma: Seção 2 apresenta uma visão geral das linguagens de domínio específico, destacando o estado da arte e as principais tecnologias usadas para reconhecer DSLs. A Seção 3 descreve em detalhes a ferramenta ANTLR que vem sendo utilizado principalmente para construção de tradutores e interpretadores para DSLs. A Seção 4 apresenta um estudo de caso com o exemplo do uso da ferramenta ANTLR para a construção de reconhecedores de linguagens. Finalmente, a Seção 5 conclui o trabalho e destaca alguns trabalhos futuros que podem ser conduzidos como temas em cursos de pós-graduação.

5.2. Visão Geral das DSLs

Na literatura ainda não existe uma definição precisa do que é uma linguagem de domínio específico (DSL), porém usaremos a definição de que DSLs são micro-linguagens de programação usadas para resolver tarefas específicas onde o domínio da aplicação é bem definido. São exemplos de problemas específicos: formatação de dados, formatação de arquivos de configuração, protocolos de redes, busca e recuperação de documentos ou informações a partir de padrões textuais, entre outros. As DSLs são importantes para o desenvolvimento de *software* porque representam um meio mais natural, robusto e manutenível do que simplesmente escrever *software* em uma linguagem de propósito geral [Parr, 2007]. A vantagem de criar DSLs é que as soluções podem ser apresentadas no idioma e no nível de abstração do domínio do problema e, com isso, especialistas no domínio podem entender, modificar, validar e até fazer programas mais facilmente. Portanto, o *gap* existente entre os profissionais da área de domínio do problema (clientes) e os profissionais da área de tecnologia da informação (profissionais de TI) é minimizado.

O exemplo mais conhecido de DSL é a linguagem de expressões do aplicativo *EXCEL* da Microsoft. Outros exemplos incluem: *AutoCAD* para projeto de arquitetura, *ProEngineer* para modelagem mecânica, *Verilog* para descrição de *hardware, Matlab* para cálculo numérico, *Lex e Yacc* para geração de analisadores léxicos e sintáticos, *SQL* para consulta a banco de dados, *HTML* para descrição de páginas Web e até mesmo construtores de *GUI* (Graphics User Interface) podem ser vistos como uma DSL.

As DSLs são usualmente pequenas e oferecem somente um conjunto restrito de notações e abstrações e podem ser classificadas em **externas** e **internas**. **DSLs externas** são escritas em uma linguagem diferente da linguagem principal da aplicação e são transformadas nesta usando alguma forma de compilador ou interpretador. A principal vantagem é a possibilidade de expressar o domínio em uma forma mais fácil para ler e modificar. Uma desvantagem óbvia é que você tem que construir o tradutor, mas ferramentas como ANTLR ajudam bastante nesta atividade. Entretanto, a principal desvantagem das DSLs externas é que elas falham na **integração simbólica**, pois a DSL não é realmente ligada à linguagem base da aplicação. O ambiente da linguagem base não tem conhecimento do que se está fazendo.

As **DSLs internas** transformam a linguagem base em uma própria DSL. Dessa forma elas eliminam o problema da integração simbólica. Tem-se também todo o poder de expressão da linguagem base disponível e do conjunto de ferramentas existentes. A principal desvantagem está na expressividade dos mecanismos simbólicos limitados à sintaxe e estrutura da linguagem base. Lisp e modelos de objetos adaptativos são exemplos de DSLs internas. Outros exemplos desta abordagem são: uma linguagem de controle de robô embutida em Haskell e uma linguagem de desenho *PIC-like* embutida em ML.

Quanto à metodologia para projetar uma DSL, utiliza-se três etapas, a saber:

- **Análise:** compreende os seguintes passos: i) identificar o domínio do problema; ii) obter o conhecimento relevante sobre esse domínio; iii) juntar esse conhecimento em noções semânticas e operações sobre elas – combinadores; e iv) projetar uma DSL que descreve concisamente aplicações no domínio;
- **Implementação**: têm-se os passos: v) construir uma biblioteca (API) ou *framework* que implementa as noções semânticas; e vi) projetar e implementar um compilador (ou interpretador) que traduz (ou interpreta) programas na DSL em uma seqüência de chamadas a bibliotecas;
- **Utilização**: deve-se: vii) escrever programas na DSL para todas as aplicações projetadas e compilá-las.

Existem várias ferramentas para suporte a criação de DSLs, conhecidos na literatura como ***language workbenches***. São exemplos dessas ferramentas:

- Intentional Software [IntentSoft, 2010]

  Usa uma abordagem radicalmente nova para criação de *software*. O conhecimento do especialista no domínio do problema é registrado em uma forma que pode ser transformado diretamente em código executável, acelerando grande parte do processo de criação de *software*.
- Microsoft Visual Studio 2010 DSL SDK [VisualStudio, 2010]

  Fornece ferramentas e *templates* para construir projetos gráficos, baseados em formulários de linguagens de domínio específico para o Visual Studio 2010.
- The Meta Programming System [MPS, 2010]

Plataforma universal para projetar linguagens de domínio específico junto com o suporte a ferramentas e ambientes, desenvolvida pela JetBrains (http://www.jetbrains.com). Permite aos programadores definirem linguagens tão facilmente quanto eles podem escrever programas.

- ANTLR Parser Generator [AntLR, 2010]

  Fornece um framework para construir reconhecedores, interpretadores, compiladores e tradutores a partir de descrições gramaticais contendo ações em uma variedade de linguagens destino.

Várias iniciativas têm surgido na área de criação das *language workbenches*. A área é tão "quente" que em 2011 deverá ser realizado a *Language Workbench Competition 2011* (ver http://www.languageworkbenches.net/).

Para mais informações sobre DSLs o artigo [vanDeursen et al., 2000] consiste em uma boa referência, pois os autores fornecem um *survey* da literatura existente, cobrindo tópicos sobre terminologia, riscos e oportunidades, exemplos de DSLs e questões sobre projeto e implementação, listando referências relevantes para cada tópico. Outra referência bastante atual sobre DSLs é o livro "Domain Specific Languages" de Martin Fowler [Fowler, 2010]. Neste trabalho, iremos focar apenas na ferramenta ANTLR que será descrita em detalhes na próxima seção.

5.3. A Ferramenta ANTLR

O ANTLR (*ANother Tool for Language Recognition*) é uma ferramenta desenvolvida por Terrence Parr, utilizada na construção de compiladores, tradutores, interpretadores e reconhecedores. Atualmente na versão 3, o ANTLR é um software livre, lançado sobre a licença BSD, que vem sendo utilizado principalmente para construção de tradutores e interpretadores para Linguagens de domínio específico. Seu download pode ser feito em http://www.antlr.org. Além do ANTLR, pode-se obter o ANTLRWorks, uma IDE específica para edição e debug, e plug-ins para IDEs já conhecidas como Eclipse, Netbeans e Visual Studio.

O ANTLR utiliza um analisador sintático descendente recursivo LL(*) para reconhecer linguagens especificadas através de gramáticas livres de contexto, expressadas através de EBNF (*Extended Backus-Naur Form*).O ANTLR suporta várias linguagens-alvo, como Java, C, Python e outras.

Uma gramática é uma sequência de regras que descreve a estrutura de uma linguagem especificada. A notação de gramática utilizada pelo ANTLR é derivada da EBNF e baseada no YACC (http://dinosaur.compilertools.net/yacc/index.html), onde regras começam com letra minúscula e *tokens* com letra maiúscula. O ANTLR trabalha com quatro tipos de gramáticas: léxica (*lexer*), sintática (*parser*), árvore (*tree*) e combinada (léxica + sintática).

A Figura 2 mostra o processo completo de processamento de uma linguagem. Nesse trabalho, será explicado o processo até a construção da gramática de árvore.

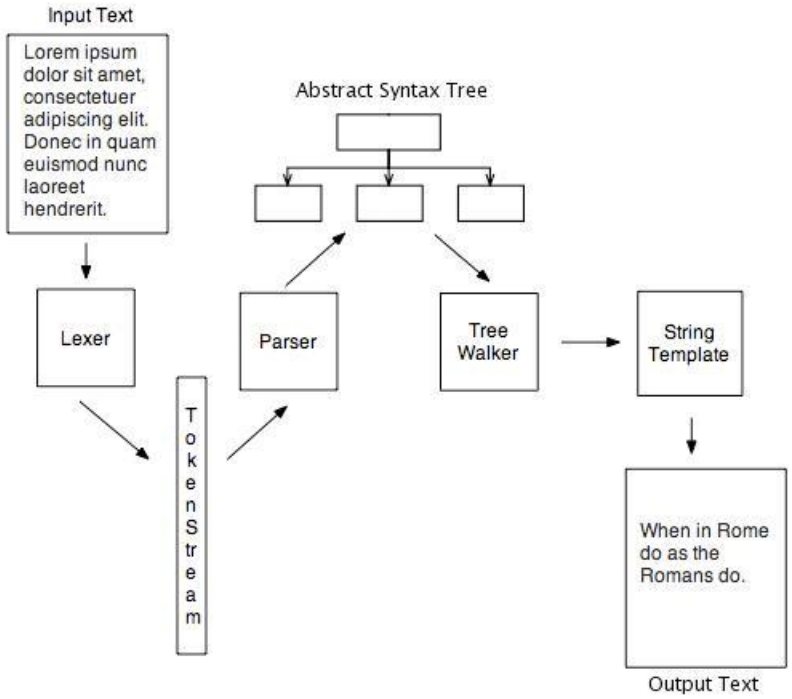


Figura 2: Diagrama de fluxo de dados do ANTLR

Uma gramática léxica (*lexer grammar*) tem como função realizar a análise léxica que consiste em dividir a *stream de entrada* em *tokens*. Uma gramática sintática (*parser grammar*) tem como função realizar a analise sintática, que consiste em processar uma *stream de tokens* e verificar se estão sintaticamente corretos de acordo com a gramática.

5.3.1. Criando a gramática

Vamos criar uma gramática de exemplo que tem as seguintes características:

- Definição de variáveis (identificador letra seguido de um ou mais letras ou dígitos)
- Operação de adição (+) e subtração (-)
- Números inteiros
- Impressão do valor de uma variável.

Alguns exemplos de expressões válidas na nossa linguagem:

```
inteiro var;
var = 1;
var = 10 + var;
imprime var;
```

Vamos começar definindo nossa gramática combinada, chamando-a ***gramatica1***. Uma gramática de nome `<gramatica1>` deve estar definida em um arquivo de nome `<gramatica1.g>`.

```
grammar gramatica1;
```

A regra inicial dessa gramática será chamada ***prog***, e será composta de um ou mais comandos (***stat***). As regras de uma linguagem são um conjunto de sentenças válidas de uma linguagem. Uma regra então define uma frase ou subestrutura. Cada regra contém uma ou mais alternativas. Uma alternativa pode referenciar outras regras e, se uma regra a referencia ela é considerada recursiva.

```
prog: stat+;
```

Uma regra sintática é definida por letras minúsculas. As regras suportam alguns operadores de expressões regulares, definidos na Tabela 1.

Operador	Descrição
()	Cria um grupo. A um grupo pode ser atribuído um *label* ou operador
.	Coringa. Casa com qualquer regra (sintática) ou *token* (léxico).
?	Opcional
*	Zero ou mais
+	Um ou mais
|	Alternativa
~	Negação
..	Intervalo entre dois valores

Tabela 1: Operadores de expressões regulares

Um ***stat*** pode ser uma impressão, atribuição ou uma definição de variável, seguido de ponto e virgula.

```
stat: (INTEIRO ID |
       ID IGUAL expr |
       IMPRIME ID) PONTOEVIRGULA;
```

No exemplo acima podemos ver a utilização de *tokens*. Um *token* é a menor unidade atômica de uma gramática. Uma entrada é dividida em *tokens* através de uma gramática léxica. Um *token* é definido por letras maiúsculas.

O campo *tokens* permite a nomeação de literais e criação de *tokens* imaginários. Um literal é uma sequência de caracteres entre aspas simples. Os literais têm suporte a Unicode, utilizando-se os respectivos valores em hexadecimal. A nomeação de literais permite dar um nome mais descritivo que o literal, como por exemplo, utilizar MOD como apelido para '%'. *Tokens* imaginários são *tokens* que não estão associados a nenhum caractere da entrada, e são muito úteis na criação de árvores de sintaxe abstrata (do inglês: ***Abstract SyntaxTree – AST***).

```
tokens {
        VARDEF;
        IGUAL = '=';
        PONTOEVIRGULA = ';';
        INTEIRO = 'inteiro';
        IMPRIME = 'imprime';
        MAIS = '+';
        MENOS = '-';
       }
```

Vamos agora definir os *tokens* ID e INT através de uma regra léxica. Um identificador na nossa linguagem é uma letra seguida por zero ou mais letras ou dígitos. Um inteiro é uma sequência de um ou mais dígitos.

```
ID  : ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')* ;
INT : '0'..'9'+;
```

Para obter mais legibilidade e reutilização, vamos criar as regras LETRA e DIGITO e substituí-las nas regras léxicas acima. Como essas regras não podem ser casadas individualmente, vamos declará-las com o modificador *fragment*, que permite definir fragmentos que não podem ser utilizados isoladamente.

```
fragment LETRA: 'a'..'z' | 'A'..'Z';
fragment DIGITO: '0'..'9';
```

As regras ID e INT então ficariam:

```
ID: LETRA (LETRA | DIGITO)*;
INT: DIGITO+;
```

Um problema surge quando há um excesso de espaços ou quebras de linha, para isso é preciso definir o *token* WS (*White space*), que ignora esse tipo de entrada. Existem duas formas de se fazer isso. A primeira é colocando o *token* em um canal oculto. O analisador só passa por um canal, ignorando os demais. O canal padrão é 0 (*DEFAULT*) e o canal escondido padrão é *HIDDEN*. A segunda forma é utilizar a função pré-definida `skip()`, que não gera o *token*. A primeira opção consome mais memória, mas é útil para quem deseja criar um debug para o usuário da linguagem, informando exatamente onde ocorreu o erro, através dos atributos pré-definidos *line*, que mostra a linha onde foi encontrado o *token* e *pos*, que mostra a coluna onde foi encontrada o *token*. Outro atributo pré-definido importante é o *text*, que informa o texto que casou o *token*. Atributos podem ser acessados dentro de ações através do operador $.

```
WS : ( ' '  |
       '\t' |
```

```
        '\r' |
        '\n' ) { $channel=HIDDEN; }
      ;
```

No nosso exemplo, uma expressão é uma operação aritmética simples de inteiros ou variáveis, envolvendo apenas os operadores mais e menos. Pode-se ter desde um simples operando até uma sequência de adições e subtrações.

```
expr : atom ((MAIS| MENOS) atom)*;
atom : INT | ID;
```

É preciso ter cuidado ao definir regras que se auto-referenciem, pois elas podem levar a um *loop* infinito, através de recursividade à esquerda, como no exemplo abaixo:

```
expr : expr((MAIS| MENOS) atom)*;
```

O ambiente ANTLRWorks que incorpora a ferramenta ANTLR é usada para facilitar a construção de reconhecedores de linguagens. Ele identifica possíveis *loops* na gramática e sugere modificações que eliminam tais problemas.

5.3.2. Ações

Agora que a gramática está pronta e que é possível realizar a análise léxica e sintática, vamos produzir a saída desejada. Essa saída será gerada através de ações semânticas. Ações são pedaços de códigos entre chaves escritos na linguagem-alvo que podem ser embutidos na sua gramática. Uma ação pode ser inserida antes da classe gerada, através da palavra reservada ***@header***, dentro das definições de atributos, através da palavra ***@members*** e de acordo com a sua posição na gramática, quando definidas dentro de uma regra.

Para salvar os valores atribuídos a um identificador, vamos criar uma tabela *hash*, através da classe Java *HashMap*. Na área da palavra ***@header***, adicionamos os `imports` necessários e no campo ***@members*** definimos o nosso atributo `memory` que irá guardar os identificadores declarados e o respectivo valor. Caso seja necessário adicionar importações no código do analisador léxico, utiliza-se a palavra ***@lexer::header***. Assim, nosso código torna-se:

```
@header {
  Import java.util.Map;
  Import java.util.HashMap;
}

@members {
  HashMap <String, Integer> memory =
                 new HashMap <String, Integer>();
}
```

Agora podemos através de ações, guardar os valores atribuídos as variáveis na memória e executar as operações de adição e subtração. Para recuperar os valores inteiros ou de variáveis de regras abaixo, utilizaremos **atributos herdados**. **Atributos herdados** passam informações *top-down*, ou seja, de uma regra superior para uma regra inferior, enquanto os **atributos sintetizados** passam informações *bottom-up* na gramática, ou seja, valores das regras inferiores podem ser usadas nas regras superiores. Para utilizar atributos herdados define-se argumentos na regra utilizando-se nome_da_regra [argumentos]. Já os atributos sintetizados são definidos através de valores de retorno utilizando nome_da_regra returns [valores_de_retorno]. Veja exemplo abaixo:

```
stat: ( INTEIRO ID { memory.put($ID.text, 0); }
      | ID IGUAL expr { memory.put($ID.text,
                    new Integer($expr.value)); }
      | IMPRIME ID {
              Integer v = memory.get($ID.text);
              if (v != null)
                  System.out.println(v);
              else
                System.out.println("ID não encontrado");
            }
      { }
      ) PONTOEVIRGULA ;

expr returns [int value] :
      a=atom { $value = $a.value; }
      ( MAIS e= atom { $value += $e.value; } |
        | MENOS e=atom { $value -= $e.value; }
      )*
      ;

atom returns [int value] :
      INT { $ value = Integer.parseInt($INT.text); }
      | ID {
              Integer v = memory.get($ID.text);
              if (v != null)
                  $value = v.intValue();
              else $value = 0;
            }
      ;
```

Atributos de regras e *tokens* podem ser chamados dentro de ações, através do operador $. Caso haja mais de uma definição da regra ou *token* dentro da regra, é necessária a utilização de *labels* para especificar a qual regra ou *token* estamos nos referindo. O exemplo abaixo demonstra como seria a definição de uma regra de condicional.

```
ifStmt : 'se' exprBool 'entao' e1=expr 'senao' e2=expr;
```

Também pode utilizar um *listLabel*, que salva uma lista de elementos através do operador +=. O exemplo abaixo demonstra como fazer uma declaração de mais de uma variável utilizando essa estrutura.

```
defVariaveis: INTEIRO ids+=ID (VIRGULA ids+=ID)*
                    PONTOEVIRGULA {$ids}
            ;
```

Um escopo é uma ação que permite definir um escopo dinâmico. Para demonstrar a utilização do escopo, segue um exemplo utilizando um escopo para criar uma tabela de símbolos apenas em uma determinada regra.

```
defVariaveis scope { List symbols; } :
        INTEIRO ID PONTOEVIRGULA;
```

Para criar escopos globais, basta definir um escopo nomeado fora da área de uma regra, como:

```
scope Escopo{
     List symbols;
}
```

5.3.3. Árvore de sintaxe abstrata e gramática de árvore

Árvores de sintaxe abstrata (AST) é uma representação intermediária utilizada pelo ANTLR e consiste em uma versão condensada da entrada, apenas com informações relevantes, removendo *tokens* como ponto-e-vírgula e delimitadores do tipo abre e fecha parênteses que não contêm dados relevantes para processamento. Além de conter apenas os *tokens* relevantes, uma AST deve ser fácil para o computador reconhecer e navegar.

