Anais do XVI SEMINCO

Seminário de Computação
3 a 5 de outubro de 2007
FURB - Campus I - Blumenau/SC

Promoção

Universidade Regional de Blumenau - FURB
Pró-Reitoria de Extensão e Relações Comunitárias - PROERC
Centro de Ciências Exatas e Naturais - CCEN
Departamento de Sistemas e Computação - DSC
Centro Acadêmico Livre de Computação - CALCOMP

Comissão Organizadora

Prof. Everaldo Artur Grahl (Coordenador)
Prof. Adilson Vahldick
Prof. Alexander Roberto Valdameri
Prof. Antônio Carlos Tavares
Prof. Dalton Solano dos Reis
Prof. Jomi Fred Hübner

Ficha Catalográfica Elaborada pela Biblioteca da FURB

Seminário de Computação (16.: 2007 : Blumenau, SC)

Anais do XVI SEMINCO / promoção Universidade Regional de Blumenau, Departamento de Sistemas e Computação; Everaldo Artur Grahl, Dalton Solano dos Reis (coordenador). - Blumenau, O Departamento, 2007. 147 p. : il.

1. Computação - Congressos. I. Grahl, Everaldo Artur; Reis, Dalton Solano dos. II. Universidade Regional de Blumenau. Departamento de Sistemas e Computação. III. Título.

CDD 004

Apresentação

A Universidade Regional de Blumenau - FURB, através do Departamento de Sistemas e Computação, realiza o XVI Seminário de Computação (SEMINCO) entre os dias 3 e 5 de outubro de 2007.

Este ano tivemos a submissão de 33 artigos provenientes de várias instituições e empresas do país, sendo que destes foram aprovados 12 artigos das seguintes universidades e empresas: CEFET, ENGSOFT, FINOM, FURB, IPTEC, PUC-RS, UFU, UNISINOS, UNIVALI e USP. A organização dos artigos nestes anais é feita conforme as seguintes áreas de conhecimento: Banco de Dados, Computação Gráfica, Engenharia de Software, Informática na Educação, Inteligência Artificial e Sistemas de Informação.

Agradecemos a todos os envolvidos na organização do evento, bem como a Comissão de Avaliação Interinstitucional que não mediu esforços em avaliar os diversos artigos submetidos à chamada de trabalhos. Esperamos que nos três dias de realização do evento as expectativas dos participantes sejam atendidas e tenhamos um grande evento. Até o ano que vem!

Comissão Organizadora

Agradecimentos

Sociedade Brasileira de Computação - SBC
Comissão de Avaliação Interinstitucional
Fundação de Apoio à Pesquisa Científica e Tecnológica do Estado de Santa Catarina - FAPESC
Projeto Acredito - FURB

Comissão de Avaliação Interinstitucional

Adelmo Luis Cechin (UNISINOS - RS)
Adriana G. Alves (UNIVALI - SC)
Alexander Roberto Valdameri (FURB - SC)
Ana Lúcia Anacleto Reis (FURB - SC)
Andrio Pinto (UNISINOS - RS)
Anita da Rocha Fernandes (UNIVALI - SC)
Dalton Solano dos Reis (FURB - SC)
Denio Duarte (Unochapeco - SC)
Douglas M. da Silva (UNISINOS - RS)
Everaldo Artur Grahl (FURB - SC)
Fabiane Barreto Vavassori (FURB - SC)
Fabio Rafael Segundo (FURB - SC)
Fernando Santos Osório (UNISINOS - RS)
Gerson Cavalheiro (UNISINOS - RS)
Jomi Fred Hübner (FURB - SC)
Lucas Ferrari de Oliveira (UFPel - RS)
Luiz Carlos Ribeiro Junior (UNISINOS - RS)
Marcel Hugo (FURB)
Marcello Thiry (UNIVALI - SC)
Maurício Capobianco Lopes (FURB - SC)
Mauro Marcelo Mattos (FURB - SC)
Paulo Cesar Rodacki Gomes (FURB - SC)
Paulo Fernando Silva (UFSC - SC)
Paulo Roberto Ferreira Jr. (UFRGS - RS)
Rafael Cancian (UNIVALI - SC)
Rafael Heitor Bordini (University of Durham - UK)
Reinaldo A. C. Bianchi (FEI - SP)
Valguima Victoria Vianna Aguiar Odakura (USP - SP)
Vera R. N. Schuhmacher (UNISUL - SC)
Vitor Fernando Pamplona (UFRGS - RS)

Artigos Selecionados

Banco de Dados

Computação Gráfica

Engenharia de Software

Informática na Educação

Inteligência Artificial

Sistemas de Informação

WebModeler: uma ferramenta CASE para modelagem de banco de dados relacional na web

Alexander R. Valdameri[1], Juarez Bachmann[1]

[1]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

arv@furb.br, juarez.bachmann@gmail.com

Resumo. *Este trabalho apresenta um aplicativo para definição do modelo lógico nos projetos de bancos de dados relacionais, desenvolvido para o ambiente web utilizando a plataforma Java, a arquitetura MVC e alguns outros padrões de projeto, como DAO e Command. Para tornar a interface do sistema mais amigável ao usuário, foi utilizado o conjunto de técnicas conhecido por Ajax, disponibilizando recursos semelhantes aos de aplicativos desktop, o que permite inclusive que sejam criados diagramas para modelar as tabelas, colunas e relacionamentos de forma gráfica.*

1. Introdução

A variedade de ferramentas CASE existentes no mercado com o propósito de auxiliar na modelagem de bancos de dados relacionais é grande, sendo que a maioria delas têm características em comum, como a possibilidade de modelar os dados de forma gráfica (desenhando tabelas e relacionamentos) e salvar os modelos criados em arquivos binários, que podem ser enviados a outras pessoas e utilizados por elas, desde que estas utilizem a mesma ferramenta. Outra característica marcante presente nestes softwares é dificuldade de manutenção sobre os mesmos, visto que são aplicações *desktop.* Além do custo para aquisição e instalação, têm a manutenção, que gera certa dificuldade para atualização caso estejam instaladas em inúmeras máquinas dentro de uma empresa, pois a atualização precisará ser feita em cada uma delas.

Uma alternativa interessante para amenizar eventuais problemas decorrentes das características dos aplicativos *desktop* é a internet, que vem passando por transformações significativas nos últimos anos. Neste momento, a web está virando uma plataforma: todo tipo de aplicativo é executado no *browser*. Inicialmente, foram os serviços de e-mail que saíram do *desktop* e foram disponibilizados na grande rede mundial de computadores. Agora, já há editores de textos, planilhas, agendas, diários, álbuns de fotos, enciclopédias e muitos outros serviços como estes. Além disso, diversos softwares corporativos são executados nos navegadores, como os portais de compras, por exemplo.

Diante deste contexto, pode-se dizer que há uma forte tendência em transformar os aplicativos *desktop* em ferramentas para o ambiente web e os softwares de modelagem de banco de dados também podem ser enquadrados nesta situação. Sendo assim, o objetivo do presente artigo é apresentar o desenvolvimento e operacionalidade de um aplicativo web (chamado WebModeler) que permite a definição de modelos lógicos para projetos de banco de dados relacional.

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 7-18.

2. Desenvolvimento da ferramenta

A ferramenta desenvolvida implementa os casos de uso exibidos na Figura 1.

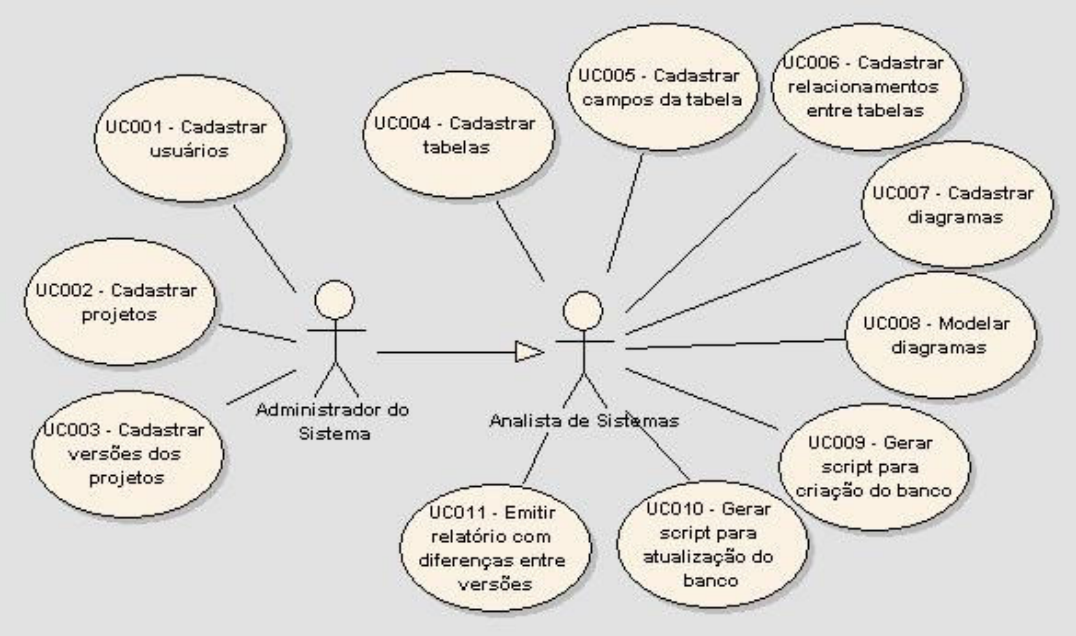


Figura 1. Casos de uso do sistema

Desta forma, as funcionalidades presentes neste software são:

- Cadastramento dos usuários que poderão acessar o sistema.
- Cadastramento dos projetos controlados.
- Ligação entre projetos e usuários, definindo permissões de acesso.
- Controle das versões de cada projeto, a fim de manter o registro histórico das alterações feitas no projeto com o passar do tempo.
- Cadastramento das tabelas existentes em cada versão, assim como suas colunas e relacionamentos. Para as colunas, é possível informar um tipo de dado, bem como definir se a mesma é parte da chave primária da tabela ou não. É possível ainda efetuar este cadastramento graficamente, na forma de diagramas.
- Geração de *scripts* com os comandos SQL para a criação do banco de dados nos SGBD Oracle e MSSqlServer e também para a atualização do banco de dados nestes SGBD com base na comparação entre duas versões de um projeto.
- Emissão de um relatório com a descrição das diferenças existentes entre uma versão e outra de um projeto, identificando tabelas e colunas que tenham sido incluídas, alteradas ou excluídas.

Nas seções 2.1, 2.2 e 2.3 há um detalhamento das técnicas e tecnologias mais relevantes empregadas no desenvolvimento deste aplicativo. Já a seção 2.4 descreve o processo de implementação da ferramenta.

2.1. Ambiente de desenvolvimento

Para a implementação foi utilizada a linguagem Java, de acordo com a especificação JEE (até pouco tempo chamada J2EE), sendo que o ambiente de desenvolvimento utilizado foi o Eclipse, na versão 3.2. O servidor de aplicações utilizado para executar o software foi o Apache Tomcat (versão 5.5) e o SGBD utilizado foi o MySql.

2.2. Padrões de projeto de software

Bond et al. (2003) descreve os padrões dentro da área de software como uma forma de capturar parte da experiência dos arquitetos e projetistas de sucesso, para que ela possa ser aplicada mais amplamente, ou seja, "um padrão é uma idéia reutilizável sobre como resolver um problema em particular, encontrado no domínio da arquitetura ou projeto" [Bond et al., 2003].

Os padrões podem ser divididos em tipos, sendo que segundo Bond et al. (2003), os mais comuns são os padrões de arquitetura (que definem o estilo geral do sistema, como a quantidade de camadas que a aplicação terá e o relacionamento entre elas) e os padrões de projeto (que se encontram no nível dos artefatos como classes e componentes). Além destes, também há os chamados padrões do JEE, que são um conjunto de padrões que têm sido identificados dentro das soluções com base no JEE, padrões gerais já identificados e utilizados anteriormente em outras plataformas [Bond et al., 2003].

A seguir, são descritos os padrões utilizados neste trabalho.

2.2.1. *Model-View-Controller* (MVC)

Gamma et al. (2000), descreve MVC como um padrão de arquitetura que divide o software em três tipos de objetos para aumentar a flexibilidade e a reutilização. O primeiro é o Modelo (*Model*), que consiste na camada onde ocorre o processamento das informações, onde os dados são gravados nos bancos de dados e recuperados dos mesmos, ou seja, é no Modelo que ficam as regras de negócio. O segundo tipo de objeto é a Vista ou Visão (*View*), cuja função é a de apresentar as informações ao usuário, isto é, este objeto é a interface do sistema. O terceiro e último tipo é o Controlador (*Controller*), que define como a interface reage às entradas do usuário. A Visão sempre deve refletir a situação atual do Modelo, sendo que o Controlador é responsável por controlar o fluxo entre as duas outras camadas.

2.2.2. *Data Access Object* (DAO)

Segundo Bond et al. (2003) o DAO é um padrão do JEE que visa criar objetos que encapsulem o acesso aos dados por trás de uma interface comum, que pode ser implementada de diferentes formas para diferentes fontes de dados.

Sullivan (2003) explica que os desenvolvedores JEE utilizam este padrão para separar a lógica de acesso a dados (baixo nível) da lógica das regras de negócio (alto nível). Com isso, de acordo com Bond et al. (2003), ficam amenizados dois problemas que ocorrem com freqüência nos sistemas onde o código de acesso a dados é mesclado com a lógica do negócio: dificuldade de manutenção e pouca flexibilidade. Isto ocorre porque ao utilizar este padrão, uma alteração no código de acesso a dados só precisará

ser feita no método específico da classe DAO em questão, não sendo necessário procurar e alterar inúmeros pontos do sistema que possuem os mesmos comandos SQL.

2.2.3. *Command*

Segundo Gamma et al. (2000) o padrão *Command* visa encapsular uma solicitação como um objeto, permitindo parametrizar clientes com diferentes solicitações, fazer o registro (log) de solicitações e também suportar operações que podem ser desfeitas.

Este padrão, também conhecido como *Action* ou *Transaction*, possibilita também estruturar um sistema em torno de operações de alto nível, estrutura esta muito comum em sistemas que suportam transações. Assim, pode-se dizer que o padrão *Command* fornece uma maneira de modelar transações [Gamma et al., 2000].

O princípio básico desse padrão é a criação de uma classe abstrata `Command`, com uma operação abstrata `execute()`, sendo que esta operação é implementada nas subclasses concretas de `Command`. Em um editor de textos hipotético, cada botão do menu que fosse acionado criaria uma instância de uma subclasse de `Command`, de acordo com sua funcionalidade. Um botão chamado Salvar Documento, por exemplo, instanciaria um objeto da classe `SalvarCommand` e em seguida dispararia o seu método `execute()`, sendo que o mesmo aconteceria para os outros botões e classes. Desta forma, cada operação é executada na sua totalidade por uma classe específica, facilitando a compreensão e manutenção do sistema.

2.2.4. *Front Controller*

Em Bond et al. (2003) o *Front Controller* é definido como um padrão JEE no qual um componente intercepta a requisição do usuário e direciona ou agrega valor a mesma.

A definição encontrada em Sun Microsystems (2002b) diz que este padrão consiste em definir um componente simples que seja responsável pelo processamento das requisições da aplicação. Este componente recebe a requisição, encaminha-a para o componente responsável por processá-la (outra classe ou *servlet*) e após o processamento seleciona a página que deve ser apresentada ao usuário. Além disso, este componente centraliza funções relativas à segurança e tratamento de erros, facilitando a manutenção do software [Sun Microsystems, 2002a].

2.3. Ajax

O termo Ajax foi criado por Jesse James Garret em fevereiro de 2005, como um acrônimo para *Asynchronous JavaScript and XML*, mas atualmente é usado para englobar todas as tecnologias que possibilitam ao navegador se comunicar com o servidor sem atualizar toda a página atual [Asleson e Schutta, 2006].

Segundo Asleson e Schutta (2006), o Ajax pode ser considerado como uma técnica e não uma tecnologia específica. Esta técnica agrega tecnologias como JavaScript, *eXtensive Markup Language* (XML), *Cascading Style Sheets* (CSS), *Document Object Model* (DOM) e o objeto do JavaScript chamado `XMLHttpRequest` para tornar as páginas web mais dinâmicas, disponibilizando recursos como criação de menus de contexto que aparecem ao clicar com o botão direito do *mouse*, possibilidade de deslizar controles (arrastar e soltar) e trocar informações com o servidor de forma assíncrona, sem a necessidade de atualizar toda a página ao

fazê-lo. Todas estas tecnologias são tratadas no *browser*, ou seja, "o Ajax é uma abordagem do lado cliente e pode interagir com a JEE, .NET, PHP, Ruby e scripts CGI – realmente não depende do servidor" [Asleson e Schutta, 2006], sendo que a tecnologia mais recente (e talvez mais importante) relacionada ao termo, é o objeto `XMLHttpRequest`, responsável pela comunicação assíncrona entre o navegador e o servidor web.

2.4. Processo de implementação da ferramenta WebModeler

A implementação do sistema ocorreu conforme os passos descritos nas seções seguintes.

2.4.1. Desenvolvimento da camada de persistência e recuperação dos dados

Nesta etapa foram desenvolvidas as classes que implementam o padrão de projeto DAO, ou seja, as classes responsáveis por gravar e recuperar as informações no banco de dados. Para cada tabela do banco de dados da aplicação foi criada uma classe DAO correspondente, com métodos para inserir, alterar, excluir e listar seus respectivos registros. A Tabela 1 apresenta os principais métodos criados na classe UsuarioDAO.

Tabela 1 – Principais métodos da classe UsuarioDAO

Método	Objetivo
carregar(int)	Retorna um objeto do tipo Usuario, com os dados do usuário cujo código foi passado como parâmetro.
excluir(int)	Exclui do banco de dados o usuário cujo código foi passado como parâmetro.
gravar(Usuario)	Recebe um objeto do tipo Usuario como parâmetro e insere/altera os dados do mesmo no banco.
listar()	Retorna uma lista de objetos do tipo Usuario com todos os usuários cadastrados no sistema.

2.4.2. Desenvolvimento do núcleo de processamento de requisições do sistema

A parte do sistema aqui nomeada como "núcleo de processamento de requisições" nada mais é do que a implementação do padrão *Front Controller* em conjunto com o padrão *Command*, além de ser a camada *Controller* da arquitetura MVC.

Seguindo o que foi exposto anteriormente acerca do padrão *Front Controller*, criou-se uma classe de mesmo nome, a qual consiste num *servlet* Java que, segundo o que foi definido no arquivo de configuração web.xml, é responsável por atender todas as requisições cuja *url* termine com ".do".

Para o tratamento das requisições foi empregado o padrão de projeto *Command*. Com isto, foi implementada uma super-classe com o mesmo nome deste padrão, a qual possui apenas um método denominado `executar()` onde a lógica de cada `Command` foi escrita, ou seja, é neste método que ocorre o processamento das requisições e é onde se encontram as regras de negócio do software. Esta classe foi estendida para implementar o atendimento de cada tipo de requisição existente em cada módulo do sistema. Para atender a requisição de exclusão de um usuário, por exemplo, o

`FrontController` recebe a requisição de *url* "usu_excluir.do", instancia um objeto da classe `ExcluirUsuarioCommand` e dispara seu método `executar()`, sendo este funcionamento idêntico para as demais requisições.

2.4.3. Implementação dos cadastros básicos do sistema

Concluído o desenvolvimento da camada de persistência e do núcleo de processamento de requisições, foram implementadas as rotinas relativas aos cadastros de projetos, versões, usuários, tabelas, colunas, relacionamentos e diagramas.

Nesta fase, o esforço foi concentrado no desenvolvimento das telas (páginas jsp) com os campos necessários para cada cadastro e também das respectivas classes `Command` responsáveis pela recuperação, inclusão, alteração e exclusão dos registros. As telas de cadastro se comunicam assincronamente com o servidor através do objeto `XMLHttpRequest`, isto é, sempre que um botão é acionado, um código JavaScript é executado para criar o objeto `XMLHttpRequest` e enviar a requisição ao servidor. Após o servidor processar a solicitação através de uma classe `Command`, o mesmo devolve para o *browser* um arquivo XML que é tratado por outro código JavaScript, atualizando somente os valores dos campos ao invés de atualizar a página inteira, como ocorre em aplicações web tradicionais.