ASTs geralmente são confundidas com árvores sintáticas (*parse tree)*, mas é diferente desta, pois enquanto uma *parse tree* representada toda a *stream* de entrada, a AST contém apenas *tokens* relevantes. A Figura 3 mostra uma *parse tree* e a AST equivalente para uma determinada entrada.

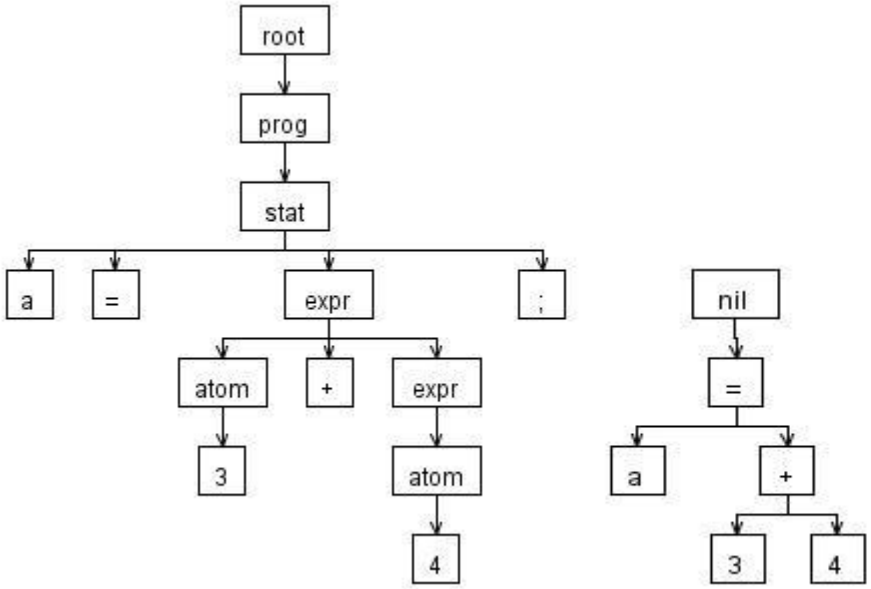


Figura 3: Parse Tree (esquerda) x AST (direita)

Para gerar uma AST a partir de uma gramática sintática ou combinada, especifica-se no campo ***options*** a opção "*output=AST"*. Pode-se utilizar operadores ou regras de reescrita para especificar o formato da AST gerada. Operadores para gerar AST são semelhantes aos operadores de expressões regulares, e são colocados após as regras. A Tabela 2 mostra os operadores de uma AST.

Operador	Descrição
!	Não gera símbolo na AST.
^	Cria um novo nó pai na AST.

Tabela 2: Operadores usados para gerar uma AST.

A maneira recomendada de se definir uma AST é utilizando regras de reescrita, através do operador ->. O seguinte formato é utilizado:

```
^(pai filho1 ...filhoN)
```

O exemplo abaixo mostra as regras de reescrita para gerar uma AST para a gramática de nosso exemplo.

```
prog: stat+ -> stat+;
stat: (INTEIRO ID       -> ^(VARDEF INTEIRO ID)
      | ID IGUAL expr   -> ^(IGUAL ID expr)
      | IMPRIME ID      -> ^(IMPRIME ID)
      ) PONTOEVIRGULA;
expr: atom ((MAIS | MENOS)^ atom)*;
atom: INT | ID ;
```

Uma gramática de árvore (*tree grammar*) é uma gramática que recebe uma AST como entrada e realiza o processamento desta. No campo *options* a opção ***tokenVocab=nome_da_gramatica*** permite utilizar *tokens* definidos por outra gramática.

Para construir uma gramática de árvore, é possível reusar as regras de reescritas definidas na gramática sintática (ou combinada). O exemplo abaixo mostra a criação da regra de definição de variáveis na gramática de árvore a partir da regra de reescrita da gramática combinada.

```
<<gramática combinada>>
    stat: INTEIRO ID -> ^(VARDEF INTEIRO ID);

<<gramática de árvore>>
    stat: ^(VARDEF INTEIRO ID);
```

Ás vezes é preciso fazer uma pequena alteração nessa transição, como no exemplo de uma regra de condicional SE-SENAO onde o senão é opcional.

```
<<gramática combinada>>
    se : 'se' exprBool 'entao' e1=expr
     ('senao' e2=expr -> ^('se' exprBool $e1 $e2)
      | -> ^('se' exprBool $e1)
     );

<<gramática de árvore>>
    se: ^('se' exprBool expr expr?) ;
```

Com a gramática de árvore gerada, a próxima etapa é utilizar *StringTemplate* para gerar uma saída desejada. Um *template* é basicamente uma *string* com buracos onde se pode colocar conteúdo desejado. A discussão sobre o uso de *templates* está fora do escopo deste trabalho, porém, exemplos de *StringTemplate* estão disponíveis em http://stringtemplate.org.

5.4. Estudo de Caso

Nesta seção apresentamos um estudo de caso desenvolvido no contexto do projeto de pesquisa “Ambiente de Geração de Diagramas UML a partir de Descrições em Linguagem Natural”, desenvolvido no Laboratório de Engenharia de Software e Informática Industrial (EaSII) da UFPI. O protótipo desenvolvido recebe uma descrição textual qualquer escrita em português e, após analisar alguns padrões textuais pré-definidos, gera as possíveis classes, atributos e métodos. A Figura 4 mostra uma visão geral do processo.

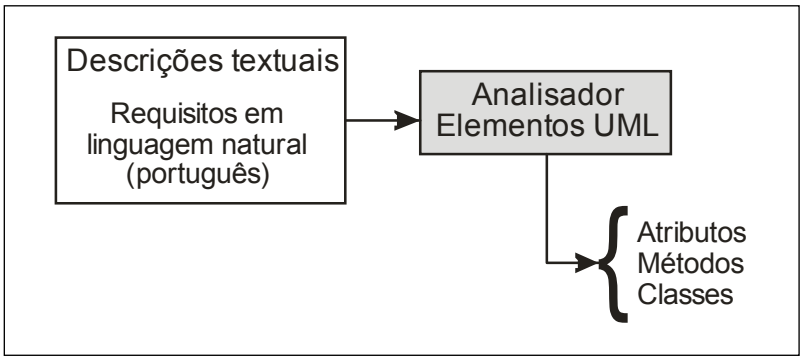


Figura 4. Elementos UML a partir de descrições textuais: visão geral.

O protótipo possui uma interface bastante simples, com opções para criar, abrir e salvar um documento, além de uma opção para executar a ferramenta de análise. Para a separação das palavras (*tokens*) e para a especificação dos padrões gramaticais foi usada a ferramenta ANTLR em conjunto com o ambiente NetBeans IDE. Após o processo de separação das palavras (fase léxica), faz-se o processo de etiquetagem das palavras consultando uma estrutura do tipo *HashMap* que armazena o dicionário de palavras. Com as palavras classificadas, a fase de análise sintática é executada para identificar os possíveis padrões gramaticais da descrição. Os padrões identificados são apresentados ao usuário na forma de listas com as possíveis classes, atributos e métodos da descrição.

A Figura 5 mostra as regras léxicas usadas para identificar os *tokens* do arquivo texto e a Figura 6 apresenta os padrões textuais usados para especificar as regras gramaticais dos elementos UML usados. Nestas figuras a sintaxe utilizada está de acordo com a notação EBNF (*Extended Backus Naur Form*) usada pelo ANTLR.

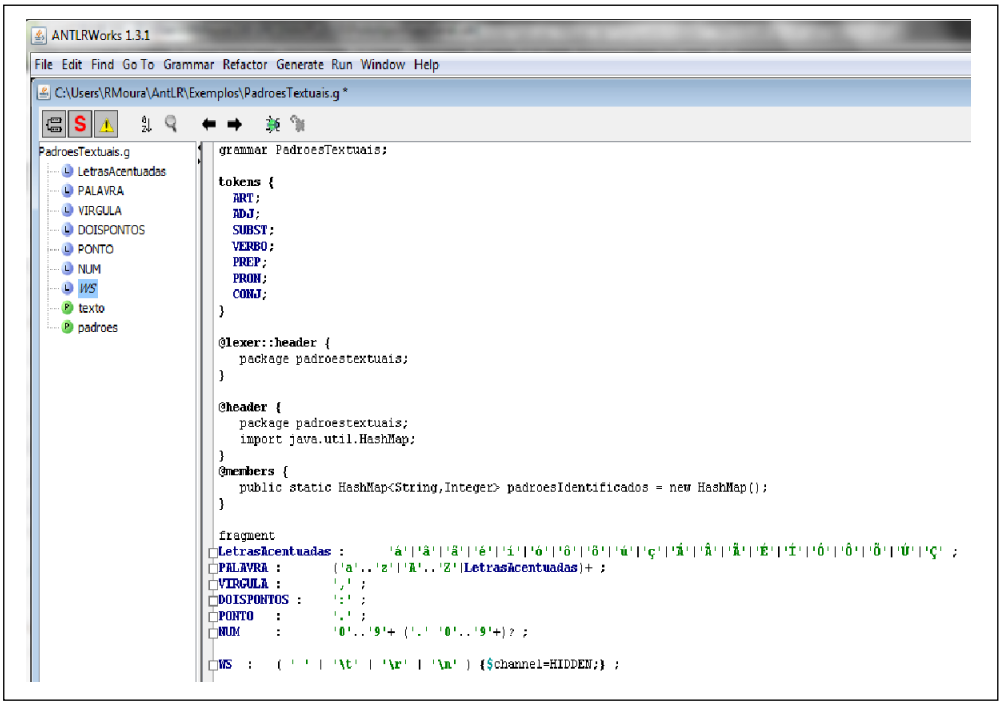


Figura 5. Elementos UML: regras léxicas para separar tokens.

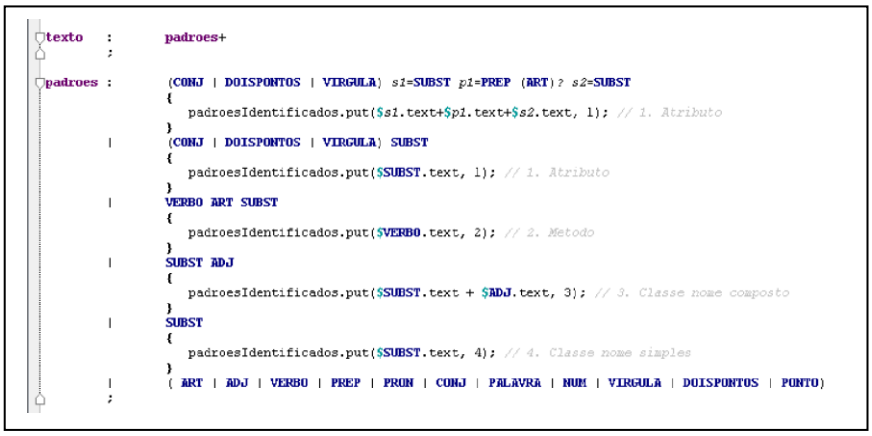


Figura 6. Elementos UML: padrões textuais.

Vale ressaltar que as descrições textuais, especificadas em linguagem natural (português), podem conter acentuação normal. Esta limitação foi resolvida através da regra léxica:

fragment
LetrasAcentuadas :
'á'|'â'|'ã'|'é'|'í'|'ó'|'ô'|'õ'|'ú'|'ç'|'Á'|'Â'|'Ã'|'É'|'Í'|'Ó'|'Ô'|'Õ'|'Ú'|'Ç' ;

Os *tokens* definidos na parte inicial da figura 5, a saber: ART, ADJ, SUBST, VERBO, PREP, PRON e CONJ são usados para definir os padrões gramaticais usados para identificar os elementos da UML, usados nas regras sintáticas da gramática. Por exemplo, um possível atributo de uma classe UML é identificado através dos padrões:

(CONJ | DOISPONTOS | VIRGULA) s1=SUBST p1=PREP (ART)? s2=SUBST

```
{
   padroesIdentificados.put($s1.text+$p1.text+$s2.text, 1); // 1. Atributo
}
```

(CONJ | DOISPONTOS | VIRGULA) SUBST

```
{
   padroesIdentificados.put($SUBST.text, 1); // 1. Atributo
}
```

Um método é identificado através do padrão:

VERBO ART SUBST

```
{
   padroesIdentificados.put($VERBO.text, 2); // 2. Método
}
```

As classes UML podem ser identificadas com nome simples ou composto através dos padrões:

SUBST ADJ

```
{
   padroesIdentificados.put($SUBST.text + $ADJ.text, 3); // 3. Classe nome composto
}
```

SUBST

```
{
   padroesIdentificados.put($SUBST.text, 4); // 4. Classe nome simples
}
```

Em todas as regras, uma ação para armazenar o elemento UML identificado em uma tabela de símbolos é executada. Finalmente, um padrão gramatical sem nenhuma ação associada é definido com o objetivo de desconsiderar os trechos de texto que não casam com nenhum dos padrões descritos na gramática. Esta regra é:

(ART | ADJ | VERBO | PREP | PRON | CONJ | PALAVRA | NUM | VIRGULA | DOISPONTOS | PONTO)

É importante destacar que a ordem em que os padrões são definidos é de suma importância para a identificação correta dos elementos UML. Note que se a última regra fosse colocada no início nenhum padrão do tipo método seria reconhecido.

A seguir apresenta-se um exemplo de requisitos de *software* (ver Figura 7a) com os elementos UML identificados pelo protótipo desenvolvido (ver Figura 7b).

Uma loja de CDs possui CDs para venda e para locação.

Um cliente pode comprar ou locar uma quantidade ilimitada de CDs.

Para locar é obrigatório que o cliente esteja cadastrado em a loja, ou seja, tenha preenchido uma ficha de cadastro que deve ser renovada a cada 06 meses.

Em a ficha de cadastro deve constar: nome de o cliente, endereço, data de cadastro, telefone e email.

Figura 7a. Requisitos em linguagem natural.

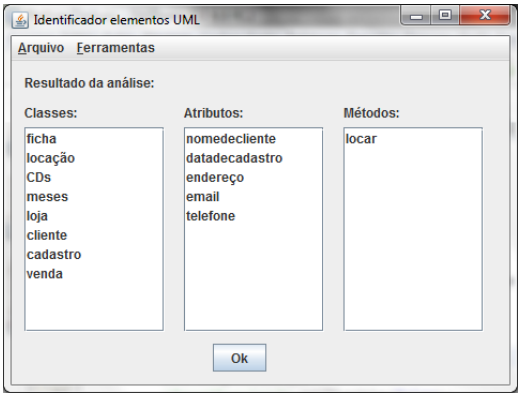


Figura 7b. Resultado da análise.

Na Figura 7b, note que o método **locar** foi encontrado porque o padrão *"Verbo Artigo Substantivo"* foi identificado na descrição textual: "locar uma quantidade". As classes **ficha, locação, CDs, meses, loja, cliente, cadastro e venda** foram identificadas devido ao padrão "*Substantivo*" na descrição. Uma limitação do protótipo desenvolvido é que os atributos e métodos não são associados automaticamente às classes. Esta associação deverá ser feita diretamente pelo usuário. Finalmente, é importante destacar que os elementos UML identificados são apenas sugestões que este tipo de ferramenta apresenta ao usuário. No entanto, a decisão final das classes, métodos e atributos do sistema depende da intervenção do usuário.

Este estudo de caso mostra alguns resultados animadores, porém, muito trabalho ainda tem que ser realizado para atingir o objetivo principal do projeto de pesquisa "Ambiente de Geração de Diagramas UML a partir de Descrições em Linguagem Natural".

5.5. Conclusão e Trabalhos Futuros

Este trabalho mostrou uma visão geral das linguagens de domínio específico (DSL), destacando os principais conceitos da área das DSLs, bem como as principais referências sobre o tema. Em seguida, apresentou-se um tutorial sobre a ferramenta ANTLR focado na construção de um reconhecedor para uma gramática de expressões. Um estudo de caso para identificação de elementos UML a partir de padrões textuais foi utilizado para exemplificar o uso da ferramenta ANTLR em aplicações do mundo real.

A ferramenta ANTLR tem sido bastante utilizada para academia para a criação de tradutores e reconhecedores de linguagens seja de propósito geral ou específico. Na UFPI os alunos de graduação têm usado ANTLR nas atividades práticas da disciplina construção de compiladores, bem como para desenvolver atividades de pesquisa dentro do laboratório de Engenharia de Software e Informática Industrial, sobretudo para reconhecer padrões textuais em descrições de linguagem natural.

Trabalhos futuros sobre o estudo de caso apresentado incluem: i) identificar a mesma classe, atributo e/ou método quando se tratar de palavras no plural e no singular; ii) aprimorar os padrões gramaticais para identificar os elementos da UML; iii) gerar um ambiente gráfico com o diagrama conceitual da UML; iv) melhorar o protótipo e realizar experimentos em cenários reais de utilização; e v) realizar uma comparação dos resultados obtidos com outros trabalhos relacionados.

Como tema para trabalhos futuros, encorajamos o estudo das ferramentas para criação de DSLs, bem como a exploração dessas ferramentas como facilitador do acesso à informação digital no desenvolvimento de sistemas de busca e recuperação de documentos ou informações a partir de padrões textuais. Extrair conhecimento de texto não estruturado e transformá-lo em conhecimento estruturado é o foco central na área de extração da informação.

Referências

AntLR (2010), AntLR Parser Generator. Disponível em: http://antlr.org/

Fowler, M. (2010), Domain Specific Languages. Addison-Wesley Professional, 2010.

IntentSoft (2010), Intentional Software Corporation. Disponível em: http://www.intentsoft.com/

MPS (2010), The Meta Programming System. Disponível em: http://www.jetbrains.com/mps/

Parr, T. (2007), The Definitive ANTLR Reference: Building Domain-Specific Languages. The Pragmatic Programmers. Disponível em: http://www.pragprog.com/titles/tpantlr/the-definitive-antlr-reference.

Taha, W. (2008), Domain Specific Languages. IEEE International Conference on Computer Engineering and Systems (ICCES), 2008.

van Deursen, A., Klint, P. & Visser, J. (2000), Domain-specific languages: an annotated bibliography. ACM SIGPLAN Not. Volume 35 (6), New York, NY, USA, 2000.

VisualStudio (2010), Visual Studio Visualization and Modeling SDK (was DSL SDK). Disponível em: http://code.msdn.microsoft.com/vsvmsdk.