2.4.4. Implementação do módulo de modelagem gráfica

A modelagem de um banco de dados no sistema de forma gráfica visa disponibilizar ao analista uma interface amigável, na qual seja possível visualizar facilmente a estrutura do banco que está sendo projetado. Essa interface exibe, basicamente, as tabelas cadastradas (com suas colunas) e os relacionamentos entre elas, sendo as tabelas representadas por meio de retângulos em cujo interior aparecem as colunas e seus respectivos tipos de dados. Já os relacionamentos são representados por linhas conectadas às duas tabelas relacionadas, com uma seta na extremidade ligada à mestre.

No desenvolvimento desta parte do sistema, o Ajax foi utilizado tanto ou mais do que nas telas de cadastro. Utilizando JavaScript, DOM, CSS e o objeto `XMLHttpRequest` foi possível criar recursos para exibir as tabelas e relacionamentos, criar menus de contexto (botão direito do *mouse*) e arrastar e soltar (*drag-and-drop*) as tabelas. As tabelas são "desenhadas" na tela através da criação de um objeto HTML do tipo DIV (*tag* da linguagem HTML), objeto este capaz de definir uma área do HTML que pode ser formatada diferentemente do restante da página, além de reconhecer eventos como clique do *mouse* ou acionamento do teclado. Para que seja possível arrastar e soltar as tabelas, basicamente foi necessário associar uma função JavaScript ao evento `onmousemove` da página, função esta que trata o momento em que o usuário move o *mouse*, reposicionando o DIV selecionado de acordo com a posição do *mouse.*

Para que o usuário possa informar os dados da tabela e das suas colunas, foram disponibilizadas duas formas de acesso à tela de cadastro de tabelas: através de um duplo-clique sobre o DIV ou através do menu de contexto que aparece ao clicar com o botão direito do *mouse* sobre o mesmo. Ambas as opções levam o usuário à tela de cadastro de tabelas (a mesma que já havia sido desenvolvida anteriormente) que é aberta em uma nova página do *browser*. Após informar os dados desejados, o usuário

aciona um botão que fecha esta tela e faz com que, através de uma outra função JavaScript, o DIV relativo à tabela em questão seja redesenhado para exibir corretamente o nome da tabela e os dados de suas colunas. Esta função que atualiza o DIV na tela utiliza o objeto `XMLHttpRequest` para obter as informações necessárias do servidor assincronamente, sem a necessidade de atualizar a página inteira, ou seja, apenas o DIV é atualizado.

Para cadastrar um relacionamento, foi incluída uma opção no menu de contexto das tabelas, a qual abre a tela de cadastro de relacionamentos em uma nova página do *browser*. Depois de feito o cadastro, ao pressionar o botão para adicionar o relacionamento no diagrama, a tela de cadastro é fechada e o relacionamento é desenhado na tela. Para formar a linha e a seta que representam um relacionamento, são criados inúmeros pequenos DIVs, cada um ocupando 2 *pixels* de altura e outros 2 de largura, sendo que foi implementado um algoritmo que, a partir das posições das tabelas mestre e filha no diagrama, cria estes inúmeros DIVs até conectar uma tabela à outra.

Por fim, foi implementada a rotina de gravação da modelagem gráfica, para que os dados das tabelas e relacionamentos adicionados no diagrama sejam salvos no banco de dados, onde também foi utilizado o objeto `XMLHttpRequest` para que a gravação seja feita assincronamente.

2.4.5. Implementação do módulo de geração de *scripts*

Após a modelagem há a possibilidade de gerar arquivos com os comandos SQL necessários para criar um banco de dados e também para atualizar um banco com base nas alterações feitas entre uma versão de um projeto e outra. Até o momento, o sistema faz a geração destes *scripts* apenas para os SGBD Oracle 10g e MSSqlServer 2000, podendo suportar outros bancos em versões futuras. Na tela do sistema onde são cadastradas as tabelas, o usuário informa tipos de dados próprios do sistema para as colunas, com nomenclaturas em Língua Portuguesa (Numérico, Texto Variável, Data, etc.) ao invés de tipos próprios de algum SGBD (Varchar, Number, Int, etc.), sendo que os tipos próprios do sistema e os tipos nativos dos SGBD foram relacionados conforme descrito na Tabela 2.

Tabela 2 – Relacionamento entre os tipos de dados

WebModeler	**Oracle**	**MSSqlServer**
Texto Variável	Varchar2	Varchar
Texto Fixo	Char	Char
Inteiro	Integer	Int
Inteiro Curto	Smallint	Smallint
Inteiro Longo	Integer	Bigint
Numérico	Decimal	Decimal
Data	Date	DateTime
Data/Hora	DateTime	DateTime
Binário	Blob	Binary

O relacionamento descrito na Tabela 2 foi elaborado com base nas informações de Oracle (2005) e Battisti (2001) e, uma vez definido este relacionamento, foram desenvolvidas as classes e rotinas para geração dos *scripts* de criação do banco de dados. Logo em seguida, foram implementadas as classes responsáveis por gerar os *scripts* de atualização de bancos de dados através da comparação entre duas versões de um projeto. Estas classes utilizam o relacionamento apresentado na Tabela 2 para gerar os comandos SQL para cada banco, sendo que é utilizada a sintaxe do SQL padrão e não é possível personalizar os tipos de dados a serem utilizados para cada banco ou a sintaxe dos comandos. Por fim, foram desenvolvidas as classes que geram um relatório indicando quais tabelas e colunas foram adicionadas, alteradas ou excluídas de uma versão para a outra. Tanto os *scripts* quanto o relatório são apresentados ao usuário no *browser* na forma de um arquivo PDF único.

3. Operacionalidade e principais telas da ferramenta

A Figura 2 exibe um diagrama que ilustra o fluxo de trabalho no WebModeler, mostrando a divisão das funcionalidades disponíveis para o Administrador e para o Analista bem como a seqüência em que cada funcionalidade deve ser executada.

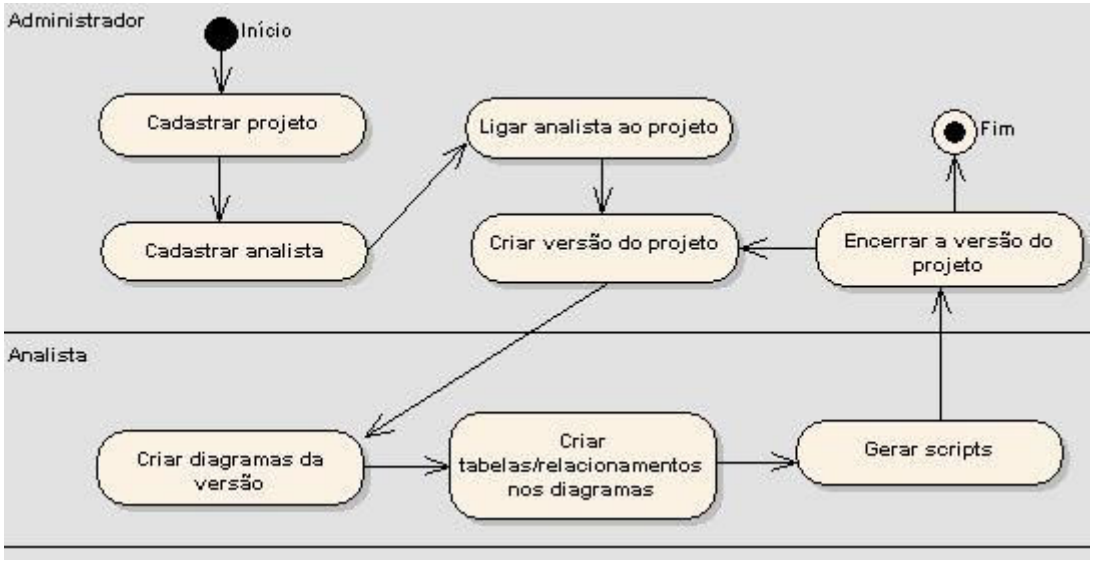


Figura 2. Fluxo de trabalho no WebModeler

A tela de cadastro de projetos é ilustrada na Figura 3.

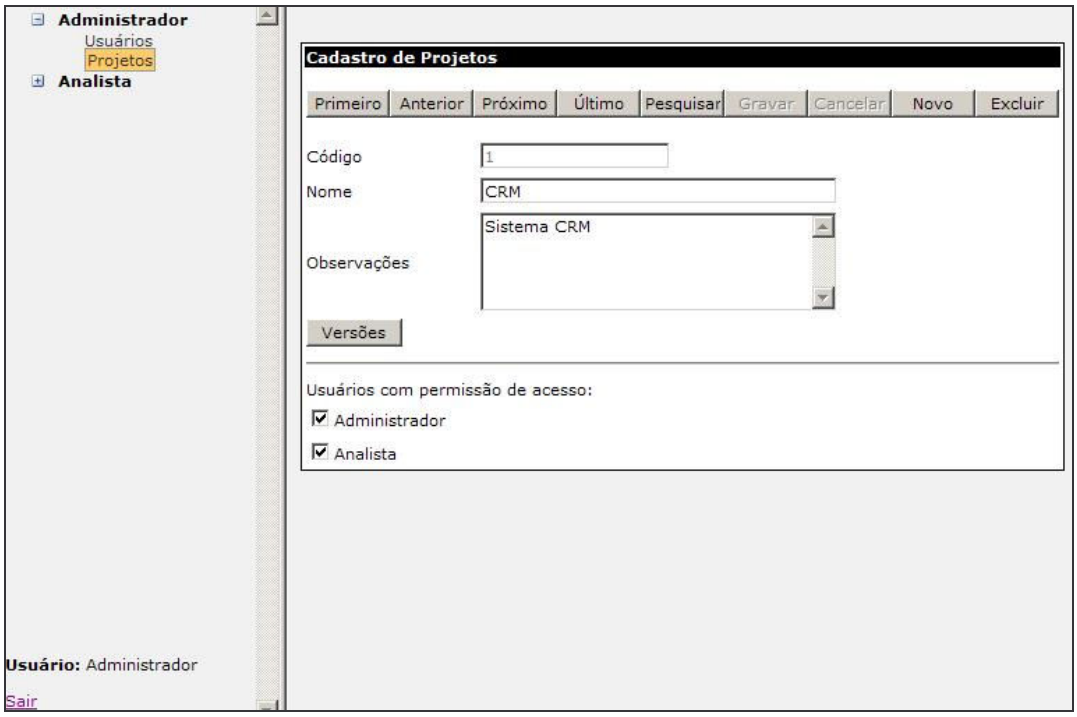


Figura 3. Tela de cadastro de projetos

Todas as outras telas de cadastro (versões dos projetos, usuários, diagramas, relacionamentos e tabelas) têm interface bastante semelhante à de cadastro de projetos e, por isso, nem todas serão exibidas neste artigo.

A Figura 4 demonstra a tela de cadastro de tabelas, sendo que na parte superior da mesma é informado o nome da tabela e na parte inferior são cadastradas as colunas.

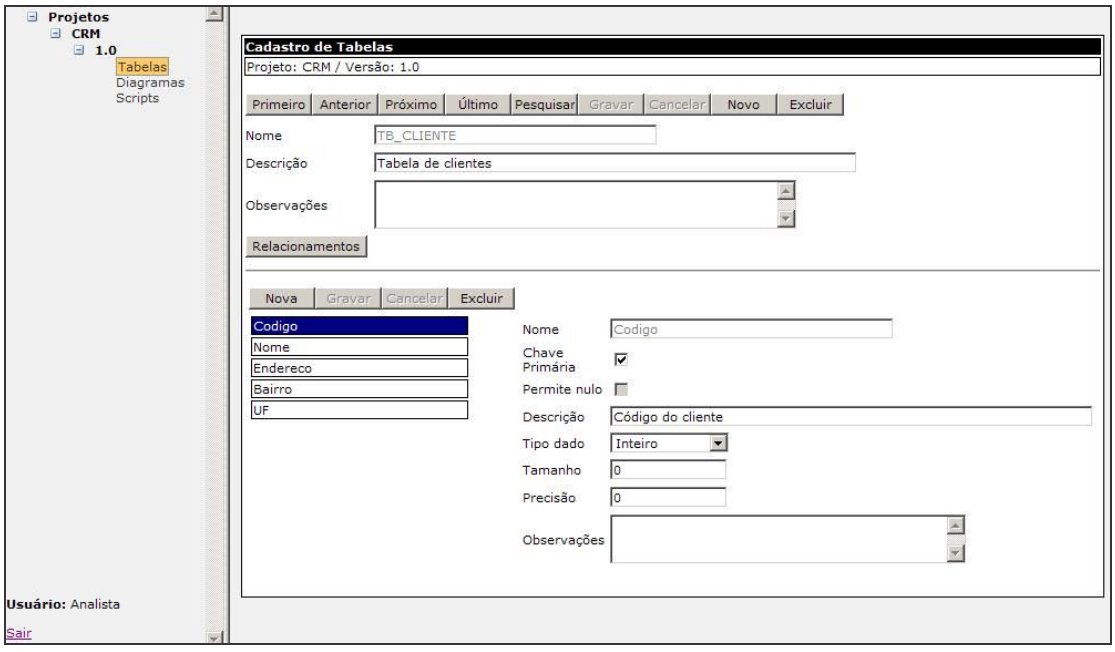


Figura 4. Tela de cadastro de tabelas

A Figura 5 exibe a tela de cadastro de relacionamentos entre as tabelas.

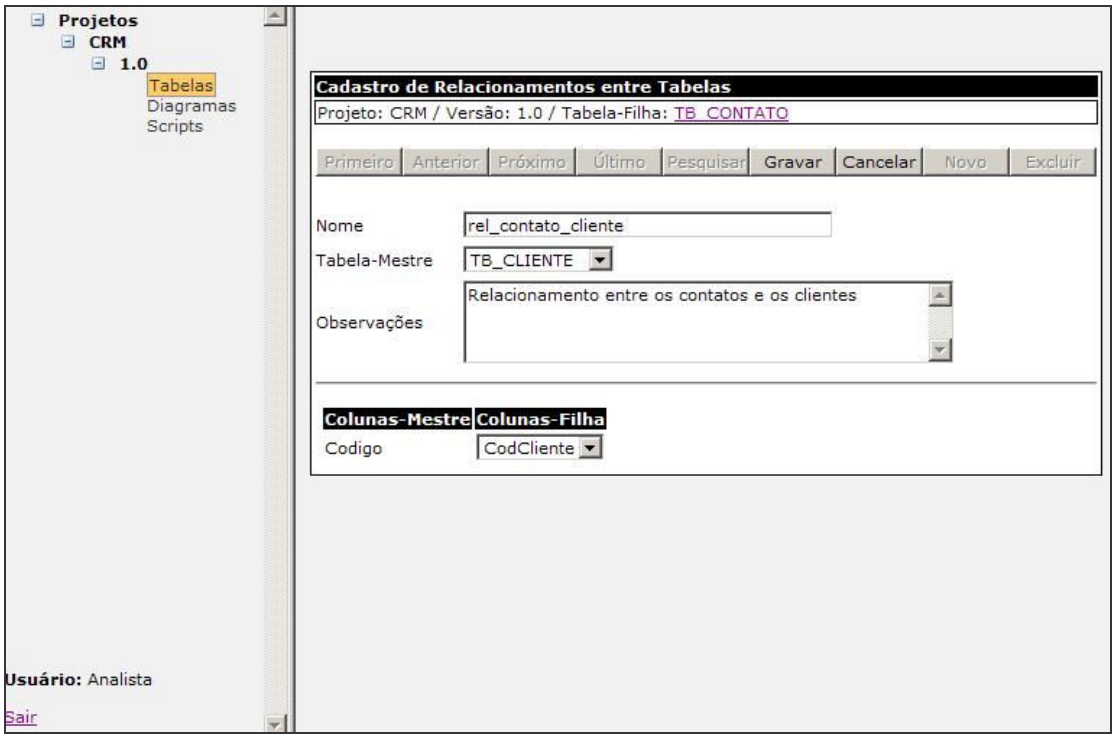


Figura 5. Tela de cadastro de relacionamentos entre tabelas

Já a Figura 6 mostra a tela de modelagem gráfica.

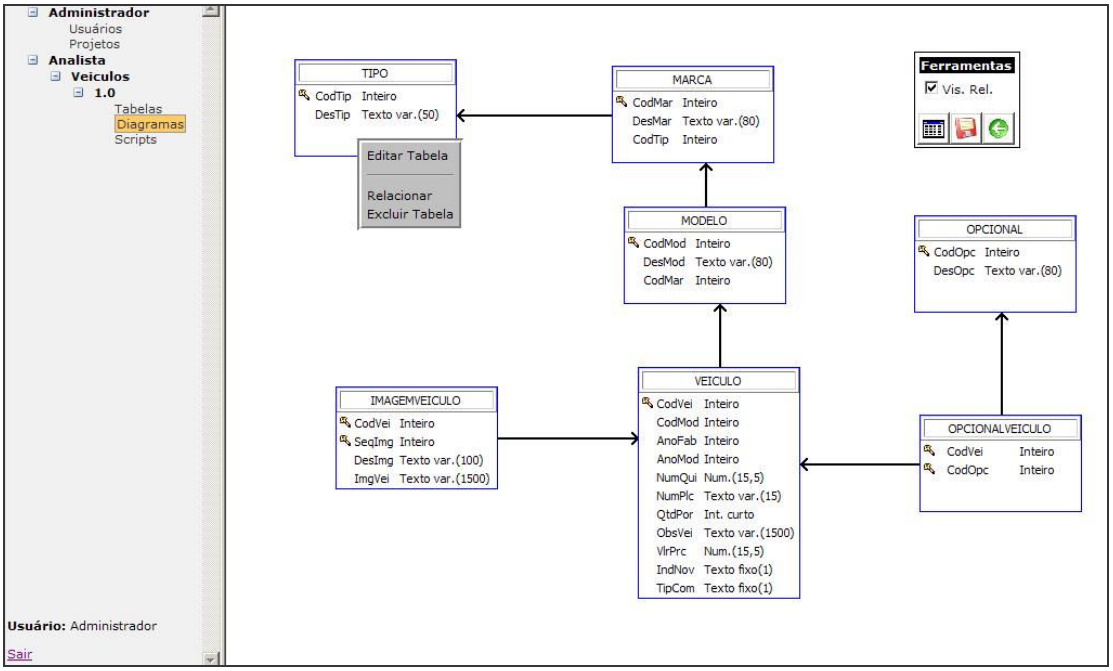


Figura 6. Tela de modelagem gráfica

Por fim, a Figura 7 ilustra a tela de geração de *scripts* e do relatório de diferenças entre versões de um projeto.