Capítulo 6

Empreendedorismo e Gestão em TI

Paulo César Coutinho dos Santos

Abstract

Imagine an organization without information technology today is very close to impossible. Even in small business, the technology has been used as a tool that optimizes not only serves as the processes for managing a company. This chapter presents various aspects of the importance of IT, from its conception and implementation in the organization to the management and administration of technological resources as well as to provide subsidies for the design and implementation of entrepreneurial action can be considered both internally and externally in any corporation.

Resumo

Imaginar uma organização sem Tecnologia da Informação hoje é bem próximo do impossível. Mesmo no pequeno negócio, a Tecnologia vem sendo usada como uma ferramenta que não só otimiza os processos como serve para a administração de uma empresa. Este capítulo apresenta diversos aspectos da importância da TI, desde sua concepção e implementação na organização até a gerência e administração de recursos tecnológicos além de apresentar subsídios de como a concepção e implementação desta pode ser considerada ação empreendedora tanto interna como externamente em qualquer corporação.

6.1. A Revolução do Empreendedorismo

"O empreendedorismo é uma livre tradução que se faz da palavra *entrepreneurship*, que contem as idéias de iniciativa e inovação. É um termo que implica uma forma de ser, uma concepção de mundo, uma forma de se relacionar". (DOLABELA, 2006).

O século XX foi caracterizado por várias transformações em curtos períodos de tempo, onde foram desenvolvidas as grandes invenções que mudaram a vida das pessoas. Em sua maioria, essas invenções são frutos de inovação, de algo que nunca existiu ou de um novo olhar dado a coisas já existentes, mas que ninguém antes observou de outra forma.

Por trás dessas invenções, existem pessoas ou equipe de pessoas que possuem características especiais que não se submetem a enxergar apenas o que lhes é permitido enxergar, a identificar oportunidades onde apenas possa existir numa holística muito fácil ou até mesmo observar aquilo que ninguém observou e a partir de então ser privada de experiências. Estas pessoas com características diferenciadas, que possuem motivação singular, tem a necessidade de serem reconhecidas e admiradas pelos seus esforços e não hesitam em deixar sua marca, no qual denomina-se o nome de Empreendedores.

Até a década de 80 o ensino de administração era voltado apenas para formar profissionais para administrar grandes empresas e não para criar empresas, pois os empregos oferecidos pelas grandes empresas nacionais e multinacionais bem como a estabilidade que se consegui nos empregos em repartições públicas, eram muito convidativos, com bons salários, status e possibilidade de crescimento dentro da organização.

A partir de então era considerado loucura jovens recém formados aventurar-se na criação de seu próprio negócio. Com essa crença, com essa visão acomodada, fica fácil entender porque as pessoas, não se sentiam estimuladas a buscar um outro caminho, que gere satisfação e realização, e que muitas vezes só pode ser encontrado através de um negócio próprio que atenda às expectativas mais pessoais e possa se encaixar mais adequadamente à sua vida.

6.1.1. O E-Empreendedor

Na década de 1990, o aparecimento da *World Wide Web*, o desenvolvimento dos *browsers* (navegadores), a diminuição de custos de acesso, o aumento de conteúdos, entre outros fatores, fizeram com que a Internet tivesse um crescimento em larga escala e passasse a ser bastante utilizada para inúmeros fins.

As necessidades e os anseios dos usuários da *web* foram o motor da grande exploração e massificação deste meio. Nelas, o empreendedorismo pode ser pensado ou comparado como na vida real, isto quando se trata de ambientes virtuais. Os empreendedores passaram a ter várias oportunidades para escolher temas dos mais diversos, tendo voz, espaço e oportunidade para dá vida, divulgar, aprimorar suas idéias e/ou suas possibilidades de negócios. Um grande agrupamento de indivíduos, com necessidades, e com um crescente número de adeptos a cada dia, despertou e ainda desperta nos empreendedores e nas empresas o potencial comercial da Internet.

A comercialização na Internet começou seu crescimento em ritmo acelerado em 1991, onde nesta época já havia cerca de 9 mil domínios comerciais e, logo no final de 1994 já existiam 21.700 destes.

"A comercialização do espaço cibernético estará mais próxima da experiência histórica das ruas comerciais emergentes da palpitante cultura urbana que dos shoppings centers espalhados na monotonia dos subúrbios anônimos". (CASTELLS, 2006).

A pergunta que se faz hoje é: Estamos vivendo um novo momento ou reinventando momentos que parecem terem sido já reinventados no passado e mais uma vez estão sendo? Questiona-se isso porque, como tudo muda muito rápido, já se falando em *Web 3.0* onde o empreendedor tem que agir para nunca perder as oportunidades?

Um artigo do *New York Times* sobre a conferência da *Web 2.0* em outubro de 2007, enfatizou que as empresas *Web 2.0* estão nadando em dólares. Além disso, Mary Meeker, do Morgan Stanley Investment Bankers, falou durante a conferência que "as tendências de alto nível da *Web 2.0* são bastante convincentes". John Cisco, presidente CEO da Cisco Systems, prevê que a *Web 2.0,* redes sociais e colaboração *on-line* serão arautos de uma nova explosão de produtividade. (GOOSSEN, 2008).

Graças aos timbres empreendedores dos homens e mulheres nos mais diversos cantos do planeta explodiram no início de 2001 as empresas "pontocom", estas voltadas para a internet, onde 24 horas por dia, 07 dias por semana, 365 dias no ano estão à disposição dos mais diversos tipos de usuários e disponibilizando os mais diversos tipos de serviços, estes desde a venda de produtos à pontos de encontro.

Com base em tudo apresentado anteriormente, em se tratando de empreendedorismo, vale frisar que o empreendedorismo é essencial para tornar a inovação acessível, assim como a inovação é essencial para o empreendedorismo. E graças a isso que hoje a *Web 2.0* é um sucesso e certamente continuará sendo, até a *3.0.*

Para os grandes empreendedores, ou empreendedores que pensam grande, as novas formas, meios ou mídias serão parte fundamental do planejamento de novos projetos para um futuro bem próximo, assumindo importantes papéis como ponto-de-venda, veículo gerador de relacionamento com o consumidor ou *prospect*.

Pode-se citar como exemplo de oportunidade surgida e aproveitada, a onda das empresas de publicidade *on-line*. Estas surgiram para suprir uma necessidade mais que real de fazer com que a publicidade na *web* fosse mais agressiva e logicamente tornasse as organizações mais competitivas, exemplo disso foi a necessidade de se pensar em uma outra opção para o (antigo) *banner* tradicional, de 7,5cm por 2,5cm contendo mensagens curtas e gráficos simples não conseguia passar informações suficientes para uma boa exposição da marca ou produto, o que estimulou ainda mais as empresas investirem em seus próprios sites, pois surgiram outras formas de divulgação. Outros meios alternativos de anúncios também começaram a ser estudados, como no caso da General Motors, primeira montadora em todo o mundo a lançar um novo modelo de automóvel – o Astra – por meio de uma campanha veiculada na *Web* antes de chegar à outros tipos de propagandas e obteve um enorme sucesso, com um aumento considerável de visitas ao seu *site.*

Tudo resumi-se a importância de ser direcionado ao usuário final ou o consumidor *web.* Por suas inúmeras opções e possibilidades de "surfar", a internet tornou-se o determinante indispensável e essencial do negócio focado em usuários: se uma empresa não agrada seus usuários, ela não dura muito. Por outro lado, empresas que entregam o que o usuário quer se beneficiam de um rápido aumento no número de visitantes que seu *site* recebe.

Larry Farrell, especialista em empreendedorismo reconhecido internacionalmente, escreve: "A principal visão que todos os empreendedores devem ter é um quadro claro de um grupo específico de clientes que precisam e estão dispostos a pagar por um conjunto específico de produtos e serviços. Não há nada mais básico para um empreendedor." Farrell explica que um empreendedor de sucesso é quase sempre tanto "uma pessoa voltada a produtos" quanto "uma pessoa voltada a clientes" Farrell conta a história de Walt Disney, "o maior criador de produtos da história da indústria de entretenimento", e conclui dizendo: "O truque é tornar-se apaixonadamente

especializado em seus próprios produtos e consumidores. Eles são as duas idéias mais relevantes para seu negócio." (GOOSSEN, 2008).

Hoje, a rede mundial conta com uma imensa quantidade de *sites* comerciais, sendo que a as estratégias empreendedoras em muitos deles, representa sua principal fonte de renda e sobrevivência, estas advindas desde um simples produtos ou serviços colocados à disposição do cliente, a forma de interação com o mesmo ou até mesmo uma simples propaganda disponibilizada no mesmo. No Brasil, é possível ter uma noção de quão forte tornou-se o meio Internet pelo volume de domínios *.BR* – mais de um milhão até 2007, disponíveis na rede.

6.1.2. O Perfil do Empreendedor Digital

De acordo com a maioria dos autores, empreendedor é alguém que identifica uma oportunidade e cria uma organização para aproveitá-la, mas o processo empreendedor envolve todas as funções, atividades e ações associadas à identificação de oportunidades e a criação de organizações para aproveitá-las.

O empreendedor tem como característica básica o espírito criativo e pesquisador, através do qual mantém constante busca por novos caminhos e novas soluções, sempre amparada na identificação das necessidades das pessoas. Essa é a essência do empresário de sucesso: a busca de novos negócios e oportunidades, e a preocupação sempre presente com a melhoria do produto. Enquanto a maior parte das pessoas tendem a enxergar apenas dificuldades e insucessos, o empreendedor deve ser otimista e buscar o sucesso, a despeito das dificuldades.

Saber conviver com o risco e tirar proveito das oportunidades, são as características mais presentes da atividade empresarial, o que define realmente o perfil do empreendedor. Ao contrário da maioria das pessoas o empreendedor é otimista e persistente. Vê nas ameaças grandes oportunidades e a partir delas criam, ousam e trabalham para construir o sucesso com suas próprias mãos.

Diversas características de personalidade que marcam o perfil do empreendedor podem ser denominadas "qualidades" essenciais ao sucesso do empreendedor e conseqüentemente da empresa.

Mas o que significa Perfil do empreendedor de sucesso? Por que identificá-lo?

O perfil do empreendedor é baseado num conjunto de fatores de comportamentos e atitudes que contribuem para o sucesso. Estes conjuntos podem variar de lugar para lugar. E a identificação do perfil do empreendedor de sucesso é feita para que se possa aprender a agir, adotando comportamentos e atitudes adequadas. Mas é importante ter consciência de que ainda não se pode estabelecer uma relação absoluta de causa e efeito. Ou seja, se uma pessoa tiver tais características, certamente vai ter sucesso. O que se pode dizer é que, se determinada pessoa tem as características e aptidões mais comumente encontradas nos empreendedores, mais chances terá de ser bem sucedida.

A literatura sobre empreendedorismo tem a particularidade de reunir idéias de diversos estudiosos das ciências humanas para explicar o processo de empreender e estabelecer o perfil dos empreendedores de sucesso, mas parece que nenhuma delas consegue explicar como e por que as novas organizações são fundadas. Isso talvez se

justifique pelo fato de que os empreendedores não representam um grupo homogêneo e surgem em contextos sociais diferentes e, por isso, os modelos propostos não conseguem abarcar as diferentes formas de operar um negócio e torná-lo bem-sucedido.

As teorias de base econômica que atribuem ao empreendedorismo fator importante para desencadear processos de crescimento e desenvolvimento econômico associaram o empreendedor à inovação, à criatividade e à capacidade de assumir riscos moderados.

O empreendedor como o principal agente do desenvolvimento interfere no equilíbrio ou inércia do mercado, provocando mudanças capazes de gerar prosperidade econômica.

6.2. Identificando e Analisando Oportunidades

Identificar uma oportunidade é o primeiro passo para uma pessoa que quer se tornar um empreendedor, e é primordial saber diferenciar oportunidade de idéia. Segundo o Dicionário Aurélio "oportunidade é ocasião, ensejo, lance, circunstância adequada ou favorável, conveniência". Por sua vez, idéia "é a representação mental de uma coisa concreta ou abstrata, elaboração intelectual, concepção" (FERREIRA, 1999).

A oportunidade para o empreendedor é a concepção da idéia, a idéia por si só não produz se não tiver a ocasião para pô-la em prática. Segundo Timmons (1994 apud DOLABELA, 1999) "uma idéia não passa de um instrumento na mão de um empreendedor. Da mesma forma como um pincel pode gerar uma obra-prima ou medíocre, dependendo das mãos que o utilizam, uma idéia depende de um bom empreendedor para se transformar em um negócio de sucesso".

Uma idéia por si só não é suficiente para a criação de um negócio de sucesso e que os empreendedores devem sempre buscar a identificação de oportunidades. Mas como fazer isso? Quais os passos? Ou, qual o melhor caminho a seguir?

Obviamente que antes de criar um negócio deve-se fazer um estudo mais criterioso de investimento e retorno, de análise da concorrência e, principalmente, do mercado potencial. Mas o início de tudo se dá pela oportunidade identificada e, a partir daí, para uma análise mais detalhada. Nunca se começa um negócio sem antes passar pelas fases de identificação da oportunidade e de planejamento e análise de viabilidade.

É muito importante fazer uma Avaliação da Idéia de Negócio, no sentido de perceber se esta poderá constituir-se numa oportunidade de negócio. A análise de oportunidades é feita através de diversas ferramentas, e uma delas é conhecida com 3Ms que serve para o empreendedor analisar o potencial da oportunidade e quais oportunidades são as mais interessantes.

Os 3Ms são definidos como "Demanda de Mercado", "Tamanho e Estrutura do Mercado" e " Análise de Margem"(DORNELAS, 2003).

Ao analisar o primeiro "M", Demanda de Mercado, o empreendedor deve procurar responder as seguintes questões:

- Qual a audiência-alvo?
- Qual a durabilidade do produto/serviço no mercado?(ciclo de vida)
- Os clientes estão acessíveis?
- Como os clientes vêem o relacionamento com a minha empresa?

- O potencial de crescimento deste mercado é alto?
- O custo de captação do cliente é recuperável no curto prazo?

O segundo "M", Tamanho e Estrutura do Mercado está relacionado a outras questões críticas:

- O mercado está crescendo, é emergente? É fragmentado?
- Existem barreiras proprietária de entrada? Ou excessivos custos de saída?
- Quantos competidores/empresas-chave estão no mercado? Eles controlam a propriedade intelectual?
- Em que estágio do ciclo de vida está o produto?
- Qual é o tamanho do mercado em R$ e o potencial para se conseguir uma boa participação de mercado?
- Como a indústria está segmentada, quais são as tendências, que eventos influenciam os cenários?

Já no terceiro "M" de Análise de Margem aplicam-se as seguintes questões e atividades:

- Determine as forças do negócio;
- Identifique as possibilidades de lucro;
- Analise os custos;
- Mapeie a cadeia de valor do negócio;
- Procure saber como seu produto/serviço chega até o cliente final.

Os 3Ms são abrangentes e envolvem questões críticas que, se respondidas e bem entendidas, com certeza serão úteis na avaliação e seleção das melhores oportunidades para serem desenvolvidas e capitalizadas pelo empreendedor.

6.3. Negócios Digitais

Com o aumento de usuários com Internet banda larga e velocidades bem acima de 300kbps, especificamente no Brasil, novos métodos de comunicação e troca de arquivos se popularizaram no meio virtual. Paralelamente a isso, a *Web* cresceu e buscou cada vez mais uma participação mais efetiva da audiência. O termo "*Web 2.0*" pode ser definido como uma segunda geração da *World Wide Web*. Sua principal característica é a potencialização das formas de publicação, compartilhamento e organização da informação, além de ampliar os espaços para interação entre os participantes. "A *Web 2.0* refere-se não apenas a uma combinação de técnicas informáticas [...], mas também a um determinado período tecnológico, a um conjunto de novas estratégias mercadológicas e a processos de comunicação mediados pelo computador" (PINHO, 2000).

A idéia é de que o ambiente *on-line* torne-se mais dinâmico e os usuários colaborem com a organização dos conteúdos. É o caso do *Youtube*, *Wikipédia*, *Google Docs, Orkut*, *Meebo*, *Bloglines, Dropbox* e *Voo2Doo*, que representam alguns desses *sites* de compartilhamento potencializado e viraram mania em todo o mundo. A definição de uma "segunda geração" *Web* trata-se, para muitos, apenas de um golpe de

marketing, fruto de sua "origem" em um congresso chamado *Web 2.0,* promovido pelo editor de livros Tim O'Reily.

A Internet sempre apresentou características interativas, o reforço deste aspecto seria um movimento natural do meio, não oferecendo motivos para a denominação dessa fase como uma "segunda geração". Tal fato exigiria que houvesse um rompimento de todas as idéias e características da "primeira geração", o que não aconteceu.

A Internet nada tem a ver com marketing – ou mercado – de massa. Ela é sobre pessoas, indivíduos com aspirações, necessidades, desejos e base cultural únicos. Não se trata de um mercado de 60 milhões de pessoas, mas de 60 milhões de mercados de uma só pessoa. (VASSOS, 1997).

Sites voltados ao social como o *Orkut*, *LinkedIn*, *MySpace*, *Youtube* e o *Facebook e, atualmente o Twitter,* fizeram com que meados de 2006 e 2009 fosse denominado o ano da "*Web* social" em decorrência do enorme sucesso desse gênero. Também foi observado um avanço na área tecnológica da Internet, com a expansão e disseminação dos "*blogs*" e "*wikis*" de todos os assuntos possíveis. Com mais de 600 milhões (IBOPE/NETRANKING, 2010) de usuários de Internet hoje, seu crescimento foi grandioso e continua guiado, entre outras razões, pela difusão e o estímulo do acesso à Rede de forma massificada e pelo contínuo ingresso de empresas, indústrias e serviços que se utilizam das vantagens de custos e facilidades de comunicação para se relacionarem com seu público.

Já *Web* definida como *"Web 3.0*" já circula pelos meios acadêmicos e indústria de tecnologia, prenunciando a "terceira geração" da Rede, a ser implementada de 05 a 10 anos. O futuro indica uma *Web* semântica, na qual, via conceitos de inteligência artificial, a Internet teria capacidade de produzir e armazenar conhecimento com base no histórico de ações do internauta, de forma semelhante ao processo que ocorre no cérebro, ao deduzir algo a partir de experiências anteriores.

6.4. Startups como Desenvolvimento Econômico

Traduzindo para o português STARTUP significa em linhas gerais INICIALIZAÇÃO. Portanto, partindo-se do pressuposto que todo processo de inicialização é um *start* contínuo, não se pode deixar de conceber o referido significado, à inspiradora boa ação das empresas que nascem com o objetivo de serem inovadoras.

Esse tipo de inspiração é extremamente positiva para o mercado. Caso contrário, só existiria uma empresa de automóveis, um fabricante de brinquedos, um rede de farmácias.

A verdade é que, mesmo no cenário mundial, apenas um pequeno percentual das empresas são baseadas em idéias realmente inovadoras. Sim, essas empresas são muito importantes, mas não menos importantes que todas as outras empresa que não são baseadas em idéias inéditas. São essas últimas que de fato geram a maior parte do fluxo de capital mundial e que sustentam a rede econômica. Acredita-se que toda essa questão está relacionada ao risco.