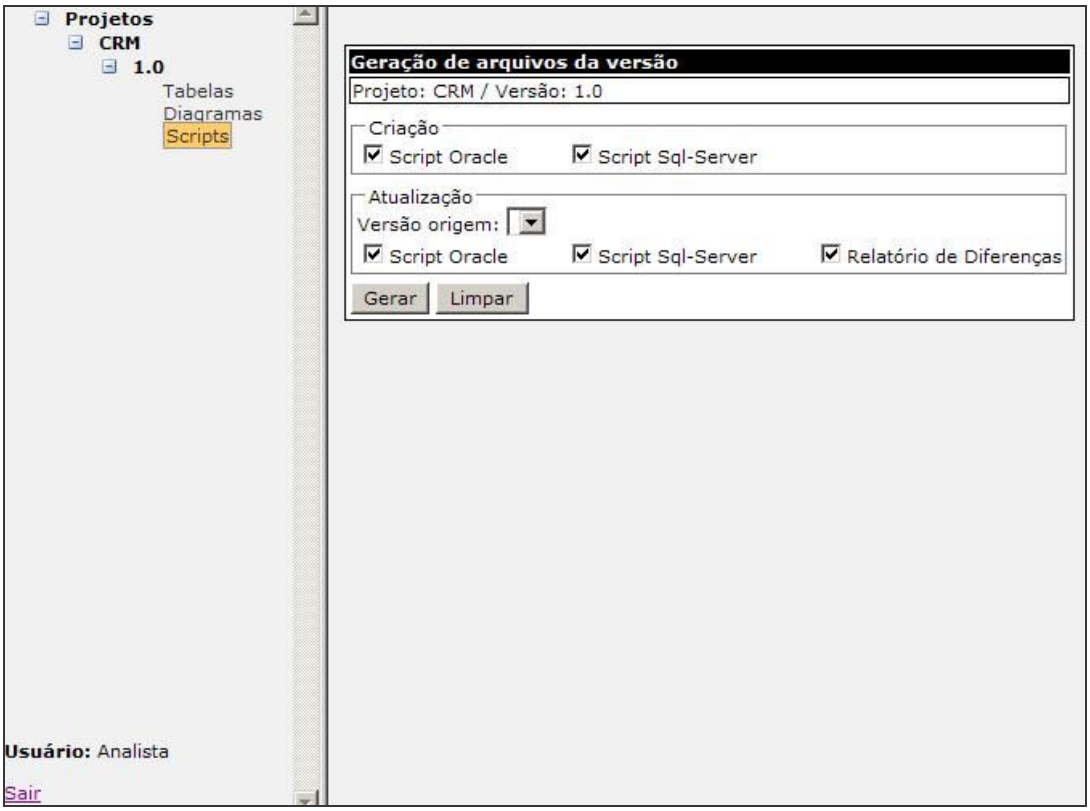


Figura 7. Tela de geração de *scripts*

4. Considerações finais

O software cujo desenvolvimento e operacionalidade foram apresentados neste artigo permite a modelagem de bancos de dados relacionais no ambiente web, possibilitando o cadastramento de projetos e suas versões, bem como das tabelas e relacionamentos existentes em cada versão. Essa modelagem pode ser feita inclusive de forma gráfica, como ocorre em diversas ferramentas CASE disponíveis no mercado, sendo que no software ora apresentado, as tabelas são modeladas definindo tipos de dados próprios do sistema, ao invés de tipos nativos do SGBD onde o banco será criado. Essa independência de algum SGBD específico pode ser considerada importante, principalmente porque diversas linguagens de programação modernas, como Delphi e Java, por exemplo, permitem a geração de programas cujo código-fonte é totalmente compatível com vários SGBDs relacionais, ou seja, não é necessária uma preocupação com o SGBD que será utilizado posteriormente e, com a ferramenta apresentada neste trabalho, também é possível criar o modelo do banco de dados sem esta preocupação. É possível ainda gerar *scripts* para a criação do banco de dados nos SGBD Oracle e MSSqlServer e, graças ao controle de versões, também é possível gerar *scripts* de atualização de bancos de dados.

Com relação às ferramentas CASE existentes no mercado, o software implementado tem algumas limitações, principalmente na questão da modelagem gráfica, pois não há recursos como impressão e *zoom* do diagrama, além de recursos para formatação das tabelas e relacionamentos. Muito da limitação da modelagem gráfica do WebModeler deve-se à atual falta de recursos dos *browsers* e do JavaScript em se tratando de criação de elementos gráficos (retas, figuras geométricas, etc.), sendo

que uma alternativa para amenizar estas limitações em versões futuras é a utilização da linguagem *Scalable Vector Graphics* (SVG), linguagem esta utilizada para criar recursos gráficos na web, mas sem suporte nativo no Internet Explorer. Em contrapartida às limitações, o WebModeler oferece vantagens como o acesso remoto (por tratar-se de um aplicativo web), controle de projetos e versões, cadastro de usuários e restrições de acesso por meio do relacionamento entre projetos e usuários.

Finalmente, com relação ao mercado para utilização da ferramenta ora apresentada, percebeu-se que além dos fins comerciais (empresas desenvolvedoras de software) a mesma pode ser utilizada para fins didáticos, em especial em disciplinas acadêmicas que visam prover conhecimentos básicos a cerca de bancos de dados relacionas, projeto de bancos de dados e modelagem de sistemas.

Referências

Asleson, R. e Schutta, N. T. (2006) "Fundamentos do Ajax", Rio de Janeiro: Alta Books.

Battisti, J. (2001) "SQL Server 2000: administração e desenvolvimento: curso completo", Rio de Janeiro: Acxel Books.

Bond, M. et al (2003) "Aprenda J2EE em 21 dias: com EJB, JSP, Servlets, JNDI, JDBC e XML", Tradução João Eduardo Nóbrega Tortell,. São Paulo: Pearson Education do Brasil.

Crane, D., Pascarello, E. e James, D. (2007) "Ajax em ação", Tradução Edson Furmankiewicz e Carlos Schafranski, São Paulo: Pearson Prentice Hall.

Gamma, E. et al. (2000), "Padrões de projeto: soluções reutilizáveis de software orientado a objetos", Tradução Luiz A. Meirelles Salgado, Porto Alegre: Bookman.

Oracle (2005) "Oracle Database SQL Reference 10g Release 2: Datatypes", Disponível em: <http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330/>. Acesso em: 24 maio 2007.

Sullivan, S. (2003) "Advanced DAO programming", Disponível em: <http://www-128.ibm.com/developerworks/library/j-dao/>. Acesso em: 30 abr. 2007.

Sun Microsystems (2002a) "Core J2EE Patterns: Front Controller", Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html>. Acesso em: 24 abr. 2007.

Sun Microsystems (2002b) "Front Controller", Disponível em: <http://java.sun.com/blueprints/patterns/FrontController.html>. Acesso em: 24 abr. 2007.

Utilizando Redes Neurais Artificiais no Controle de Robôs Móveis Aplicados ao Combate de Incêndios Florestais

Gustavo Pessin[1], Fernando Osório[1], Soraia Musse[2], Vinícius Nonnemmacher[3], Sandro Souza Ferreira[3]

[1]PIPCA – Universidade do Vale do Rio dos Sinos (UNISINOS)
Av. Unisinos 950 – São Leopoldo – RS – Brasil

[2]Faculdade de Informática – PUC-RS
Av. Ipiranga, 6681 – Porto Alegre – RS – Brasil

[3]GT JEDi – Universidade do Vale do Rio dos Sinos (UNISINOS)
Av. Unisinos 950 – São Leopoldo – RS – Brasil

fosorio@unisinos.br, soraia.musse@pucrs.br, {pessin, vnonnenmacher, sandro.s.ferreira}@gmail.com

Resumo. *O objetivo deste artigo é detalhar o projeto e o desenvolvimento de um sistema multi-agente que opera em um ambiente virtual de simulação realística[1]. Neste sistema, uma equipe heterogênea de agentes autônomos trabalha cooperativamente a fim de realizar com sucesso a identificação e o combate de incêndios florestais, sem intervenção humana. Os robôs de combate são fisicamente simulados e as informações sensoriais de cada robô (e.g. GPS, bússola, sonar) servem de entrada para uma Rede Neural que controla os atuadores do veículo para assim realizar navegação com desvio de obstáculos de um ponto aleatório do terreno até um ponto de atuação no combate ao incêndio. Os resultados das simulações demonstram que a Rede Neural controla satisfatoriamente os robôs móveis e que o sistema proposto pode vir a ter um papel muito importante no planejamento e execução de operações reais de combate a incêndios florestais.*

1. Introdução

Com a evolução das pesquisas em robótica, cada vez mais os robôs estão se tornando complexos em termos físicos. A grande variedade de estudos em morfologia robótica tem desenvolvido variações de robôs dotados de diversos meios de locomoção (*e.g.* pernas, rodas, esteiras). Em paralelo a este desenvolvimento temos a evolução constante de uma gama extremante grande de sensores (*e.g.* sistemas de visão, posicionamento, detecção de obstáculos). O desenvolvimento de algoritmos e técnicas para coordenar estes conjuntos físicos em um ambiente dinâmico é um desafio extremante complexo [Go et al. 2004]. Dotar robôs autônomos de capacidade de raciocínio inteligente e de interação com o meio em que estão inseridos é uma área de pesquisa que tem atraído a atenção de um grande número de pesquisadores [Dudek and Jenkin 2000]. Existem diversas áreas onde a habilidade de um único agente autônomo não é suficiente ou

[1] Código-fonte e vídeos das simulações disponíveis em http://pessin.googlepages.com

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 19-30.

eficiente para a realização de uma tarefa, em alguns destes casos, como patrulhamento, vigilância, resgate ou exploração o mais indicado é a aplicação de sistemas multi-robóticos. Sistemas multi-robóticos são sistemas onde robôs autônomos trabalham cooperativamente a fim de cumprir uma missão, podendo existir interação entre os robôs ou não [Osagie 2006].

Um grande sonho de nossa sociedade é a aplicação de sistemas robóticos substituindo atividades que coloquem em risco a vida humana, em atividades onde a atuação de humanos é deficitária ou onde a atuação humana de certa forma é ineficiente. A tarefa de monitoração e combate de incêndios em áreas florestais é um caso onde a aplicação de um sistema multi-robótico poderia diminuir consideravelmente os prejuízos humanos, materiais e ambientais. Com relação a incêndios florestais, anualmente registram-se cerca de 45.000 incêndios nas florestas da Europa. Entre 1989 e 1993, só na zona mediterrânea 2,6 milhões de hectares florestais foram destruídos pelo fogo, o equivalente ao desaparecimento do mapa de um território com a dimensão da Bélgica em cinco anos [CE 2006]. Os incêndios florestais causam diversos tipos de danos humanos, materiais e ambientais. Danos ambientais na fauna e flora afetam desde o solo até o aspecto de sobrevivência de áreas não incendiadas, prejudicando os ambientes naturais e o planejamento florestal. As perdas humanas são mais dramáticas, onde este custo dificilmente pode ser quantificado. Quanto a prejuízos humanos, por exemplo, na Austrália, em 1983 um incêndio que atingiu cerca de 400.000ha matou 75 pessoas. No Canadá e EUA, incêndios entre 1969 e 1994 mataram cerca de 52 pessoas. No Brasil, um incêndio no Paraná, em 1973 provocou 110 mortes [LIF 2006]. A extensão territorial do Brasil e a diversidade de sua cobertura vegetal, bem como o número expressivo de ocorrências de incêndios florestais verificados no país, são fatores que enfatizam a necessidade de um sistema cada vez mais aprimorado e consistente de detecção e combate de incêndios florestais [Batista 2004]. Diversas iniciativas a fim de incrementar a capacidade de reação de órgãos públicos e civis no sentido de evitar desastres tem sido uma das preocupações junto a órgãos como a Secretaria Nacional de Defesa Civil, levando a criação de novos CEPEDs (Centro de Estudos para a Prevenção de Emergências e Desastres). Importantes iniciativas, como a RBV – Rede Brasileira de Visualização, financiada pela FINEP, também tem sido incentivadas, onde a competência de Segurança e Defesa (Civil e Militar) da Rede vem sendo foco de desenvolvimentos junto a nossa Universidade e na qual este projeto se integra.