Falemos do Brasil em especial. É fato que não temos uma rede de *venture* capital bem desenvolvido, ou pelo menos não tão bem desenvolvido como nos EUA, berço de grandes empresas de tecnologia com modelos de negócio inovadores. O

empreendedor, aquele que tem a veia ou que a está descobrindo, ao analisar uma oportunidade de um novo negócio, precisa antes de tudo avaliar o risco do empreendimento. Qual a chance desse negócio dar certo ou falhar? Pela falta de uma bem desenvolvida rede de *venture* capital no Brasil, o empreendedor vê na adaptação de um modelo de negócio criado por terceiros uma bela oportunidade de reduzir drasticamente o risco do negócio. Nesse caso, as *VC*s atuam, olhando pelo aspecto pessoal do empreendedor, como um porto seguro que irá lhe permitir arriscar mais do que com um modelo de negócio totalmente inovador.

Imagine um sujeito que tem uma pequena economia guardada e que não pode se dar ao luxo de arriscar demais. Uma vez que exista um *VC* criando um porto seguro para o empreendedor (referindo-se exclusivamente ao aspecto pessoal do empreendedor) este pode se dar ao luxo de arriscar mais, seja com um modelo de negócio extremamente inovador e inédito ou não. Fábio Seixas quebrou três empresas antes de criar o Camiseteria. "Fechei minha terceira empresa em 2002 e fiquei longos 3 anos buscando uma oportunidade, justamente na época pós-bolha onde o mercado ainda estava ensaiando sua volta. Quando surgiu a oportunidade de nacionalizar o modelo de negócio do Threadless, vi ali uma forma de criar o que viria a ser o meu novo empreendimento e ainda assim fazê-lo com risco bastante moderado e sem necessidade de capital de risco. Eu poderia ter insistido em querer criar uma empresa baseada numa idéia inovadora e inédita (eu pensava nisso, obviamente) mas teria sido romântico demais da minha parte."

Como empreendedor, não se pode ter dúvidas de que sonhar em, algum dia, criar uma empresa de sucesso inovadora e inédita. Crer-se que esse é o nirvana para qualquer empreendedor. Ou seja, o desenvolvimento da rede de *VC*s no Brasil irá permitir que mais empresas com idéias inovadoras e inéditas surjam e conquistem o mundo. Mas não menos empresas baseadas em modelos de negócio criados por terceiros.

Portanto, Acreditar que inovação é mais abrangente do que somente criar algo novo a partir do nada é papel crucial e inevitável aqueles que vêem nas *startups* modelos de negócios promissores e valorosos. A prova disso é que se pode inovar criando uma solução diferente (e melhor) para problemas que já existem e já foram resolvidos de outras formas. Um exemplo bem real, é o iPod. Não foi nem de longe o primeiro Mp3 player portátil do mercado, no entanto consideram-no um produto inovador. Ele apenas proveu uma solução diferenciada (em termos usabilidade, design) para um problema que já existia e já havia sido solucionado. Além disso, ele inovou na resolução de um outro problema que faz parte da vida de quem usa um mp3 player portátil: conseguir as músicas para o aparelho (iTunes). Também considera-se inovação, adaptar uma solução adotada em outro lugar, para resolver problemas no contexto local do empreendedor.

6.5. Tecnologia da Informação e Sistemas de Informação

Desde a invenção do telégrafo elétrico em 1837, passando pelos meios de comunicação de massa, e até mais recentemente, o surgimento da grande rede de comunicação (Internet), o ser humano tem de conviver e lidar com um crescimento exponencial do volume de dados disponíveis.

O domínio da informação disponível é uma fonte de poder, uma vez que permite analisar fatores do passado, compreender o presente, e principalmente, antever o futuro.

Para entender este domínio, precisamos observar a diferença essencial entre um dado e uma informação. Um "dado" é uma característica qualquer obtida diretamente de um objeto, um ser ou um sistema. Uma "informação" é a consequência do processamento aplicado a esses dados, ou seja, é o resultado dos dados trabalhados e organizados. E "processar dados" consiste em aplicar aos dados um conjunto de operações lógicas e matemáticas, que produzam uma informação que pode ser usada para tomar decisões.

Em uma empresa, dados são números, nomes, gráficos e se tornam informação quando são interpretados por seus funcionários para uma ação estratégica da empresa. Assim, a informação se constitui em um dos principais patrimônios de uma empresa. Podemos afirmar que o fator determinante para o sucesso das organizações, qualquer que seja o seu porte ou ramo de atividade, depende, cada vez mais, de informações.

Elas são essenciais para as atividades de qualquer nível hierárquico empresarial. Até bem pouco tempo, os sistemas de informação nas organizações se baseavam na utilização de grandes arquivos, em geral manipulados por um arquivista, que era o responsável por organizar os dados, registrá-los, catalogá-los e recuperá-los, quando necessário.

O aumento do número de informações disponíveis nas empresas e a necessidade das áreas trabalharem de forma integrada tornaram o armazenamento, a recuperação e a integração das informações disponíveis uma tarefa difícil e não muito confiável. Com o surgimento dos microcomputadores veio à possibilidade de um acesso rápido e confiável destas informações.

Porém, a Tecnologia da Informação também cuida de áreas como planejamento, desenvolvimento de sistemas, suporte ao *software* e *hardware* e processos de produção e operação.

6.5.1. Tecnologia da Informação

O termo Tecnologia da Informação é comumente utilizado para designar o conjunto de recursos não humanos dedicados ao armazenamento, processamento e comunicação da informação, bem como o modo que esses recursos estão organizados em um sistema capaz de executar um conjunto de tarefas.

É salutar frisar que a TI não se restringe a equipamentos *(hardware),* programas *(software)* e comunicação de dados. Existem tecnologias relativas ao planejamento de informática, ao desenvolvimento de sistemas, ao suporte de *software*, aos processos de produção e operação, ao suporte de *hardware*, etc. A sigla TI, Tecnologia da Informação, abrange todas as atividades desenvolvidas na sociedade pelos recursos da informática.

Se tratar de uma espécie de difusão social da informação em larga escala de transmissão, a partir destes sistemas tecnológicos inteligentes.

Pequenas e grandes empresas dependem dela para alcançar maior produtividade e competitividade. Através de passos simples ensinados por empresas do ramo, muitas alcançam sucesso e alavancam maiores rendimentos.

6.5.2. Sistemas de Informação

Sistema, no âmbito da informática, trata-se de qualquer conjunto de componentes e processos por ele executado, que visa transformar determinadas entradas em saídas (saídas do sistema). Esses sistemas são subdivididos em outros componentes (subsistemas), e que, por sua vez, podem ser decompostos em novos sistemas menores.

Sistema de Informação (em inglês, Information System) é a expressão utilizada para descrever um sistema automatizado (que pode ser denominado como Sistema de Informação Computadorizado) que abrange pessoas, máquinas e/ou métodos organizados para coletar, processar, transmitir e disseminar dados que representam informação para o usuário.

Os atuais conceitos de Sistemas de Informação relacionam também os Sistemas de telecomunicações e/ou equipamentos relacionados. São sistemas ou subsistemas interconectados que utilizam equipamentos na aquisição, armazenamento, manipulação, gestão, movimento, no controle, na exposição, na troca, no intercâmbio, na transmissão, ou na recepção da voz e/ou dos dados, e inclui o *software* e *hardware* utilizados. Em relação a esta última definição, é comum nos meios acadêmicos a utilização do termo Tecnologias da Informação e Comunicação (ICT - *Information and Communication Technologies*).

6.5.2.1. Tipos de Sistemas de Informação

Muitos Sistemas de Informação (SI) começam como sistemas manuais e se transformam em computadorizados que estão configurados para coletar, manipular, armazenar e processar dados. SI baseados em computadores, são compostos por: *hardware, software*, banco de dados, telecomunicações, pessoas e procedimentos:

Hardware: consiste no equipamento, o computador usado para executar as atividades de entrada, processamento e saída.

Software: consiste nos programas e nas instruções dadas ao computador.

Banco de Dados (BD): é uma coleção organizada de fatos e informações.

Telecomunicações: permitem às empresas ligar os sistemas de computador em verdadeiras redes de trabalho.

Pessoas: consiste no elemento importante na maior parte dos sistemas de informação. Incluem todas as pessoas que gerenciam, executam, programam e mantêm o sistema de computador.

Procedimentos: incluem as estratégias políticas, métodos e regras usadas pelo homem para gerar os sistemas. Ex: descrevem quando cada programa deve ser executado, quem tem acesso a certos fatos em um BD, dentre outros.

No mundo coorporativo, a palavra de ordem é competitividade. Entre as empresas isso significa que seus processos desde a comercialização até a prestação de serviço devem ser rápidos e eficientes sob pena de perca de mercado e assim *now how* sob seus clientes e fornecedores. Assim sendo, a tecnologia vem se tornando um aliado para que as organizações tornem-se mais ágeis e melhorem a qualidade se seus produtos e serviços. Além disso a TI melhora a produtividade de seus funcionários e colaboradores e faz com que todos os processos, internos ou externos, sejam tratados de forma respeitosa e igualitária, pois recebem atenções cabíveis em cada caso.

6.6. Gestão da Tecnologia da Informação

Muitos autores pensam a TI como elemento estratégico e modificador na disputa da concorrência, podendo trazer vantagens competitivas através do incremento da satisfação dos clientes, aumento da receita, participação no mercado, bem como outras formas de atuação na competição. Para estes autores, a TI está afetando a competição através da mudança na estrutura dos setores de atuação, na criação de vantagens competitivas e na criação de novos produtos ou serviços.

A Glaxo Wellcome, da Grã-Bretanha, é uma das maiores empresas farmacêuticas do mundo. Ela descobriu, em 1996, que uma combinação de dois de seus medicamentos, o Epirir e o Retrovir, eram eficazes no tratamento de alguns casos de AIDS. Médicos do mundo todo começaram a prescrevê-los em massa. Esse excesso de demanda acabou com os estoques dos dois medicamentos nas farmácias.

A Glaxo precisava produzir e embarcar Epirir e Retrovir rapidamente. Mas a demanda incrementada, usada para planejar a produção, a programação de entregas e os níveis de estoque, era muito difícil de prever.

Para resolver o problema, a Glaxo desenvolveu um sistema especial de informações em rede, abrangendo toda a empresa, baseado em Tecnologia ROLAP (processamento analítico *online* relacional). Basicamente, esse sistema processa os dados assim que as transações ocorrem. O sistema trabalha com uma enorme quantidade de dados externos e internos. Usando estes dados e modelos de Sistemas de Apoio a Decisão, os analistas de mercado da Glaxo tiveram condições de localizar e quantificar as origens de demanda, gerando relatórios concisos e projeções em minutos.

A demanda projetada alimentou modelos de SAD para estabelecer planos de produção adequados, programação de entregas e níveis de estoque ao longo de sua cadeia de suprimentos.

Em conseqüência, a Glaxo tornou o seu processo de distribuição mais interligado, eliminando, para atacadistas e varejistas no mundo inteiro, o risco da repentina falta de estoques de medicamentos de grande demanda como os citados acima. Outra vantagem adicional foi a redução dos custos operacionais. Além disso, as informações são armazenadas de forma centralizada e assim podem ser difundidas e repassadas aos funcionários da Glaxo de forma mais rápida e eficiente. A solução de TI, unida a uma boa prática de gestão, propôs à empresa uma maximização de resultados de negócios e, ao mesmo tempo, salvar vidas.

Sob esse aspecto, a TI tem um importante papel para alavancar as estratégias competitivas. Investindo em sistemas de informação avançados e computadorizados para melhorar sua própria eficiência, as empresas conseguem desenvolver novos produtos e serviços e/ou soluções que não seriam possíveis sem uma forte capacidade de TI. O fato é que a Gestão da Tecnologia da Informação associada a boas políticas corporativas possibilitam as empresas alavancarem seus investimentos anteriores em larga escala e ainda promoveram a “felicidade” de integrantes ou parceiros de negócios.

Portanto, a solução informatizada de um problema empresarial pode exigir a integração de dois os mais sistemas para dirimir ou resolver o(s) problema(s) surgido(s) em qualquer fase de processos ou procedimentos da organização. Assim, para uma

melhor Gestão, baseada nos processos de Tecnologia da Informação, é importante criar de forma real e precisa, sistemas de apoio a decisão que combinados com sistemas especialistas, possam conceder o "estado da arte" ou o caminho das pedras de uma determinada solução para problemas que possam surgir antes ou durante os processos da organização.

6.6.1. Planejamento Estratégico da TI

Para Weill & Broadbent (2000), a infraestrutura de TI é a base da capacidade da tecnologia de informação, tida como serviços confiáveis compartilhados pela empresa e coordenados centralmente, geralmente pelo grupo de sistemas de informação. A atenção dispendida na busca pela harmonia da Tecnologia de Informação com a empresa pode afetar significativamente a competitividade e eficiência do negócio.

Nesta discussão, o ponto principal é saber como a TI pode ajudar a alcançar vantagem competitiva e estratégica para a empresa. Logo, considerando o que ora foi apresentado pode-se afirmar que é fundamental o planejamento da TI antes de sua utilização para evitar que a organização corra riscos desnecessários, pois o uso crescente da TI, ao mesmo tempo em que potencializa a capacidade das organizações em obter, manter ou combater vantagens competitivas, também eleva os riscos de gestão inerentes a qualquer tipo de decisão e ação.

Rezende (2003) afirma que "o planejamento estratégico é fundamental para a sobrevivência das organizações que estão preocupadas com sua inteligência empresarial ou organizacional". A tecnologia de informação é essencial para as empresas.

Audy e Brodebeck (2003) afirmam: "atualmente não faz mais sentido discutir se a TI é estratégica ou não, sendo a questão deslocada para como utilizar a TI de forma alinhada à estratégia organizacional".

O planejamento estratégico da TI é a forma como a organização irá tratar e gerir a tecnologia visando ao futuro, adequando-se da melhor maneira possível para obter incremento na competitividade, ampliação dos seus negócios e maior performance de seus investimentos tecnológicos. É fundamental que as organizações utilizem-se de um planejamento estratégico de tecnologia de informação alinhado com seu planejamento estratégico empresarial. Para Lederer e Sethi (apud AUDY e BRODBECK, 2003), o planejamento estratégico de sistemas de informação é o processo de identificação de um portfólio computadorizado de aplicações para dar suporte ao plano de negócios da organização e auxiliar na concretização dos objetivos organizacionais.

Segundo Rezende (2003), o planejamento estratégico da TI é: "um conjunto de ferramentas e técnicas que possibilitam a definição de estratégias de ação ao longo de um período de aproximadamente um a três anos". Durante seu processo é necessário que seus componentes sejam avaliados e redefinidos caso estejam fora do esperado.

O Planejamento Estratégico da Tecnologia da Informação é um processo dinâmico e interativo para estruturar estratégica, tática e operacionalmente as informações e conhecimentos organizacionais, a TI (e seus recursos: *hardware*, *software*, sistemas de telecomunicações, gestão de dados e informação), os sistemas de informação e do conhecimento, o perfil das pessoas envolvidas e a infra-estrutura necessária para o entendimento de todas as decisões, ações e respectivos processos da

organização (REZENDE, 2002). Daí a demonstração da complexidade e a importância de um bom planejamento de um planejamento estratégico da TI.

Podemos dizer que o planejamento estratégicos da TI auxilia a organização a gerenciar e aplicar da melhor forma os recursos computacionais disponíveis, visando a melhorias na forma de tratar, disseminar e armazenar as informações necessárias ao negócio da instituição, e dar suporte ao planejamento estratégico empresarial em busca de seus objetivos.

Para obtenção do sucesso é fundamental que o Plano Estratégico Empresarial e o planejamento estratégico da TI estejam interligados entre si, e que ambos sejam reavaliados com o decorrer do tempo e não esquecidos dentro das gavetas das corporações. As metodologias para o Planejamento Estratégico da TI são importantes, pois auxiliam nas decisões e definições estratégicas, além de colaborarem para a integração entre Tecnologia da Informação e Gestão Empresarial.

Para Rezende (2003), "a relevância do planejamento dos sistemas de informação, dos sistemas de conhecimento e da informática ou tecnologia da informação é reiterada pela escolha de uma metodologia para sua implementação adequada, dinâmica e inteligente".

As metodologias podem contribuir no grande desafio da TI: gerenciar a infraestrutura de tecnologias; gerenciar o orçamento financeiro; buscar novas arquiteturas necessárias a transformação organizacional nos processos de negócios e de tomadas de decisões organizacionais.

A tecnologia da informação, por meio de metodologias de planejamento de informação, conhecimentos e informática, pode apoiar as organizações nas atividades mercadológicas competitivas, planejando, gerindo e flexibilizando sua infraestrutura, para permitir gerar informações necessárias de forma oportuna para uso estratégico. As primeiras metodologias foram criadas no fim da década de 60. Destacam-se dentre elas:

- BSP (*Business Systems Planning*) da IBM que explora o planejamento de sistemas de informação e as relações deles com os negócios, de maneira que os sistemas de informação possam dar suporte às necessidades organizacionais.
- SSP (*Strategic Systems Planning*) define um modelo funcional de negócios pela análise funcional das áreas da organização, que geram a arquitetura dos dados combinada com a necessidade de informações da organização.
- IE (*Information Engineering*) estabelece técnicas para a construção de modelos organizacionais, modelos de dados e modelos de processos.
- CSF (*Critical Success Factors*) procura identificar as necessidades de sistemas de informação da alta administração da organização, em que principais fatores críticos de sucesso são descritos para a gestão, numa abordagem *top-down* (de cima para baixo, ou seja do alto escalão para o baixo escalão empresarial). Ela deve ser utilizada em conjunto com uma metodologia de planejamento estratégico de TI pois ela não é uma metodologia propriamente dita.

Abordagem por estágios de crescimento da organização apresenta uma metodologia com um critério de administração da evolução dos sistemas de informação na organização.

Para Rezende e Abreu (2003), "as metodologias são ferramentas que possibilitam a definição de estratégias de ação ao longo de um período de aproximadamente um a três anos". Alguns de seus objetivos são:

- Estabelecer ferramentas de controle de qualidade, produtividade, prazos e custos;
- Elaborar plano de desenvolvimento (ou aquisição), implantação, implementação de Sistemas de Informação;
- Padronizar e simplificar a Tecnologia da Informação.

O modelo de planejamento estratégico em TI proposto por King em 1978, que serve de base para os atuais, consta de três fases ou etapas:

- Análise ou diagnóstico, a análise da situação atual;
- Processo ou planejamento, definição das entradas e saídas de informação, recursos e outros itens;
- Procedimentos ou execução, implantação das definições e controle do cumprimento das metas por medidas de performance.

Nessas fases, são elaborados roteiros de planejamento de atividades, de desempenho, de definição de papéis dos participantes, de reuniões de trabalho e de revisões de ações.