Nossas principais metas neste projeto são: (*i*) Recolher informações sobre dados florestais, tipos de vegetação, topografia, e comportamento de incêndios para criar o ambiente virtual mais realista possível; (*ii*) Simular incêndios em florestas, reproduzindo de forma bastante realista o ambiente e a propagação dos focos de incêndio; (*iii*) Pesquisar ferramentas e técnicas de combate à incêndios florestais utilizadas por bombeiros; (*iv*) Implementar a simulação de agentes móveis autônomos colaborativos capazes de formar uma brigada de combate a incêndios; (*v*) Estudar métodos de aprendizado de máquina e suas vantagens para o modelo; e (*vi*) Estudar a robustez das ações dos agentes pela leitura de dados de sensores sujeitos a erros.

Neste artigo apresentamos na Seção 2 características importantes de robótica móvel que tenham relação com as necessidades deste trabalho. Na Seção 3 descrevemos brevemente conceitos de simulação e modelagem, as ferramentas pesquisadas e as bibliotecas de simulação selecionadas para o desenvolvimento do trabalho. Na Seção 4

descrevemos técnicas e operações reais de identificação e combate a incêndios florestais. Na Seção 5 descrevemos o ambiente desenvolvido, a operação multi-agente de identificação e combate, o desenvolvimento e a aplicação da Rede Neural no controle dos robôs móveis. Finalizamos apresentando a conclusão do trabalho realizado.

2. Robótica Móvel

Um robô móvel é um dispositivo mecânico montado sobre uma base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o ambiente [Marchi 2001, Bekey 2005]. A interação com o ambiente se dá através de ciclos de percepção-ação que consistem em três passos fundamentais: (*i*) Obtenção de informação através de sensores; (*ii*) Processamento das informações para seleção de ação; e, (*iii*) Execução da ação através do acionamento dos atuadores.

Esse conjunto de operações, em uma análise superficial, pode parecer simples, porém o controle robusto de sistemas robóticos tem complicações físicas e mecânicas (como cinemática e dinâmica), eletrônicas (como falta de precisão de sensores) e computacionais que tornam a criação de um conjunto de regras uma tarefa árdua e sujeita a erros. Sensores são os mecanismos de percepção de um robô e realizam medições físicas (*e.g.* contato, distância, orientação, temperatura), provêm sinais ou dados crus que precisam ser interpretados pelo "cérebro" do robô. A interpretação destes sinais deve ser a única maneira de um robô autônomo entender o ambiente que o cerca para poder realizar as mudanças de ação necessárias [JPL/NASA 2007]. Atuadores são os mecanismos de ação de um robô (*e.g.* motores, pistões, braços manipuladores), controlados por circuitos eletrônicos que recebem valores de ação, valores estes que devem ser calculados pelo "cérebro" do robô e devem estar de acordo com especificações de fabricantes [JPL/NASA 2007].

3. Simulação e Modelagem

Experimentos em robótica móvel podem ser realizados de duas formas: diretos em um robô real ou em um robô simulado em um ambiente virtual realista [Pfeifer e Scheier 1999]. Usualmente, experimentos em robótica móvel utilizando um robô real exigem enorme despendimento de tempo e de recursos financeiros. Para que seja possível a implementação física real, o sistema multi-agente que propomos deve ser projetado, desenvolvido e testado anteriormente em ambientes de simulação realísticos. A simulação de sistemas robóticos é especialmente necessária para robôs caros, grandes, ou frágeis [Go et al. 2004]. É uma ferramenta extremamente poderosa para agilizar o ciclo de desenvolvimento de sistemas de controle robóticos eliminando desperdício de recursos, tanto financeiros como computacionais. Para que uma simulação seja útil, entretanto, ele deve capturar características importantes do mundo físico, onde o termo *importantes* tem relação ao problema em questão [Go et al. 2004]. No caso deste trabalho, é fundamental que existam restrições físicas no modelo e que exista a possibilidade de trabalho em um terreno irregular, provido de obstáculos. Para o desenvolvimento deste trabalho foram pesquisadas algumas ferramentas de simulação, porém, nenhuma mostrou possuir o conjunto completo das características requisitadas. Uma pequena descrição de cada uma das ferramentas pesquisadas é fornecida a seguir.

O *Microsoft Robotics Studio* (msdn.microsoft.com/robotics) tem como objetivo prover um ambiente para desenvolvimento de simulação robótica. Permite a simulação em terrenos irregulares e é livre apenas para uso não comercial. O desenvolvimento dos sensores, dos controles inteligentes e da comunicação entre os robôs depende de programação (*e.g.* C#, VB.NET). O *Webots* (www.cyberbotics.com) é um simulador de robôs móveis que tem como base a biblioteca de simulação física ODE, inclui modelos prontos de alguns robôs comerciais e modela sensores com a capacidade de detecção de obstáculos, visão, e manipuladores simples. O usuário pode programar cada robô utilizando *C/C++*. É um produto comercial. Outros ambientes de simulação estudados foram o *Khepera Simulator* (diwww.epfl.ch/lami/team/michel/khep-sim), o *Mission Simulation Facility* (ase.arc.nasa.gov/msf), o *JUICE* (www.natew.com/juice) e o *Simulator BOB* (www.tu-harburg.de/ti6/mitarbeiter/ pst/Sim/Simulator.html).

Um dos requisitos básicos para nosso ambiente de simulação é que todas as bibliotecas de programação sejam software livre e em linguagem *C/C++*. As bibliotecas utilizadas no desenvolvimento do sistema são: (*i*) ***Open Dynamics Engine:*** (www.ode.org) é uma biblioteca desenvolvida para a simulação física de corpos rígidos articulados [Smith 2006]. É utilizada para o desenvolvimento dos robôs e do mundo colisivo. Uma estrutura articulada é criada quando corpos rígidos são conectados por algum tipo de articulação, como, por exemplo, um veículo terrestre que tem a conexão de rodas em um chassi. A ODE não tem como objetivos realizar simulação de outras dinâmicas além da dinâmica de corpos rígidos (*e.g.* ondas, fluídos, corpos flexíveis, fraturas). O sistema de detecção de colisão é nativo e suporta diversas primitivas (*e.g.* esfera, caixa, cilindro, plano, raio). A utilização da ODE no nosso ambiente é fundamental por fornecer restrições físicas, principalmente na definição da morfologia dos robôs; (*ii*) ***Open Scene Graph:*** (www.openscenegraph.org) é uma biblioteca para desenvolvimento de aplicações gráficas *3D* de alta performance. Baseada no conceito de grafos de cena, provê ao desenvolvedor um ambiente orientado a objeto sobre a *OpenGL* (www.opengl.org), liberando este da necessidade de implementação e otimização de chamadas gráficas de baixo nível. Um grafo de cena permite a representação de objetos em uma cena com uma estrutura que permite a criação de grupos que podem compartilhar propriedades, assim, podemos definir uma propriedade comum em um nível hierárquico mais alto e todos os objetos inferiores receberão esta propriedade [Barros et al. 2007]; (*iii*) ***DrawStuff:*** como a ODE é completamente independente de visualizador, iniciar a criação dos corpos e das simulações pode ser uma tarefa bastante árdua caso não tenhamos uma forma simples e fácil de visualizar os objetos. Por este motivo, a biblioteca *DrawStuff* é disponibilizada em conjunto com a ODE. Basicamente, o *DrawStuff* é um ambiente de visualização de objetos *3D* que tem o propósito de permitir a demonstração visual da ODE sendo uma biblioteca bastante simples e rápida para utilização; (*iv*) ***Demeter:*** (www.tbgsoftware.com) é uma biblioteca desenvolvida para renderizar terrenos *3D*, desenvolvida para ter rápida performance e boa qualidade visual, pode renderizar grandes terrenos em tempo-real sem necessidade de hardware especial, depende da *Simple DirectMedia Layer* (www.libsdl.org) para realizar o tratamento das texturas do terreno e da *Geospatial Data Abstraction Library* (www.gdal.org) para carregar arquivos de elevação.

4. Técnicas Reais de Identificação e Combate de Incêndios Florestais

A fim de melhor entender como proceder no combate a incêndios florestais, e assim planejar as estratégias a serem implementadas nos agentes autônomos, foi realizado um estudo sobre as técnicas reais de operação. Este estudo teve como base os trabalhos de [Antunes 2000, Batista 2004, CPTEC/INPE 2006, LIF 2006, Remel and Pereira 2001]. A operação de combate ou supressão de um incêndio envolve seis etapas distintas [LIF 2006]: (*i*) *Detecção*: tempo decorrido entre a ignição ou início do fogo e o momento que ele é visto por alguém, alguns métodos são o uso das torres de vigilância, patrulhamento terrestre, patrulhamento por avião ou imagens de satélites; (*ii*) *Comunicação*: tempo compreendido entre a detecção do fogo e o recebimento da informação por um responsável; (*iii*) *Mobilização*: tempo gasto entre o recebimento da informação da existência do fogo e a saída do pessoal para combate; (*iv*) *Deslocamento*: tempo compreendido entre a saída do pessoal de combate e a chegada da primeira turma ao local do incêndio; (*v*) *Planejamento*: tempo gasto pelo responsável pelo combate (QG) para avaliar o comportamento do fogo e planejar a estratégia de combate; e (*vi*) *Combate*: tempo consumido na operação de combate ou eliminação do incêndio. Existem quatro métodos de combate ao fogo nos incêndios florestais [LIF 2006]: (*i*) *Método direto*: usado quando a intensidade do fogo permite uma aproximação suficiente da brigada à linha de fogo, são usadas as seguintes técnicas e materiais: água (bombas costais, baldes ou moto-bombas); terra (pás); ou batidas (abafadores); (*ii*) *Método paralelo ou intermediário*: usado quando não é possível o método direto e a intensidade do fogo não é muito grande, consiste em limpar, com ferramentas manuais, uma estreita faixa, próxima ao fogo, para deter o seu avanço e possibilitar o ataque direto; (*iii*) *Método indireto*: usado em incêndios de intensidade muito grande, consiste em abrir aceiros com equipamento pesado (*e.g.* trator, motoniveladeira), utilizando ainda um contra-fogo, para ampliar a faixa limpa e deter o fogo, antes que chegue ao aceiro; e (*iv*) *Método aéreo*: usado nos incêndios de copa, de grande intensidade e área e em locais de difícil acesso às brigadas de incêndio, são usados aviões e helicópteros, especialmente construídos ou adaptados para o combate ao incêndio.