Complementando os estudos, Prenkumar e King (1991 apud AUDY e BRODBECK, 2003) consideraram o planejamento estratégico de TI como um processo e como fator determinante. Para seu sucesso, identificaram a existência de planos integrados pelos seguintes componentes:

- Missão, papel que a informação e a TI exercem no contexto organizacional;
- Avaliação dos objetivos implementados e extração dos objetivos futuros e definição dos objetivos futuros;
- Regras gerais para orientar o desenvolvimento dos SI;
- Direção geral para o desenvolvimento dos sistemas e para a composição da infraestrutura de suporte tecnológico;
- Programas de desenvolvimento, atividades e regras que servem de guia para a implementação dos SIs e da tecnologia disponibilizada;

Os modelos para elaboração de planejamento estratégico de TI não devem ser estáticos, mas sim dinâmicos, flexíveis, adaptáveis e inteligentes, devem permitir que todos os envolvidos tenham condições de avaliar os produtos gerados nas subfases, registrando sua parcela de participação, para que o projeto seja de qualidade e contribua para o alinhamento do planejamento estratégico da TI ao planejamento estratégico empresarial (REZENDE, 2002).

King e Teo (1997 apud REZENDE, 2002) defendem que um dos elementos chaves do planejamento estratégico de TI é a integração e o alinhamento com o planejamento estratégico empresarial e as estratégias da organização.

6.7. Avaliação e Melhoria da Tecnologia da Informação

Ao considerar que um dos subsistemas de uma empresa pode ser a tecnologia, foi desenvolvido um instrumento de avaliação multiescala baseado na teoria da difusão de inovação de Rogers (1995). A teoria propõe cinco atributos de uma inovação que influencia sua adoção, que são: vantagem relativa, compatibilidade, complexidade, habilidade para experimentação e habilidade para a observação. Na discussão de medição da segurança em TI, foi criado o Critério de Avaliação da Segurança de Tecnologia da Informação do Reino Unido (Information Technology Security Evaluation Criteria – ITSEC UK). De acordo com este critério, a segurança é o aspecto essencial da TI e significa: confidencialidade: prevenção da divulgação proibida de informações; integridade: prevenção da modificação proibida da informação; disponibilidade: prevenção da retenção proibida da informação ou recursos.

Já na avaliação de TI, existe uma metodologia denominada *Control Objectives for Information and Related Tecnology*, ou somente COBIT. Ela é referência para a gestão e controle de ambientes tecnológicos seguros e é utilizada para avaliar o alinhamento estratégico de TI com as áreas de negócios da empresa, aprimorando os processos desde a alta direção até o departamento de TI. A COBIT agrupa os processos de TI em quatro domínios: planejamento e organização; aquisição e implementação; entrega e suporte; e monitoração.

Diversas formas de avaliar a Tecnologia da Informação, em suas diferentes manifestações e denominações (processamento de dados, informática, sistemas de informação, etc.), foram apresentadas, tanto na literatura acadêmica como na de negócios. Segundo Moss (1993), há duas abordagens para a avaliação da TI:

Ênfase na eficiência:

Medida do desempenho com que as entradas (*inputs)* são transformadas nas saídas (*outputs*) do sistema. Foco na atividade de desenvolvimento de sistemas, interno e predominantemente técnico. Exemplo: número de linhas codificadas, custo previsto contra custo real, número de serviços prestados aos usuários, porcentagem de erros por fase do desenvolvimento, tempo de conclusão dos projetos, tempo de espera das solicitações na carteira.

Ênfase na eficácia:

Medida de como as saídas do sistema atendem aos objetivos, metas e requisitos do sistema. Foco no impacto do sistema na organização em que se insere. Exemplo: grau de satisfação dos usuários, impacto nos negócios (antes e depois da implantação de um sistema de informações), compatibilidade dos projetos de sistemas de informações com a estratégia da empresa.

Outra classificação que contribui para uma maior compreensão do processo de avaliação da TI foi elaborada por Hirscheim (1998), que, além de apresentar uma tipologia, faz uma retrospectiva de diversas ideias, técnicas e modelos para avaliação da TI.

Segundo estes autores, há consenso que, inicialmente, predominaram as avaliações sob a ótica da eficiência. Contudo, com o aperfeiçoamento da TI e com a compreensão de que seu alcance seria muito maior do que a mera automação de processos existentes, passamos a procurar a eficácia como parâmetro para a avaliação da TI. Desta forma, podemos notar um panorama de crescente preocupação com a avaliação da TI, com a consciência de que se trata de um problema complexo, que não pode ser resolvido apenas com tradicionais abordagens de viabilidade econômica de projetos. Para Hirscheim (1998), há três "zonas" nas quais se podem incluir as diferentes formas de avaliar a TI: da eficiência, da eficácia e da compreensão.

Zona da Eficiência (*Efficiency Zone*)

Caracterizada por suposições razoavelmente objetivas em relação à avaliação, enfatizando desempenho e qualidade em relação a padrões e/ou especificações de baixo nível. Há relativamente pouca controvérsia.

Zona da Eficácia *(Effectiveness Zone)*

Alinhada com a ideia de "fazer as coisas certas", envolvendo conceitos como análise de aplicações localizadas, análise custo-benefício, comparação com objetivos e estudos sobre satisfação dos usuários. Há um maior nível de controvérsia do que na zona anterior.

Zona da Compreensão *(Understanding Zone)*

Assume a avaliação como algo problemático e controverso. Busca uma maior compreensão da avaliação dentro de um particular contexto organizacional, desde o trabalho do indivíduo até a estratégia da empresa. Envolve também conceitos sociais e psicológicos.

6.8. Como Medir os Efeitos da TI

Mesmo nos primeiros lugares onde foi aplicada e utilizada a TI, nem sempre os ganhos de produtividade foram imediatamente quantificáveis, embora fossem identificáveis, porque se revelaram difíceis de medir. Os resultados dos estudos sobre impacto da TI não são conclusivos, pois sua análise está apoiada na leitura de trabalhos publicados nas décadas de 80 e 90, que investigam apenas se os efeitos foram positivos ou negativos, e suas hipóteses estão fundamentadas em questões como produtividade e retorno econômico (BARUA, 1995). Desde que a filosofia de administração científica de Frederick Taylor formalizou a análise do processo de produção, tornando atividades e processos na organização mensuráveis e quantificáveis, uma alternativa de investimento em tecnologia é aconselhável se o valor presente nos fluxos de caixa futuros descontados ao custo de capital adequado for positivo.

As dificuldades de quantificar esses ganhos não significa que os investimentos devam ser reduzidos, mas que devem ser desenvolvidos instrumentos mais sofisticados para que se possam medir tais benefícios. Ao analisar os efeitos da TI sobre as organizações americanas, Cornella (1994) identificou aspectos que justificam o desenvolvimento de instrumentos mais sofisticados, como:

- Os benefícios promovidos pela TI não são visíveis imediatamente;

- O impacto da TI é escasso se sua aplicação não vem acompanhada de trocas no gerenciamento da organização;
- A implantação da TI não tem correspondido às necessidades fundamentais da empresa;
- O impacto da TI não se evidencia em forma de retornos econômicos, mas em melhorias sociais da qualidade de vida.

Percebemos que os métodos tradicionais de avaliação de orçamento de capital não são apropriados para avaliar os investimentos de TI, pois seus benefícios, diferentemente dos investimentos da tecnologia industrial tradicional, não assumem a forma de fluxos de caixa incrementais.

Este é o desafio: desenvolver modelos de análise que consigam, efetivamente, identificar integralmente os benefícios derivados da TI, substancialmente distintos dos benefícios derivados de tecnologias tradicionais.

6.9. Inovação

Nova pesquisa da Unisys, realizada com 1.200 organizações em todo o mundo, revela uma considerável defasagem entre as metas de negócios e tecnologia da informação (TI) dos executivos e suas estimativas quanto à capacidade das organizações de alcançar esses objetivos.

Essa diferença indica que os executivos não estão aproveitando ao máximo as iniciativas e investimentos em TI e precisam de novas abordagens para modernizar os processos, os aplicativos estratégicos e as infraestruturas tecnológicas para enfrentar os principais desafios de negócios. Solicitados a expor seus objetivos de negócios mais importantes, os entrevistados do levantamento priorizaram as seguintes metas: conquistar novos clientes; estabelecer relacionamentos mais estreitos com eles; desenvolver novos produtos e serviços; aumentar as vendas e receitas; e abrir novos mercados.

Os executivos identificaram 10 características – quase todas relacionadas a investimentos em informação e TI – que consideram fundamentais para chegar a esses objetivos de negócios. Destacam-se entre elas a capacidade de apoiar a inovação, práticas de gestão de TI, tomada estratégica de decisões, abordagem de investimentos em TI, comunicações (fluxo de informação dentro da organização e entre organizações) e modelos de compras e de segurança de TI. A pesquisa pediu ainda aos executivos que avaliassem o atual estágio da sua organização em relação à capacidade de execução em cada área e onde esperam que ela esteja daqui a três anos. Seguem os resultados:

- 62% dos entrevistados pretendem, dentro de três anos, incentivar a inovação ou alcançar a liderança de mercado no apoio à inovação. Ainda assim, 70% dizem que não contemplam a inovação ou que sua capacidade é moderada ou está apenas em evolução;
- Ao descrever as práticas de gestão de TI, 52% dos entrevistados esperam tratar a TI como um investimento ou diferencial em três anos, porém 72% a consideram uma função de apoio, um meio de aumentar a produtividade ou um gasto de capital;

- 67% dos entrevistados desejam ter comunicações integradas e colaborativas com as principais partes envolvidas, mas somente 32% afirmam que já têm essa capacidade;
- Embora apenas 51% dos entrevistados possuam estratégia e modelo formais e amadurecidos de compras de TI, 75% ambicionam tê-los dentro de três anos;
- 60% dos entrevistados avaliam seu modelo de segurança como inexistente, limitado ou moderado; contudo, 75% almejam que ele seja de última geração em três anos.

Na sociedade contemporânea caracterizada pela globalização, a competência é questão de sobrevivência. Somente empresas que inovam sua postura diante dos clientes, lançando novos produtos, investindo em novos processos produtivos, valorizando e capacitando seus recursos humanos sobrevivem no mundo atual. E a tecnologia – entendida como o conhecimento da empresa – exerce papel fundamental.

Assim, o gerenciamento da tecnologia abrange a escolha de tecnologias apropriadas à realidade organizacional da empresa (pessoas, estrutura administrativa e estrutura tecnológica existente) aliado a um ambiente inovativo e empreendedor que se caracterizam em estratégia competitiva.

Esta realidade não se apresenta diferente nas MPIs, no entanto, estas empresas que permanecem no mercado após o período de nascimento e procuram a consolidação, são MPIs inovadoras e que agregam valor aos seus produtos, garantindo mercado. Na maioria dos casos são empresas flexíveis, possuem estrutura organizacional enxuta, valorizam a criatividade e suas estratégias são informais, porém focalizadas no seu negócio. Precisam vencer muitas dificuldades para vencer, principalmente em termos de desenvolvimento tecnológico, gerenciamento e recursos financeiros. Uma característica marcante é o fato destes empreendimentos refletirem a postura profissional e pessoal do proprietário e muitas vezes constitui-se num fator de sucesso.

Criatividade é um processo que conduz à produção de idéias originais para a pessoa que está criando, ou para um conjunto de pessoas, e até para o mundo inteiro. É um sistema aberto no qual qualquer pessoa pode criar. Não há limites para isto. E podem dar idéias para qualquer campo de atividade humana. Também não há restrição. Por isto, a **criatividade** tem a ver com o **pensar**. Sua ferramenta básica é a **imaginação**.

Já a inovação é um sistema fechado. A partir da nova idéia recebida, caberá uma pessoa, ou um conjunto delas, adicionar o conhecimento (ou tecnologia, como preferir) para produzir alguma coisa concreta (um processo, um equipamento, um material, um serviço, etc). Assim, a **inovação** tem a ver com o **fazer**. Sua ferramenta básica é o **conhecimento**.

Criatividade e Inovação são diferentes. Criatividade é uma característica humana, Inovação é uma competência de agir criativamente, visualizando e aproveitando oportunidades, agregando valor e incrementando possibilidades.

Esses resultados nos levam a acreditar que a maioria dos gestores tem conhecimento sobre a necessidade e a importância da TI. Mas nada é feito a curto prazo, o que por um lado pode facilitar o planejamento da introdução de TI; a demora, por outro lado, pode fazer com que as empresas percam espaço para a concorrência.

Referências

Sua Vida Digital. *Estado de São Paulo*. (2010). <http://www.link.estadao.com.br/index.cfm?id_conteudo=8792>.

Dolabela, Fernando. (2006) O segredo de Luísa. 30. ed. ver. e atual. São Paulo: Editora de Cultura.

Castells, Manuel. *A era da informação: economia, sociedade e cultura*. V.1. 9. ed. São Paulo: Paz e Terra, 2006.

Goossen, Richard J. E-Empreendedor: vencendo no mercado virtual corporativo. [Tradução: Sabine Alexandra Holler]. Rio de Janeiro: Editora Elsevier, 2008.

Ferreira, Aurélio B. H. Novo Aurélio século XXI: o dicionário da língua portuguesa. 3 ed. Rio de Janeiro: Nova Fronteira, 1999.

Dornelas, J.C.A. Empreendedorismo Corporativo. Rio de Janeiro: Campus, 2003.

Pinho, J. B. *Publicidade e vendas na Internet. Técnicas e estratégias*. São Paulo: Summus Editorial, 2000.

Vassos, Tom. (1997) *Marketing Estratégico na Internet*. São Paulo: Makron Books. Ibope/Netrankig. (2010) http://www.ibope.com.br. Julho.

Rezendo, D. A.; Abreu, A. F. Tecnologia da aplicada a sistemas de informação empresariais. 3. ed. São Paulo: Atlas, 2003.

Nmc Virtual World. New Media Consortium. (2010) <http://virtualworlds.nmc.org/2006/12/21/jenkins>. Julho.

Nmc Virtual World. New Media Consortium. (2010) <http://virtualworlds.nmc.org/2006/12/21/jenkins>. Julho.

Audy, J. L. N.; Brodbeck, A. F., Sistemas de informações, planejamento e alinhamento estratégico nas organizações. São Paulo: STS, 1998.

Rogers, E. Diffusion of innovations. New York. The Free Press, 1995.

Moss, E.; Walrad, C. Measurement: the key to application development quality. IBM Systems Journal. v.32, n.3, p.445-460, 1993.

BARUA, A.; Kriebel, C. H. & Murkhophyay. *Information technologies and business value: an analytic and empirical investigation.* Information Systems Research, Vol. 6, n. 1, p.3-23, march 1995.

Cornella, A. Los recursos de información: ventaja competitiva de las empresas. McGraw- Hill/Interamericana de España, 1994.

Santos, P. C. C. Os Novos Formatos de Empreendedorismo *On-Line* – Estudo do Empreendedorismo no Second Life. (2009) Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) Faculdade de Tecnologia do Piauí - FATEPI, Teresina-PI.

Dolabela, Fernando. Oficina do empreendedor. 6 ed. São Paulo: Cultura, 1999.

Capítulo 7

Cloud Computing: Montagem uma Infraestrutura Privada

Julio Cesar Damasceno,Fernando Antonio Aires Lins,Márcio Almeida de Macêdo

Abstract

Cloud Computing has proven to be a promising paradigm, since there is the possibility to pay only for the used resources in a flexible way where it can scale more or less. The advantage to choose for a private cloud, where resources are available only to a limited group of users, is motivated by the fact that it's construction can be made from current hardware with a reasonable cost. This short course aims to show the technologies that support the infrastructure of private cloud and show, through a practical example, the setup using low cost hardware with open source tools.

Resumo

A computação em Nuvem ou Cloud Computing tem-se mostrado um paradigma bastante promissor, pois existe a possibilidade de pagar apenas pelos recursos utilizados de maneira flexível onde os mesmo podem escalar para mais ou para menos. A vantagem para se optar por uma cloud privada, onde os recursos são disponibilizados apenas para um grupo restrito de usuários, é motivada pelo fato que a sua construção pode ser feita a partir de hardware atual com um custo razoável. Este minicurso tem por objetivo mostrar as tecnologias que suportam a infraestrutura de cloud privada bem como mostrar, através de um exemplo prático, a sua criação utilizando hardware de baixo custo com ferramentas opensource.

7.1. Introdução

Computação em nuvem é um modo de usar recursos computacionais apenas quando necessário pelo tempo que for necessário. Como a Internet é a base para a computação em nuvem, pois todos os serviços e recursos devem estar disponíveis a qualquer momento e em qualquer, porém a decissão de colocar dados sensíveis na Internet pode criar um

novo problema relacionado a privacidade e a segurança dos dados. Uma possibilidade para amenizar estes problemas é montar um infraestutura privada, onde apenas um grupo restrito de usuários terá acesso aos recursos desta nuvem.

Nas próximas seções serão apresentados os conceitos básicos para o entendimento do tema e apresentado de maneira detalhada como montar uma infraestrutura privada utilizando ferramentas *opensource*.

7.2. Conceitos Básicos

Nessa seção serão introduzidos os conceitos e tecnologias básicos para o entendimento do restante desse trabalho. Inicialmente será dada uma visão geral sobre virtualização, logo em seguida serão abordadas as divisões e propriedades de computação em nuvem, ou *cloud computing*, e uma breve intrdodução a ferramenta Eucalyptus que foi utilizada para montar a infraestrutura privada.

7.2.1. Virtualização

Virtualização é uma técnica que permite a execução simultânea dois ou mais ambientes (sistemas operacionais) distintos e isolados em uma mesma máquina. Esse conceito de virtualização relembra os antigos mainframes, que deviam ser compartilhados por vários usuários em ambientes de aplicação completamente diferentes. Essa realidade da década de 1970 foi em grande parte superada nos anos de 1980 e 1990, com o surgimento dos computadores pessoais. No entanto, atualmente há uma onda crescente de interesse sobre as técnicas de virtualização.

Nos dias de hoje, o interesse na virtualização não se atém somente ao fato de permitir o uso de um mesmo sistema por vários usuários concomitantemente, mas os principais interesses são as vantagens oferecidas por esse tipo de ambiente, são elas: segurança, confiabilidade e disponibilidade, custo, adaptabilidade, balanceamento de carga e suporte a aplicações legadas.

7.2.1.1. Definições e Conceitos

Os primeiros conceitos que devemos ter em relação à virtualização são os conceitos de instruções privilegiadas e não privilegiadas. Essas instruções fazem parte do conjunto de instruções de uma arquitetura. As instruções não-privilegiadas são aquelas que não modificam a alocação ou o estado de recursos compartilhados por vários processos simultâneos, tais como processadores, memória principal e registradores especiais. Em contrapartida, temos as instruções privilegiadas, que podem alterar o estado e a alocação desses recursos.

Um computador pode operar em dois modos distintos, o modo de usuário ou o de supervisor. O modo de usuário, também chamado de espaço de aplicação, é o modo no qual as aplicações normalmente são executadas. Neste modo, não é possível executar as instruções privilegiadas, que são restritas ao modo de supervisor.