A rapidez e a eficiência na detecção e monitoramento dos incêndios florestais são fundamentais para a viabilização do controle do fogo, redução dos custos nas operações de combate e atenuação dos danos. Para países de grande extensão territorial, como o Brasil, o monitoramento dos incêndios florestais através de imagens de satélites é o meio mais eficiente e de baixo custo quando comparado com os demais meios de detecção [Batista 2004]. O lançamento em 1972 do primeiro satélite *Landsat* possibilitou detectar alterações nas áreas florestais do espaço. Desde então, as imagens termais e de infravermelho têm sido usadas na detecção de incêndios e estudos de mapeamento, permitindo que áreas queimadas e não queimadas sejam detectadas através do contraste entre os gradientes térmicos [Remel e Pereira 2001].

5. Ambiente de Simulação

O protótipo do sistema foi implementado em *C++*, a biblioteca OSG é responsável pela saída gráfica do protótipo, a biblioteca *Demeter* é responsável pelo terreno irregular e a biblioteca ODE é responsável pelo realismo físico, tanto da morfologia robótica como da colisão entre os objetos presentes no ambiente (*e.g.* robôs, árvores, inclinação de terreno). A base do realismo de interação é o uso integrado das bibliotecas de

programação ODE, OSG e *Demeter*. Este conjunto permite que os robôs simulados fisicamente respeitem questões como gravidade, inércia e atrito. Por exemplo, mantendo uma força *f* constante nos motores lineares (torque) um veículo terá velocidade *v* em regiões planas, em regiões de declive terá *v* maior e em aclives terá *v* menor.

A implementação do protótipo (Figura 1) iniciou com a criação de um mapa que simula a integração das informações de vegetação, topografia e comportamento de fogo baseado em [Antunes 2000, Batista 2004, CE 2006, CPTEC/INPE 2006, Koproski 2005, LIF 2006, Remmel e Perera 2001]. O estudo dos modelos de florestas e resíduos florestais é de grande importância para o aprimoramento dos modelos de simulação a serem implementados em ambientes virtuais [Pessin et al. 2007]. A criação dos mapas teve como base cartas topográficas e o mapa de modelos de combustíveis florestais do Ministério da Agricultura do Brasil.

(a) (b)

Figura 1. Ambiente completo (a) Um veículo preso em uma árvore e (b) Vista do grupo em deslocamento até o incêndio.

Para a simulação da vegetação e correta propagação do incêndio, existe uma matriz oculta sob o terreno. Esta matriz possui, para cada área do terreno, o tipo de vegetação presente, assim, considerando orientação do vento, intensidade do vento e tipo de vegetação de uma área podemos construir a simulação de propagação do fogo. O fogo pode ser iniciado em uma posição aleatória ou parametrizado em uma posição inicial fixa. A velocidade de propagação respeita dados do modelo retirados de [Koproski 2005], considerando intensidade do vento e sua orientação. Quanto ao vento, tanto a sua intensidade como a sua orientação podem ser geradas aleatoriamente ou configurados a partir de dados parametrizados pelo usuário. O tempo de permanência do fogo em uma área é relacionado diretamente ao tipo da vegetação presente e se comporta baseado nos valores de tipo de vegetação, inclinação do terreno, intensidade e orientação do vento. Desta forma a propagação do fogo busca simular de modo bastante realístico a forma como o fogo se propagaria em um ambiente real. A simulação da comunicação é feita utilizando um sistema baseado em Quadro-Negro (*blackboard*) [Rezende 2003]. O sistema de comunicação simula a troca de mensagens que vão do agente monitor para o agente líder, do agente líder aos agentes de combate, e dos agentes de combate ao agente líder. Existem casos de comunicação um para um e de um para todos. A fila usada como função de Quadro-Negro armazena as seguintes informações: indicador de remetente, indicador de destinatário, e tipo da mensagem (*e.g.* aviso de incêndio, aviso de fim de incêndio, negociação de times, posição do incêndio). No ambiente desenvolvido, cada árvore existe como um modelo OSG e como um cilindro ODE, assim, o veículo da Figura 1(a) está, na verdade, colidindo com um

cilindro (não visualizado). O ambiente permite que, através de parâmetros, possamos habilitar ou não a exibição dos objetos físicos ao invés de apenas a sua representação gráfica, neste exemplo, uma árvore.

5.1. Operação de Identificação e Combate de Incêndio

O ambiente conta com um grupo de *n* robôs de comportamento reativo (controlados por uma RNA explicada na próxima Seção) e com um agente deliberativo (mecanismo de planejamento). Em se considerando arquiteturas de sistemas multi-agente, o ambiente desenvolvido possui controle centralizado de planejamento e controle distribuído de ações (local em cada robô móvel). Usamos o ambiente para simular a seguinte operação: um agente monitor (satélite) monitora o terreno da área florestal, ao identificar uma área com foco de incêndio envia uma mensagem para o agente líder. Esta mensagem contém a posição (*x,y*) do incêndio (simulando uma posição UTM) e a densidade da vegetação na área. O agente líder é o agente responsável pela definição das posições de atuação dos robôs bombeiros no combate ao incêndio. Após receber o *aviso de incêndio* do agente monitor, o agente líder envia para todos os agentes de combate uma mensagem informando *início de incêndio na posição (x,y)* e recebe a distância vetorial *d* (Equação 1) de cada agente de combate ao incêndio.

$$d(P,Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

Após o agente líder receber as respostas dos agentes de combate, envia mensagens de solicitação de formação para atuação da seguinte forma: se não existe vento, solicita os 8 agentes mais próximos ao incêndio em formação circular eqüidistante com um raio predeterminado; se existe vento, solicita os 4 agentes mais próximos ao incêndio em formação semi-circular (ferradura) no sentido contrário ao do vento, com raio definido de acordo com a intensidade do vento. Esta formação é baseada em regras e pode ser visualizada nas Figuras 2 e 3. A regra pré-programada que define a posição final de cada robô na composição da formação da equipe (quando circular eqüidistante) pode ser vista nas Equações 2 e 3.

$$x_f = x_a + r \times \cos\left(i \times \frac{360}{q} \times \frac{\pi}{180} \right) \quad (2)$$

$$y_f = y_a + r \times \text{sen}\left(i \times \frac{360}{q} \times \frac{\pi}{180} \right) \quad (3)$$

Considere x_f e y_f como as coordenadas da posição final do agente, x_a e y_a como as coordenadas da posição central do incêndio, *r* como o raio de atuação, *q* a quantidade total de agentes e *i* o índice do agente. As posições são negociadas e confirmadas com a comunicação entre os agentes da equipe. Os agentes do time possuem quatro sensores de temperatura que servem como alerta, quando a temperatura de um deles excede o máximo especificado, o agente se desloca no sentido do sensor com a menor temperatura e solicita ao agente líder a atualização da formação do time. O comportamento dos agentes de combate é reativo, deslocando-se em direção a posição de seu objetivo específico desviando de obstáculos. O método de combate de incêndio simulado é o método indireto. Os agentes de combate simulados são motoniveladoras que tem como finalidade cercar o foco de incêndio e criar um aceiro (área livre de

vegetação onde o fogo se extingue pela falta de combustível). Esta operação pode ser entendida com as Figuras 2 e 3. Quanto ao controle de posicionamento, um sensor do tipo GPS é simulado em cada robô. Em experimentos que realizamos com um GPS *Garmin Etrex* (www.garmin.com) obtivemos um erro médio de 18,6 metros. Considerando que cada agente possui seu próprio GPS o tratamento deste erro é crucial na criação dos aceiros. Tratamos esta informação de duas maneiras, a primeira faz com que o erro médio deste sensor durante o deslocamento seja usado somado a distância de criação do aceiro e também é somado ao final da área de criação, como mostra a Figura 2(b).

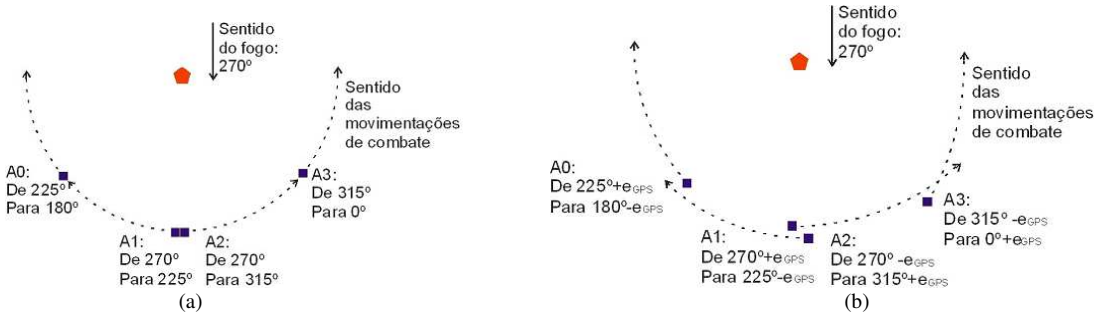
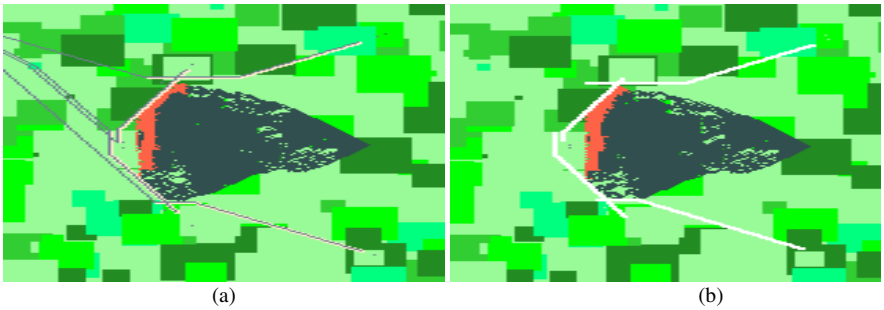


Figura 2. Exemplo de formação de combate quando o sentido do vento é de 270º e existem 4 agentes de combate: (a) Não considera erro no GPS e (b) Considerando erro no GPS, apresentando limites propositalmente redundantes

A Figura 2(a) apresenta as posições e trajetórias com "posicionamento perfeito", o que não é possível de se obter em uma situação real. Devido ao erro de posicionamento do GPS não é possível se estabelecer uma rota que se encaixe perfeitamente como a apresentada nesta Figura. Deste modo a preparação do aceiro (semi-círculo indicado na figura) não será executada de forma correta. A Figura 2(b) apresenta as correções adicionadas ao algoritmo de modo a obter uma trajetória que permita criar um aceiro incluindo o erro do GPS (e_{GPS}) no modelo de deslocamento dos agentes. As Figuras 3(a) e 3(b) demonstram os resultados obtidos em uma das simulações realizadas com o protótipo *2D*.