O modo de supervisor tem o controle total sobre a CPU, podendo executar todas as instruções do conjunto de instruções do processador, tanto as não-privilegiadas como

as privilegiadas. O sistema operacional é executado neste modo. Antes de o sistema operacional passar o controle da CPU para uma aplicação do usuário, o bit de controle de modo é configurado para o modo de usuário.

Vale lembrar que na arquitetura x86, existem quatro níveis de privilégio, que são chamados de *rings*. Os *rings* são numerados de 0 a 3, nos quais o nível 0 é o que tem maior privilégio na execução de instruções, por isso, os sistemas operacionais são executados com esse nível de privilégio.

Já em um ambiente virtualizado, temos que definir mais dois conceitos, os de sistema operacional hospedeiro e o de sistema operacional visitante. O sistema operacional hospedeiro (*Host Operating System*), refere-se ao SO nativo da máquina na qual ocorrerá a virtualização. Já o sistema operacional visitante (*Guest Operating System*), refere-se ao sistema operacional que é executado sobre a máquina virtual. Uma máquina na qual é feita a virtualização pode-se contar com apenas um SO hospedeiro sendo executado por vez. No entanto, podem ser executados diversos SOs visitantes simultaneamente.

O próximo conceito a ser discutido é de vital importância para o entendimento da virtualização. O conceito em questão é o do *Virtual Machine Monitor* (VMM), ou seja, Monitor de Máquina Virtual, também conhecido por *Hypervisor*.

O *hypervisor* provê uma camda de abstração entre o hardware e o sistema operacional que esta executando conforme é mostrado na Figura 8.1.

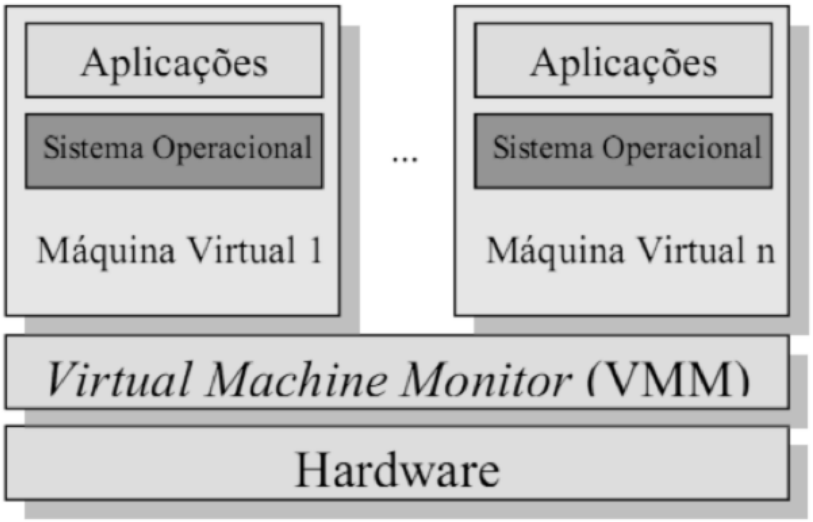


Figura 7.1: Relacionamento das Máquinas Virtuais e do VMM

O *Virtual Machine Monitor* é um componente de software que hospeda as máquinas virtuais [10]. O VMM é responsável pela virtualização e controle dos recursos compartilhados entre as máquinas virtuais, tais como, processadores, dispositivos de entrada e saída, memória, armazenamento, etc. Também é função do VMM escalonar qual máquina virtual vai executar a cada momento, semelhante ao escalonador de processos do Sistema Operacional [7].

O VMM é executado no modo de supervisor, no entanto as máquinas virtuais são

executadas em modo de usuário. Como as máquinas virtuais são executadas em modo de usuário, quando estas tentam executar uma instrução privilegiada, é gerada uma interrupção e o VMM se encarrega de emular a execução desta instrução.

7.2.1.2. Vantagens e Desvantagens

Existem diversas vantagens na virtualização, a seguir serão citadas as principais [7]:

- ***Segurança***: Usando máquinas virtuais, pode ser definido qual é o melhor ambiente para executar cada serviço, com diferentes requerimentos de segurança, ferramentas diferentes e o sistema operacional mais adequado para cada serviço. Além disso, cada máquina virtual é isolada das demais. Usando uma máquina virtual para cada serviço, a vulnerabilidade de um serviço não prejudica os demais;

- ***Confiança e disponibilidade***: A falha de um software não prejudica os demais serviços;

- ***Custo***: A redução de custos é possível de ser alcançada com a consolidação de pequenos servidores em outros mais poderosos. Essa redução pode variar de 29 a 64 [7];

- ***Adaptação às diferentes cargas de trabalho***: Variações na carga de trabalho podem ser tratadas facilmente. Ferramentas autônomas podem realocar recursos de uma máquina virtual para a outra;

- ***Balanceamento de carga***: Toda a máquina virtual está encapsulada no VMM. Sendo assim é fácil trocar a máquina virtual de plataforma, a fim de aumentar o seu desempenho; .

- ***Suporte a aplicações legadas***: Quando uma empresa decide migrar para um novo Sistema Operacional, é possível manter o sistema operacional antigo sendo executado em uma máquina virtual, o que reduz os custos com a migração. Vale ainda lembrar que a virtualização pode ser útil para aplicações que são executadas em hardware legado, que está sujeito a falhas e tem altos custos de manutenção. Com a virtualização desse hardware, é possível executar essas aplicações em hardwares mais novos, com custo de manutenção mais baixo e maior confiabilidade.

Por outro lado, existem as desvantagens da virtualização, sendo as principais:

- ***Segurança***: Segundo Neil MacDonald, especialista de segurança da Gartner, hoje em dia, as máquinas virtuais são menos seguras que as máquinas físicas justamente por causa do VMM [1]. Este ponto é interessante, pois se o sistema operacional hospedeiro tiver alguma vulnerabilidade, todas as máquinas virtuais que estão hospedadas nessa máquina física estão vulneráveis, já que o VMM é uma camada de software, portanto, como qualquer software, está sujeito a vulnerabilidades;

- ***Gerenciamento***: Os ambientes virtuais necessitam ser instanciados, monitorados, configurados e salvos [1]. Existem produtos que fornecem essas soluções, mas esse é o campo no qual estão os maiores investimentos na área de virtualização, justamente por se tratar de um dos maiores contra-tempos na implementação da virtualização. Vale lembrar que o VMWare é a plataforma mais flexível e fácil de usar, mas ainda apresenta falhas que comprometem a segurança, assim como as demais plataformas [1];

- ***Desempenho***: Atualmente, não existem métodos consolidados para medir o desempenho de ambientes virtualizados. No entanto, a introdução de uma camada extra de software entre o sistema operacional e o hardware, o VMM ou *hypervisor*, gera um custo de processamento superior ao que se teria sem a virtualização. Outro ponto importante de ressaltar é que não se sabe exatamente quantas máquinas virtuais podem ser executadas por processador, sem que haja o prejuízo da qualidade de serviço.

7.2.1.3. Virtualização total e para-virtualização

Existem duas formas de implementação dos monitores de máquina virtual: a virtualização total e a para-virtualização.

A virtualização total tem por objetivo fornecer ao sistema operacional visitante uma réplica do hardware subjacente conforme observado na Figura 8.2. Dessa forma, o sistema operacional visitante é executado sem modificações sobre o monitor de máquina virtual (VMM), o que traz alguns inconvenientes. O primeiro é que o número de dispositivos a serem suportados pelo VMM é extremamente elevado.

Para resolver esse contratempo, as implementações da virtualização total usam dispositivos genéricos, que funcionam bem para a maioria dos dispositivos disponíveis, mas não garantem o uso da totalidade de sua capacidade. Outro inconveniente da virtualização total é o fato de o sistema operacional visitante não ter conhecimento de que está sendo executado sobre o VMM, então as instruções executadas pelo sistema operacional visitante devem ser testadas pelo VMM para que depois sejam executadas diretamente no hardware, ou executadas pelo VMM e simulada a execução para o sistema visitante. Por fim, o último inconveniente da virtualização total é o fato de ter que contornar alguns problemas gerados pela implementação dos sistemas operacionais, já que esses foram implementados para serem executados como instância única nas máquinas física, não disputando recursos com outros sistemas operacionais. Um exemplo desse último inconveniente é uso de paginação na memória virtual, pois há a disputa de recursos entre diversas instâncias de sistemas operacionais, o que acarreta em uma queda do desempenho [1].

A para-virtualização é uma alternativa à virtualização total. Nesse modelo de virtualização, o sistema operacional é modificado para chamar o VMM sempre que executar uma instrução que possa alterar o estado do sistema, uma instrução sensível. Isso acaba com a necessidade de o VMM testar instrução por instrução, o que representa um ganho significativo de desempenho. Outro ponto positivo da para-virtualização é que os dispositivos de hardware são acessados por drivers da própria máquina virtual, não necessitando

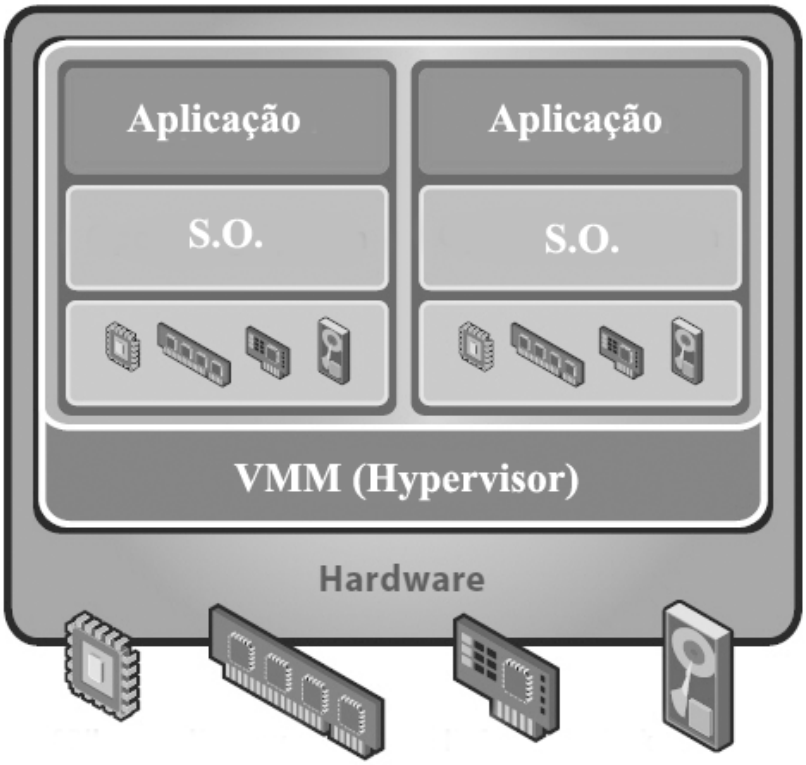


Figura 7.2: Arquitetura da Virtualização Total

mais do uso de drivers genéricos que inibiam o uso da capacidade total do dispositivo. A arquitetura da para-virtualização é mostrada na Figura 8.3.

Embora a para-virtualização apresentasse um ganho de desempenho significativo frente à virtualização total, essa disparidade tem sido superada devido à presença de instruções de virtualização nos processadores Intel e AMD, que favorecem a virtualização total. Embora tenham sido desenvolvidas para o mesmo propósito, foram criadas de maneira independente. Por esse motivo, há alguns problemas na portabilidade de máquinas virtuais de uma arquitetura Intel para a arquitetura AMD e vice-versa.

Portanto, tendo em vista as técnicas de virtualização, a decisão de qual melhor a técnica de virtualização para um dado ambiente está intimamente ligada a qual o processador da máquina física que vai hospedar as máquinas virtuais, bem como se o processador possui ou não uma extensão no seu conjunto de instruções que suporte a virtualização.

7.2.2. Cloud Computing

Nesta seção, serão apresentadas as propriedades e características essenciais de uma *cloud* e classificações usuais nessa temática.

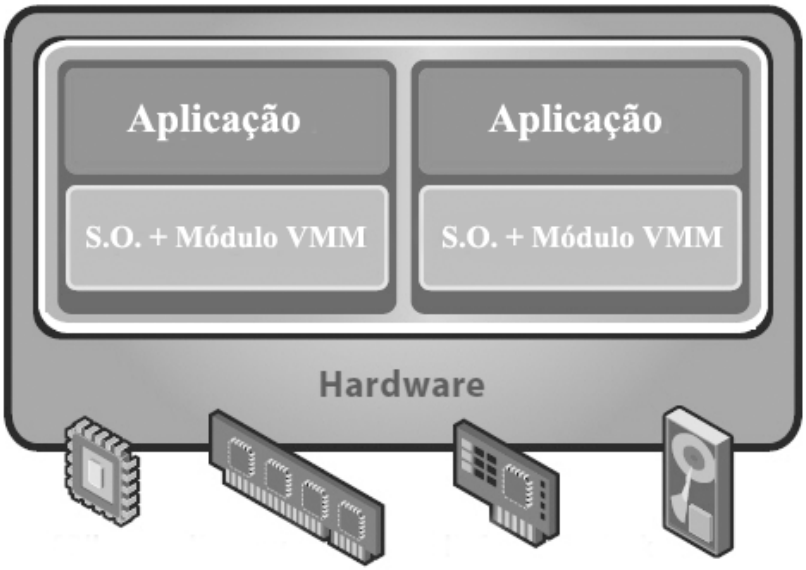


Figura 7.3: Arquitetura da Para-Virtualização

7.2.2.1. Propriedades

No contexto atual da computação, muito se fala sobre *Cloud Computing* ou computação em nuvem. Diversos trabalhos e pessoas estão associando seu trabalho/software a esta temática, mas diversas propriedades devem ser obedecidas para o ambiente ser considerado de fato uma *cloud*. Segundo a NIST (*National Institute of Standards and Technology*), cinco propriedades essenciais devem estar presentes em um projeto de *cloud*. Essas propriedades são:

- ***Auto-atendimento sob demanda (on-demand self-service)***: o usuário pode acessar os recursos da cloud e requisitar, de acordo com a sua demanda, a quantidade de recursos que será disponibilizada. O custo financeiro desta requisição do usuário poderá ser diretamente proporcional a quantidade de recursos requisitados. Se o usuário não puder dizer o que precisa, o ambiente perderá força para ser considerado como uma *cloud*;

- ***Amplo acesso a rede (broad network access)***: por ser utilizada por uma grande quantidade de usuários e por depender de infra-estrutura de comunicação adequada para sua execução, o ambiente deve possuir capacidade suficientemente grande para receber acessos simultâneos provenientes de diferentes origens. Um investimento reduzido na infra-estrutura de rede da *cloud* pode se tornar um gargalo para a sua utilização;

- ***Resource pooling***: em uma *cloud*, diversos recursos computacionais (como servidores) são disponibilizados para os usuários de forma transparente. O usuário requisita da *cloud* o que ele deseja e a mesma retorna para ele a demanda requisitada. O usuário não precisa saber que servidores estão processando a sua tarefa, e

muito menos onde esses servidores estão fisicamente (princípio de independência de localização);

- ***Elasticidade rápida (rapid elasticity)***: uma outra propriedade importante de uma *cloud* é a sua elasticidade; em suma, pode-se definir a elasticidade de uma *cloud* como sua capacidade de disponibilizar mais recursos a medida que isto for sendo requisitado pelo usuário (ou pela sua aplicação). O ambiente de *cloud* deve ser capaz de prover ao usuário, de forma rápida e padronizada, mecanismos para a requisição e disponibilização de mais recursos computacionais. Esta é uma das principais propriedades de uma Cloud; se a Cloud não tiver esta capacidade, sua própria caracterização como *cloud* fica comprometida;

- ***Serviços medidos (measured services)***: uma outra característica vital de uma *cloud* é a mensuração de seus serviços. O modelo de negócio deste tipo de ambiente é baseado no modelo pay-per-usage (pague pelo uso); desta forma, devem ser disponibilizadas formas de se medir a utilização de recursos e serviços. Um dos temas atuais nessa propriedade da *cloud* é a bilhetagem, e nele diversos esforços de pesquisa vêm sendo propostos. Se o ambiente de *cloud* não tiver como medir e cobrar pela sua utilização, o modelo de negócio associado a este novo paradigma pode estar fadado ao insucesso, tendo em vista que as empresas não terão algoritmos efetivos para efetuar a cobrança dos seus clientes.

Até este momento, foram apresentadas propriedades essenciais em um projeto de *cloud*; serão analisados agora algumas características desejáveis em uma Cloud.

Uma primeira característica desejável em uma *cloud* é a homogeneidade; se um conjunto homogêneo (produtos iguais ou similares) de recursos (softwares e hardwares) for utilizado em uma Cloud, tarefas como instalação e gerenciamento serão largamente simplificadas. Por produtos similares, deve-se entender um conjunto de softwares compatíveis e configurações/marcas de recursos computacionais (como servidores) similares (preferencialmente iguais).

Uma segunda característica fundamental é a de virtualização; utilizando este recurso, os recursos dentro da *cloud* podem ser melhor aproveitados. De fato, a utilização de virtualização pela *cloud* é um dos grandes avanços associados a esta nova tendência em comparação com outros ambientes bastante difundidos anteriormente, como *Grid Computing*.

A utilização de software de baixo custo, especialmente software livre, é outra característica marcante da *cloud*. O custo de implementação da mesma sofre uma redução significativa pela utilização de tal artifício, e este baixo custo acaba refletindo em um menor preço pelo serviço disponibilizado ao usuário final.

Por fim, uma característica também desejável (e até necessária) em um ambiente de *cloud* é a orientação a serviço. O provedor de recursos da Cloud deve propor alguma forma de cobrar a utilização de seu ambiente para os clientes; disponibilizando seus recursos como serviço, o provedor poderá tarifar a utilização de seus serviços para os clientes. Regras podem ser colocadas como SLAs (*service-level agreements* para padronizar e legalizar o negócio entre provedores de serviço e clientes. Isso trás mais confiabilidade

para o modelo de negócio e ajuda na adoção da *cloud* como opção para o parque de TI da empresa.

7.2.2.2. Classificação

Considerando o contexto variado de aplicações para a *cloud*, uma classificação atualmente bastante utilizada e difundidade é baseada no nível de serviços que a *cloud* oferta para o usuário.

- ***IaaS (Infrastructure-as-a-Service)*** é o nível que se encontra mais perto do hardware/ recursos físicos. Em geral, serviços neste nível provêem uma API de gerenciamento para a utilização de um conjunto de recursos físicos de forma que o usuário tenha acesso a funcionalidades como configurações automatizadas do sistema operacional, escalabilidade sob demanda e armazenamento. Como exemplo deste tipo de *cloud*, pode-se citar o Eucalyptus,Tycoon, Amazon EC2 e o OpenNebula;

- ***PaaS (Platform-as-a-Service)*** é um nível mais elevado de disponibilização de serviços, onde outros recursos (principalmente de software) são também disponibilizados para o usuário. Por recursos de software, pode-se entender ambientes para modelagem, programação e execução de programas/projetos. Um exemplo interessante de Cloud que se encontra neste nível é o Google App Engine, que oferece não apenas recursos de armazenamento, como um ambiente para programação onde usuários podem modelar e executar procedimentos na Cloud. Embora conte com a reconhecida vantagem de se disponibilizar mais um serviço para o cliente, um ponto também deve ser considerado nesse contexto; à medida que, em geral, os recursos físicos são alocados e utilizados através dos recursos de software do nível de plataforma, usuários têm uma restrição na forma de utilizar a Cloud. Por isso, pode-se afirmar que, quanto mais elevado o nível da Cloud, maior o número de serviços prestados para o cliente, e menor a flexibilidade para a utilização de recursos físicos e de software presentes na mesma;

- *SaaS (Software-as-a-Service)*, usuários finais podem acessar serviços que são disponibilizados diretamente pela Cloud para ele. Em geral, os clientes acessam estes serviços através de uma interface amigável (como um navegador Web). Exemplos de serviços disponibilizados pela Cloud, neste contexto, são o Google Maps e o Google Docs. Neste nível, o usuário não está interessado em desenvolver aplicações ou requisitar recursos físicos; seu desejo reside na execução de aplicações simples ou compostas que agregam valor de negócio para ele.

Um ponto relevante relacionado a estes conceitos reside na necessidade de se implementar os recursos acima descritos em uma estrutura de *cloud* para o ambiente ser considerado como Cloud. Por exemplo, se um serviço for implementado na Internet sem ser num ambiente de *cloud* (ex.: Eucalyptus), ele não pode ser caracterizado como uma Cloud do tipo software-as-a-service. Outro ponto também relevante é que as tecnologias de acesso sofrem uma forte dependência do nível de Cloud que está sendo trabalhado. A

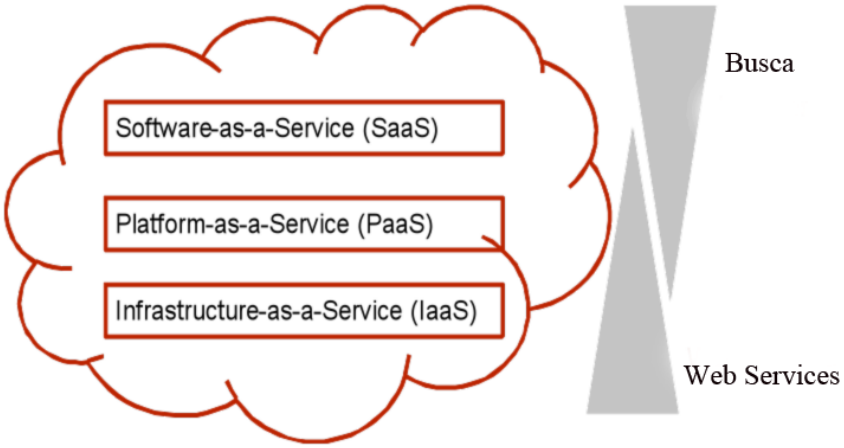


Figura 7.4: Tipos de Cloud x Tecnologias de Acesso

Figura 8.4 apresenta o relacionamento entre tecnologias de acesso e os três tipos de Cloud apresentados até este momento.

Uma outra classificação de *cloud* bastante difundida se baseia no modelo de implantação da mesma. De forma geral, usando este parâmetro, pode-se classificar as Cloud em Cloud Privada *(Private Cloud)*, Cloud Pública *(Public Cloud)*, Cloud Comunitária *(Community Cloud)* e Cloud Híbrida *(Hybrid Cloud)*.

Uma Cloud privada pode ser caracterizada como aquela operada unicamente pela própria empresa. Ela pode até ser gerenciada (em termos de infra-estrutura) por uma empresa terceirizada, mas acessível (em especial suas aplicações) apenas para os usuários internos da empresa.

Recursos físicos também são dedicados apenas as demandas computacionais internas da empresa. Uma Cloud Comunitária é caracterizada por um compartilhamento de infra-estrutura e/ou de recursos de software para um conjunto de parceiros específicos (que formam uma comunidade). Em geral, ela pode ter a sua gerência compartilhada pelas empresas participantes, por uma empresa terceirizada contratada ou até mesmo ser gerenciada por apenas uma das empresas participantes (e as outras apenas acessarem serviços da mesma).

Uma Cloud Pública é aquela disponibilizada para o público em geral (ou para um conjunto grande de usuários) e em geral disponibiliza serviços (que podem ou não incluir custos financeiros para a sua utilização) para um elevado número de usuários. O custo com recursos físicos neste tipo de Cloud é mais elevado, pois em geral a quantidade de acessos, além de elevada, pode sofrer mudanças (especialmente momentos de picos) de forma mais rápida e impactante.

Finalmente, uma Cloud híbrida é aquela composta por, no mínimo, duas outras Clouds (sejam elas privadas, públicas, híbridas ou uma combinação das mesmas). Um ponto importante a ser notado neste tipo de Cloud é a diferença de softwares/tecnologias que podem estar sendo empregadas, pois diferentes Clouds, em geral, terão uma heterogeneidade de tecnologias empregadas. Desta forma, um acordo de utilização deve ser feito para que os objetivos da união sejam atingidos.

7.2.3. Eucalyptus

O Eucalyptus [9] é um *framework opensource* para computação em núvem que implementa o modelo Infra-Estrutura como Serviço (IaaS), onde os usuários podem executar e controlar instâncias de máquinas virtuais instaladas em uma variedade de recursos físicos.

A arquitetura do Eucalyptus [2], mostrada na Figura 8.5, é composto por cinco componentes principais que juntos fornecem os serviços solicitados. Os componentes comunicam-se usando sistema de mensagens SOAP [8] de maneira segura através do WS-Security [11].

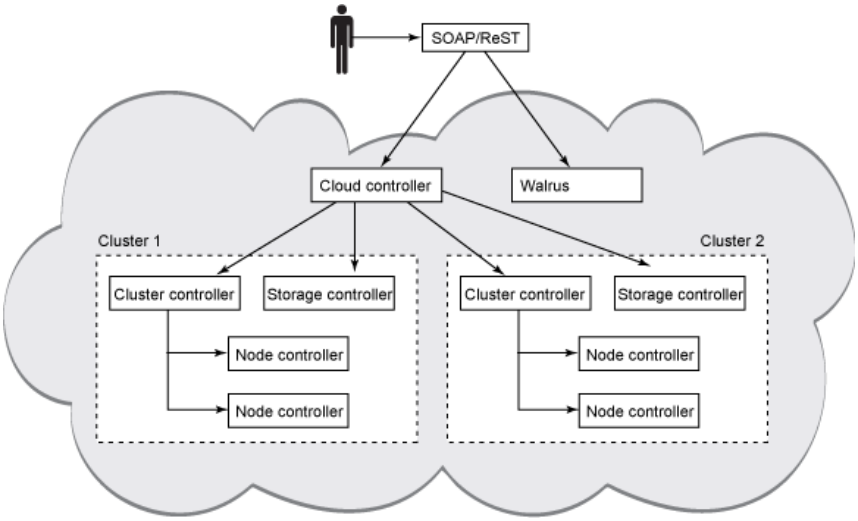


Figura 7.5: Arquitetura do Eucalyptus

- ***Cloud Controller***: É responsável pelas consultas aos nós e tomar decisões de escalonamento por meio de requisições aos *Cluster Controllers*. É a interface exposta a todos os usuários: administradores, projetistas, desenvolvedores e usuários finais. Todos os clientes se comunicam utilizando uma API baseada em SOAP ou REST [3];

- ***Cluster Controller***: É responsável por manter informações sobre a execução das maquinas virtuais, gerenciando assim toda a rede de instâncias virtuais. Os pedidos são comunicados ao *Cloud Controller* com o uso de interfaces baseadas em SOAP ou REST. Para iniciar uma nova instância ele envia requisições a um *Node Controller* com recursos disponíveis;

- ***Node Controller***: É responsável por gerenciar a execução, inspeção e finalização das máquinas virtuais. Executa em todo nó que hospedara uma instância de maquina virtual. Atualmente com suporte apenas ao Xen [12] e KVM [5], futuramente incluíra suporte ao VMWare [6];

- ***Storage Controller***: O serviço de armazenamento, baseado S3 [4] da Amazon, é responsável por armazenar e gerenciar as imagens das máquinas virtuais. As imagens são armazenadas de forma compactada e criptografada. Apenas durante a execução as imagens são decriptografadas e descompactadas;

- ***Walrus***: Permite aos usuários armazenarem os dados de maneira persistente no Eucalyptus, podendo criar e apagar listas de registros, buscar e apagar objetos, controlar política de acesso.

7.3. Montandado uma Cloud Privada

Nesta seção serão apresentados os detalhes que devem ser levados em consideração ao montar uma infraestrutura privada com o Eucalyptus. Inicialmente será definido a configuração do ambiente, depois mostrados os passos de instalação, cnfiguração, implantação de imagens e, por fim, uma introdução ao gerenciamento do cloud privada.

7.3.1. Definição do ambiente

Para instalar o Eucalyptus são necessárias pelo menos duas máquinas: uma para executar *Cloud Controller*, *Storage Controller* e *Cluster Controller* e outra para executar *Node Controller*. Podemos ter várias máquinas executando o *Node Controller*, aumentando assim a quantidade de recursos disponíveis e a quantidade de instâncias de máquinas virtuais que poderão ser executadas simultaneamente. É possível executar tudo em apenas uma única máquina, porém seria necessário ter uma especifição de hardware bem poderosa para prover todos os recursos necessários pelos componentes.

Antes de instalar e usar o Eucalyptus, é necessário avaliar os requisitos de hardware. A Tabela 8.1 mostra especificações para máquinas que executarão o *Cloud Controller* (CC), *Cluster Controller* (CLC), *Walrus* e *Storage Controller* (SC).

Tabela 7.1: Requisitos de Hardware para CC, CLC, SC e Walrus

Hardware	Mínimo	Sugerido
CPU	1 GHz	2 x 2 GHz
Memória	512 MB	2 GB
Disco	IDE 5400 rpm	SATA 7200 rpm
Espaco em Disco	40 GB	200 GB
Rede	100 Mbps	1 Gbps

As máquinas que executarão o *Node Controller* precisam ter uma configuração mais potente, pois eles irão executar instâncias das máquinas virtuais. A Tabela 8.2 mostra alguns exemplos

Tabela 7.2: Requisitos de Hardware para NC

Hardware	Mínimo	Sugerido
CPU	Extensão VT	VT, 64 bits, vários núcleos
Memória	1 GB	4 GB
Disco	IDE 5400 rpm	SATA, SCSI ou SAS 7200/10000 rpm
Espaco em Disco	40 GB	100 GB
Rede	100 Mbps	1 Gbps

7.3.2. Instalação e Configuração da Cloud

O Eucalyptus pode ser instalado a partir da compilação do código fonte ou através de pacotes específicos das distribuições Linux. Os pacotes prontos para instalação estão disponíveis para Redhat através do gerenciador de pacotes YUM e distribuições baseadas em Debian através do gerenciador de pacotes APT.

Uma alternativa à instalação através de pacotes, que é mais complicado, é utilizando a versão mais recente da distribuição Ubuntu Linux, o Ubuntu Enterprise Cloud (UEC), que fornece suporte completo para a execução de cloud baseada em Eucalyptus.

O Ubuntu Enterprise Cloud (UEC) é uma nova iniciativa de software livre da Ubuntu para facilitar a implementação, configuração e o uso de infraestruturas de cloud compatíveis com a interface dos serviços de providos pelo *Amazon WS* [4], EC2 e S3.

Inicialmente faça o download da versão mais recente do Ubuntu UEC (no momento, a 10.04), grave em um CD, selecione um computador para ser o CC, CLC, SC e Walrus. Este computador sera chamada daqui para frente de *frontend*.

Ao iniciar o *frontend* com o CD do Ubuntu UEC, escolha a opção *Install Ubuntu Enterprise Cloud* conforme mostrado na Figura 8.6. Os demais passos são os passos normais de uma instalação Linux: detecção do layout do teclado, formatação das partiçoes, muitas coisas são feitas automáticas e serão detalhadas apenas os passos referente à configuração do serviços de *cloud*.

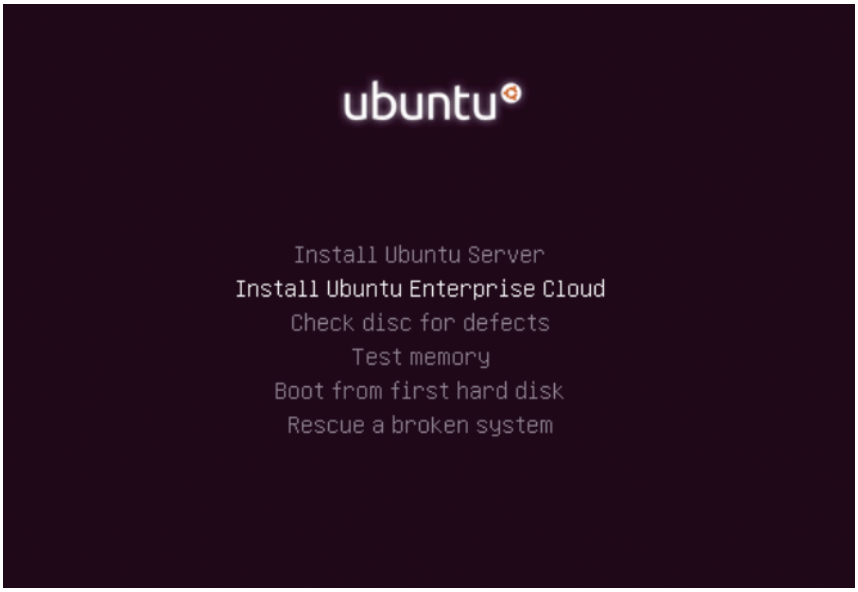


Figura 7.6: Instalação Eucalyptus: Ubuntu UEC

Após o passo de configuração do nome do host, o instalador ira informar que não encontrou nenhum *Cloud Controller* na rede e que caso não existe nenhum que seja pressionado ENTER, neste caso específico a instalação deverá ser continuada pressionando a tecla ENTER conforme mostrado na Figura 8.7

Na tela seguinte, conforme mostrado na Figura 8.8, deverão ser escolhidos quais os componentes deverão ser instalador no *fronted*, neste caso deverão ser escolhidos todos

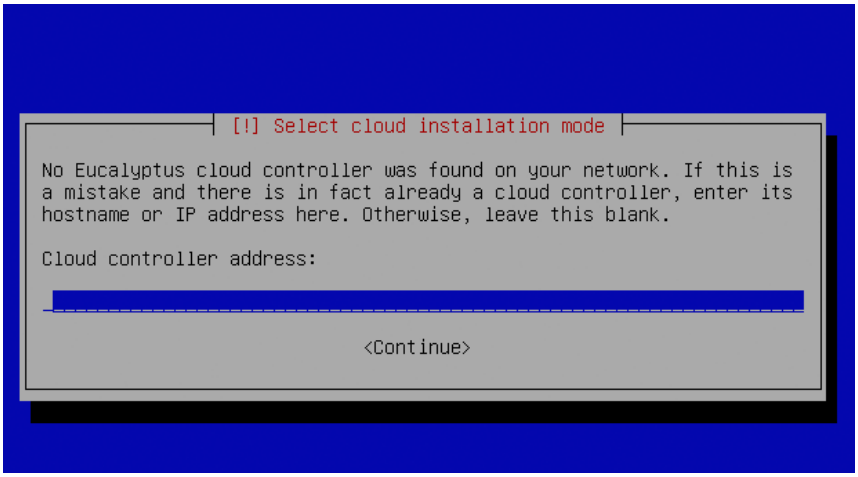


Figura 7.7: Instalação Eucalyptus: Cloud Controller

com excessão do *Node Controller*, que será instalador em outro computador.

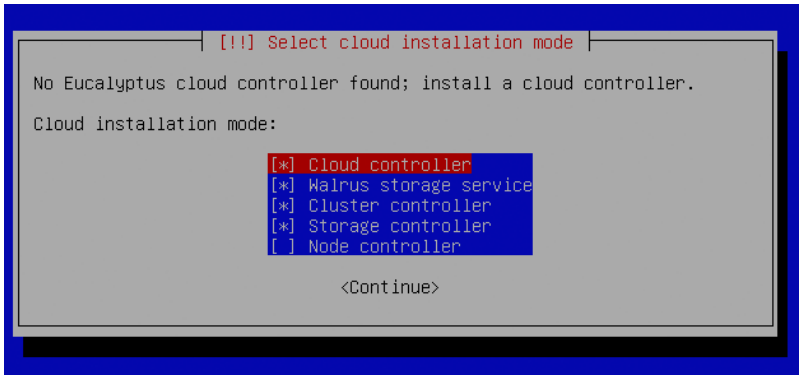


Figura 7.8: Instalação Eucalyptus: Seleção de Componentes

Será preciso especificar uma faixa de endereços IPs que serão atribuídas às instâncias das máquinas virtuais que executarão nos nós conforme mostrado na Figura 8.9. A faixa informada devera ser da classe C. No caso de nenhuma faixa ser informada, durante a execução da imagem deverá ser solicitado uma endereçamento privado explicidamente. Após esse passo será solicitado a remoção do CD de instalação e a reinicialização do *frontend*.

A instalação do *Node Controller* nos demais computadores é idêntica à instalação do *frontend*. Durante a instalção será detectado que já existe um *Coud Controller*, caso o *frontend* esteja ligado, e pedirá para selecionar o modo de instalação, neste caso específico deverá ser escolhido *Node Controller* para a instalação do nó conforme mostrado na Figura 8.10.

O registro dos nós no *frontend* é feito de forma automática, deste modo após o termino da instação a infraestrutura privada estara pronta para uso.

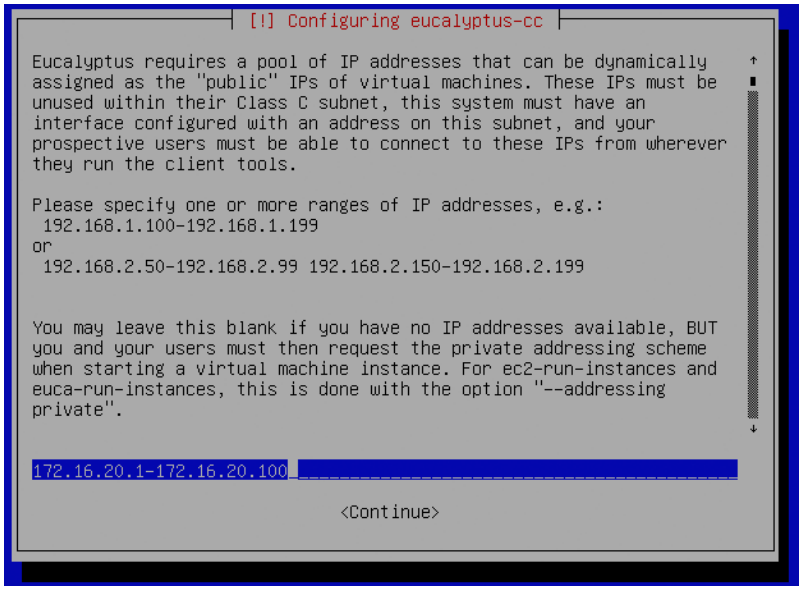


Figura 7.9: Instalação Eucalyptus: Endereçamento dos Nós

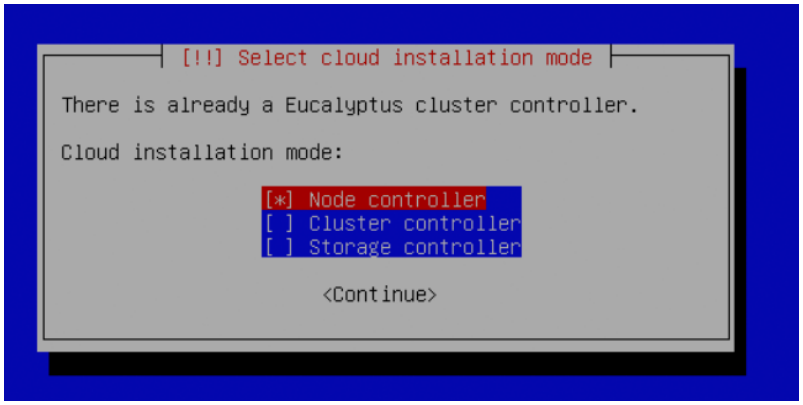


Figura 7.10: Instalação Eucalyptus: Configuração Node Controller

7.3.3. Gerenciamento da Cloud com o Eucalyptus

Após a instalação básica, um *frontend* (CC,CLC,SC e Walrus) e um nó (NC) a infraestrura fornecida pelo Eucalyptus estara pronta para uso. A administração e o gerenciamento podera ser feito WEB ou via linha de comando.

Uma interface de administração do Eucalyptus é fornecida pelo *frontend* na porta 8443. Devemos acessar esse serviço através de um navegador web requisitando o seguinte endereço: `https://192.168.142.135:8443`, onde 192.168.142.135 deverá ser substituído pelo endereço IP do *frontend*, para descobrir qual o endereço do *frontend* efetue *login* na máquina e execute o comando `ifconfig`. Após requisitar o endereço de gerenciamento uma tela, como mostrada na Figura 8.11, que solicitará um usuário e uma senha (o usuário e senha padrões são admin).

Depois do primeiro acesso a senha devera ser alterada por medida de segurança, conforme mostrado na Figura 8.12.

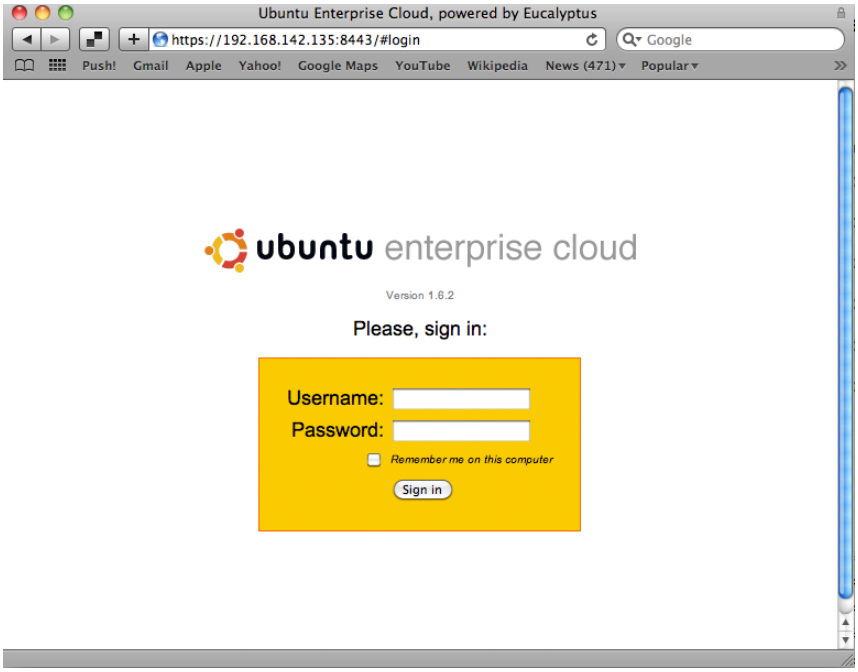


Figura 7.11: Gerenciamento da Cloud: Login

A interface de gerenciamento WEB possui algumas funcionalidas, como parametrização (CPU, memória e disco) dos tipos de instâncias, conforme mostrado na Figura 8.13.

A interface de linha de comando provê uma administração mais detalhada do Eucalyptus. A parte de gerenciamento do Eucalyptus relacionada ao registro dos diversos componentes (*node controller, cluster controller e walrus*) é realizada através do comando `euca_conf`. Alguns parâmetros do `euca_conf` são listados abaixo:

- **`-register-nodes`**: responsável pelo registro de *Node Controllers* que não foram registrados automaticamente durante o processo de instalação. A sua sintaxe é euca_conf `-register-nodes "endereço IP do Cloud Controller"`;
- **`-discover-nodes`**: faz uma busca por *Node Controllers* na rede local e faz automaticamente o registro;
- **`list-nodes`**: lista todos os nós registrados no *frontend*

Uma lista completa das opções do comando `euca_conf` é mostrada na Figura 8.14.

Para genrenciar as instâncias e os demais objetos da infraestrutura é utilizado o `euca-tool` uma ferramenta que possui diversos comandos e que foi instalado automaticamente pelo instakador do UEC. A Figura 8.15 mostra todos os comandos do `euca-tools`.

Dentre estes comando podemos destacar alguns de uso constante, como:

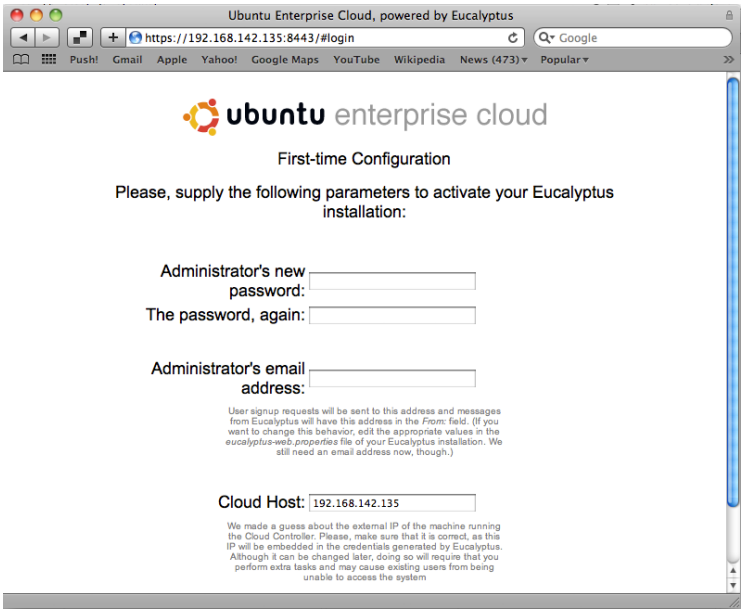


Figura 7.12: Gerenciamento da Cloud: Alterando Senha

- **`euca-describe-availability-zones verbose`**: mostra a quantidade de recursos disoníveis na *cloud* que poderão ser utilizados, ou seja, quantas instâncias podem ser executadas;
- **`euca-describe-instances`**: mostrada as instâncias que estão em execução;
- **`euca-describe-images`**: mostrada as imagens disponíveis para execução;
- **`euca-run-instances`**: executa uma determina imagem;
- **`euca-terminate-instances`**: finaliza a execução de uma determina instância;

Uma descrição dos comandos diponíveis para o gerenciamento do Eucalyptus está diponível na página do projeto: `http://open.eucalyptus.com/wiki/EucalyptusAdministratorGuide_v2.0`.

7.3.4. Implantação de uma Imagem na Cloud

A interface web de administração do Eucalyptus fornece uma maneira fácil para a implantação de uma imagem conforme mostrado na Figura 8.16:

1 Acessar a interface web através da url de administraçao: `https://<cloud-controller-ip-address>:8443`

2 Informar usuário e senha, caso necessário;

3 Escolher a aba **`STORE`**;

Figura 7.13: Gerenciamento da Cloud: Configuração de Parâmetos

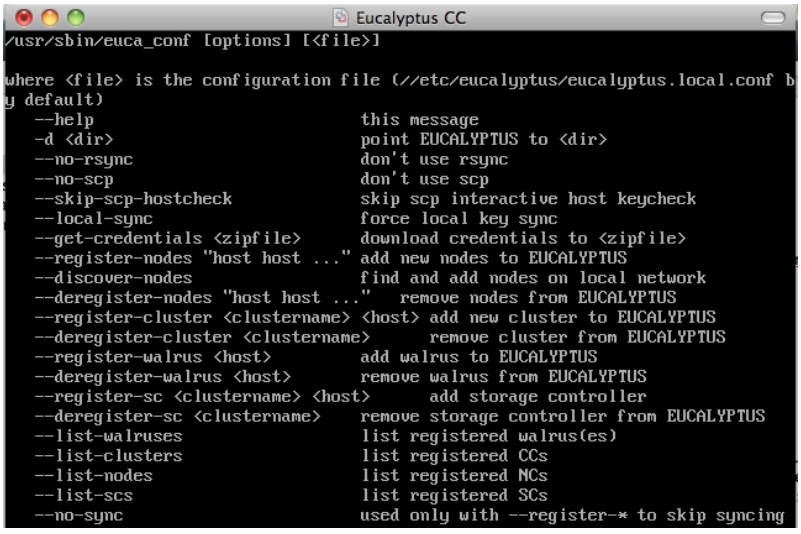


Figura 7.14: Gerenciamento da Cloud: euca_conf

4 Encontrar a imagem desejada dentro das disponíveis;

5 Instalar a imagem desejada utilizando o botão **`install`**.

Após o download automático da imagem escolhida, a mesma estara disponível para execução. Pode-se verificar as imagens disponíveis através da interface web, na aba ***IMAGES*** ou pelo comando **`euca-describe-images`** via linha de comando no *frontend*.

A execução da imagem poderá ser feita via linha de comando diretamente do *frontend*:

1 Antes de executar uma instância da imagem, sera necessário criar um par de chaves (chave SSH) para acessar a instância depois que ele iniciar. Este processo devera

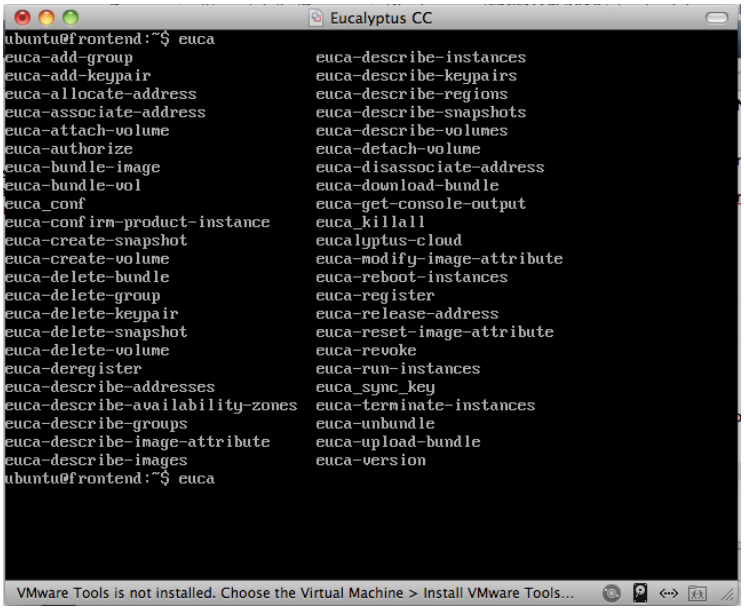


Figura 7.15: Gerenciamento da Cloud: euca-tools

ser feito apenas uma única vez, já que o par de chaves é armazenado no Eucalyptus. Para isso execute o seguinte comando:

2 Sera necessário liberar o acesso à porta 22 (SSH) da instância através do comando:
`euca-authorize default -P tcp -p 22 -s 0.0.0.0/0`

3 Executar o comando **`euca-describe-images`** para obter o EMI (*Eucalyptus Machine Images*) da imagem selecionada:

4 Executar a imagem registrada como o comando:
`euca-run-instances emi-E073107E -k mykey -t m1.small`

5 Para monitorar o status da instância utilize o comando:
`watch -n5 euca-describe-instances`

7.4. Considerações Finais

O uso do Eucalyptus para montar uma infraestrutura privada em um ambiente de computação em núvem tem-se mostrado uma solução robusta, pois além de ser uma ferramenta *opensource*, é compatível com a infraestrutura privada do Amazon AWS.

Utilizando hardware mais acessível e gastando apenas algumas horas instalando a versão do *Ubuntu Enterprise Cloud* que já vem incorporado com o Eucalytus, o que torna o processo de configuração mais fácil.

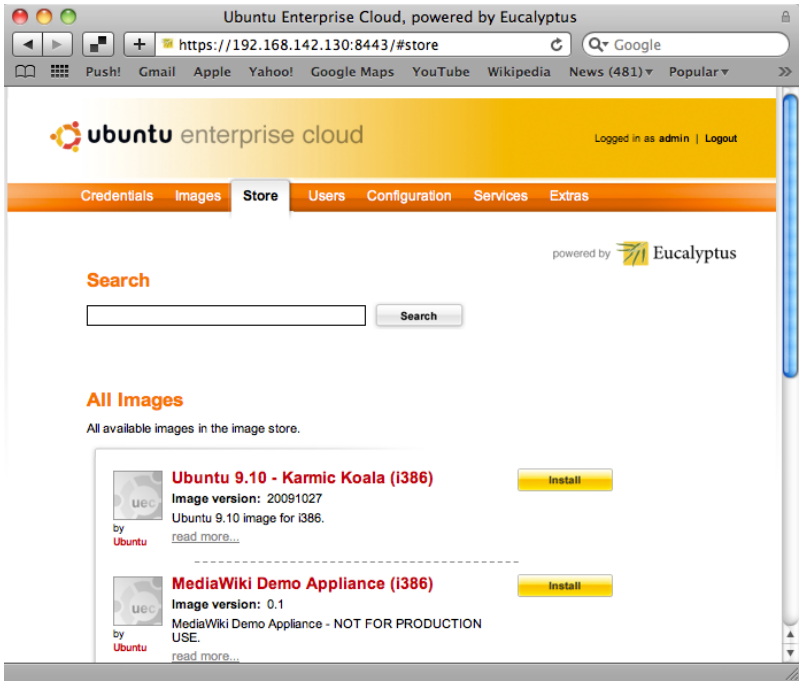


Figura 7.16: Gerenciamento da Cloud: Instalação de Imagem

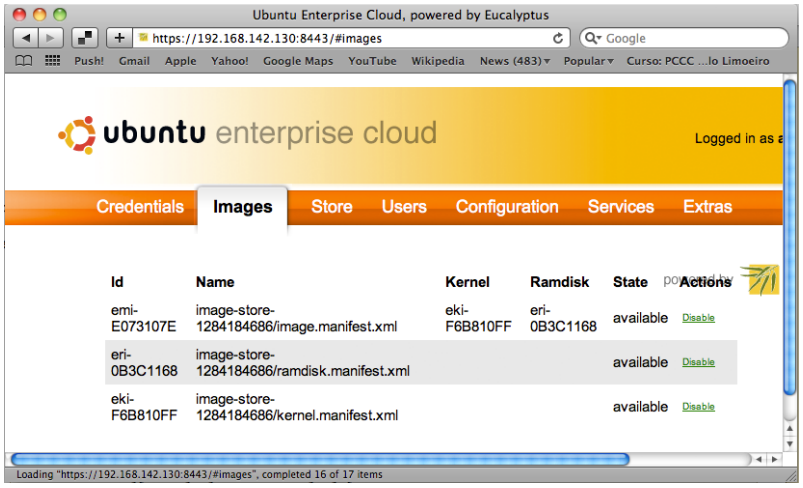


Figura 7.17: Gerenciamento da Cloud: Imagens Disponíveis

```
if [ ! -e ~/.euca/mykey.priv ]; then
    mkdir -p -m 700 ~/.euca
    touch ~/.euca/mykey.priv
    chmod 0600 ~/.euca/mykey.priv
    euca-add-keypair mykey > ~/.euca/mykey.priv
fi
```

Figura 7.18: Gerenciamento da Cloud: Criação das Chaves SSH

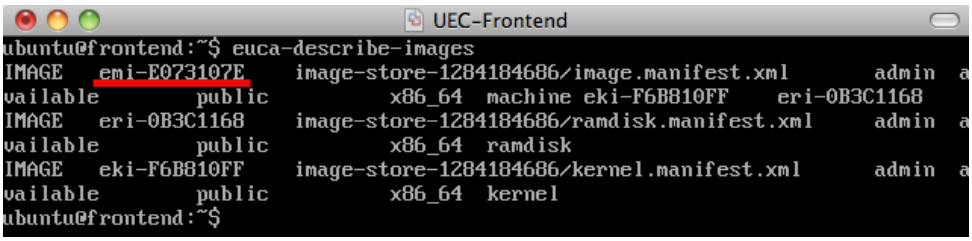


Figura 7.19: Gerenciamento da Cloud: Obtendo o EMI

Referências

[1] Alexandre Carissimi. Virtualizacao: da teoria a solucoes. *Minicursos do Simposio Brasileiro de Redes de Computadores - SBRC 2008*, pages 173–207, 2008.

[2] Prabhakar Chaganti. Servicos em nuvem para sua infraestrutura virtual, parte 1: Infrastructure as a-service (iaas) e eucalyptus. Technical report, Abril 2010.

[3] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.

[4] Garfinkel, Simson L. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical report.

[5] Irfan Habib. Virtualization with kvm. *Linux J.*, 2008(166):8, 2008.

[6] Scott Lowe. *Mastering VMware vSphere 4 (Computer/Tech)*. Sybex, 1 edition, 8 2009.

[7] Daniel A. Menasce. Virtualization: Concepts, applications, and performance modeling. *Int. CMG Conference*, pages 407–414, 2005.

[8] N. Mitra. Soap version 1.2, w3c recommendation, 24 june 2003. Technical Report August 9, W3C, 2003. http://www.w3.org/TR/soap12-part1/.

[9] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 124 –131, May 2009.

[10] Robert Rose. Survey of system virtualization techniques., Acessado em Outubro de 2010. 2010. http://citeseer.ist.psu.edu/rose04survey.html.

[11] Jothy Rosenberg and David Remy. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Sams, May 2004.

[12] Chris Takemura and Luke S. Crawford. *The Book of Xen: A Practical Guide for the System Administrator*. No Starch Press, 1 edition, 10 2009.

ERCEMAPI 2010

Escola Regional de Computação Ceará - Maranhão - Piauí

Livro Texto dos Minicursos

ISBN
978-85-7669-251-5