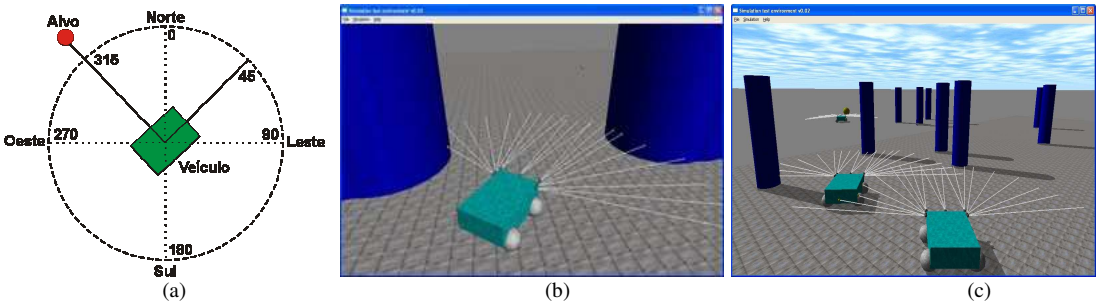
(a) (b)

Figura 3. Simulação *2D* considerando o erro de GPS: (a) Trajetórias dos robôs-bombeiros e (b) Criação do aceiro. Ambas apresentam limites propositalmente redundantes

5.2. Morfologia dos Robôs Móveis

Os robôs móveis foram desenvolvidos com a biblioteca de simulação de corpos rígidos artículos ODE. A Figura 4(b) apresenta o veículo desenvolvido. Dada a existência de

restrições físicas, a única maneira de controlar o veículo das Figuras 4(b) e 4(c) é com a aplicação de forças em seus dois motores simulados, que são: um motor angular (para o giro do volante) e um motor linear (para o torque). Além do GPS e termômetro descritos na Seção anterior, cada robô possui também uma bússola (Figura 4(a)), responsável pela obtenção da orientação do veículo e, em conjunto com o GPS, é responsável pela obtenção do azimute (orientação em que deve se deslocar para atingir o alvo). Os sensores de distância são sonares simulados, apresentando as características de capacidade de medir distâncias entre 15cm e 11m, como o Polaroid 6500 (www.senscomp.com). Nas Figuras 4(b) e 4(c), cada três linhas brancas representam um sensor.



(a) (b) (c)

Figura 4. Sensores desenvolvidos: (a) A bússola é responsável pela orientação, e, junto com o GPS, permite obter o azimute para o alvo, (b) e (c) Sensores de distância.

5.3. Rede Neural Artificial

Quando um robô recebe a coordenada da posição do incêndio do agente líder, ele deve se deslocar de sua posição atual (aleatória) até a posição de atuação. Para realizar esta navegação, desviando de obstáculos, o agente usa uma Rede Neural Artificial (RNA) do tipo *backpropagation*. As entradas da RNA são obtidas pelos sensores presentes no robô e as saídas são as forças a serem aplicadas nos atuadores. A RNA possui 9 entradas: orientação do veículo e orientação para o alvo (obtidos através de bússola e GPS simulados) mais as proximidades de obstáculos medidas por 7 sensores simulados do tipo sonar. As saídas são: força a aplicar no motor linear (torque, de 0.0 a 3.0) e força a aplicar no motor angular (giro da barra da direção, de –0.8 a 0.8). A base de dados para treinamento da RNA foi obtida a partir de um sistema de regras simples, rodamos 15 simulações com diferentes pontos de início e destino para obter dados específicos de navegação e 12 simulações para obter dados de desvios de obstáculos, obtendo um total de 7.720 registros. Esta base de dados foi dividida em 70% para treino e 30% para teste, e, usando o SNNS (www-ra.informatik.uni-tuebingen.de/SNNS/) foram geradas seis Redes Neurais Artificiais com 4, 9, 18, 24, 30 e 36 neurônios na camada oculta. Foi realizado o treino de cada RNA e análise de 500, 1.000, 1.500 e 2.000 ciclos. A Tabela 1 apresenta os dados de treino e teste das Redes Neurais desenvolvidas, os valores apresentados são o Erro Médio Absoluto (*Mean Absolute Error* - MAE) (Equação 4).

$$\left|E_{MAE}\right| = \frac{1}{N}\sum_{i=1}^{N}\left|x_i - \bar{x}_i\right| \quad (4)$$

Tabela 1. Erro Médio Absoluto das RNAs.

		Exp. 1				Exp. 2				Exp. 3			
		Treino		Teste		Treino		Teste		Treino		Teste	
RNA	Ciclos	Giro	Veloc.	Giro	Veloc.	Giro	Veloc.	Giro	Veloc.	Giro	Veloc.	Giro	Veloc.
9x4x1	500	0,228	0,479	0,223	0,478	0,220	0,450	0,222	0,445	0,230	0,472	0,227	0,472
9x4x1	1000	0,170	0,478	0,174	0,477	0,158	0,401	0,161	0,404	0,153	0,437	0,157	0,438
9x4x1	1500	0,136	0,424	0,139	0,424	0,150	0,388	0,153	0,390	0,142	0,411	0,147	0,412
9x4x1	2000	0,138	0,412	0,143	0,412	0,149	0,386	0,152	0,389	0,143	0,408	0,149	0,409
9x9x1	500	0,235	0,465	0,229	0,462	0,135	0,454	0,139	0,453	0,141	0,426	0,146	0,428
9x9x1	1000	0,144	0,413	0,145	0,413	0,131	0,406	0,135	0,408	0,128	0,376	0,134	0,379
9x9x1	1500	0,131	0,402	0,136	0,401	0,130	0,395	0,135	0,398	0,125	0,361	0,131	0,366
9x9x1	2000	0,128	0,397	0,134	0,396	0,129	0,391	0,133	0,394	0,125	0,349	0,132	0,354
9x18x1	500	0,231	0,468	0,223	0,463	0,290	0,476	0,275	0,473	0,183	0,475	0,186	0,471
9x18x1	1000	0,133	0,403	0,138	0,405	0,131	0,408	0,136	0,409	0,137	0,416	0,141	0,417
9x18x1	1500	0,134	0,396	0,139	0,399	0,125	0,344	0,130	0,346	0,134	0,392	0,139	0,396
9x18x1	2000	0,137	0,391	0,142	0,393	0,112	0,311	0,117	0,316	0,135	0,382	0,141	0,387
9x24x1	500	0,190	0,412	0,191	0,410	0,209	0,474	0,203	0,473	0,172	0,446	0,171	0,444
9x24x1	1000	0,142	0,314	0,135	0,318	0,177	0,392	0,176	0,392	0,130	0,340	0,133	0,341
9x24x1	1500	0,127	0,285	0,128	0,292	0,140	0,220	0,140	0,229	0,132	0,289	0,136	0,294
9x24x1	2000	0,096	0,261	0,100	0,269	0,130	0,205	0,132	0,215	**0,112**	**0,183**	**0,116**	**0,194**
9x30x1	500	0,145	0,383	0,150	0,384	0,159	0,382	0,157	0,383	0,188	0,309	0,187	0,317
9x30x1	1000	0,135	0,355	0,139	0,358	0,142	0,340	0,145	0,346	0,127	0,294	0,131	0,301
9x30x1	1500	0,124	0,338	0,128	0,341	0,130	0,317	0,135	0,325	0,122	0,288	0,126	0,296
9x30x1	2000	0,116	0,319	0,121	0,323	0,126	0,304	0,132	0,311	0,115	0,283	0,119	0,291
9x36x1	500	0,142	0,452	0,144	0,445	0,229	0,471	0,221	0,464	0,246	0,470	0,232	0,462
9x36x1	1000	0,139	0,400	0,142	0,401	0,136	0,402	0,138	0,401	0,172	0,452	0,147	0,446
9x36x1	1500	0,137	0,374	0,140	0,374	0,123	0,373	0,126	0,374	0,141	0,397	0,143	0,396
9x36x1	2000	0,133	0,360	0,136	0,359	0,114	0,342	0,118	0,346	0,137	0,384	0,141	0,385

A RNA que apresentou o menor MAE tanto no giro como na velocidade foi a com 24 neurônios na camada oculta, com 2.000 ciclos de treino da terceira semente aleatória. Esta RNA foi convertida em um programa C e utilizada nos robôs.

5.4. Aplicação da Rede Neural Artificial

Após a implementação da RNA no controle de navegação com desvio de obstáculos nos robôs móveis, realizamos uma série de simulações com o ambiente contendo 3 robôs, com pontos de início e destino aleatórios e com uma quantidade parametrizada de cilindros. Os resultados podem ser visto nas Tabelas 2 e 3. Para tornar as simulações mais rápidas, trabalhamos em um terreno pequeno, representado 80*m* x 100*m*, cada veículo tem dimensões de 2,5*m* x 3*m* e cada cilindro tem diâmetro de 1*m*. A Figura 5 apresenta imagens de seqüências de uma simulação onde podemos ver o resultado correto de navegação com desvio e o posicionamento final dos robôs.

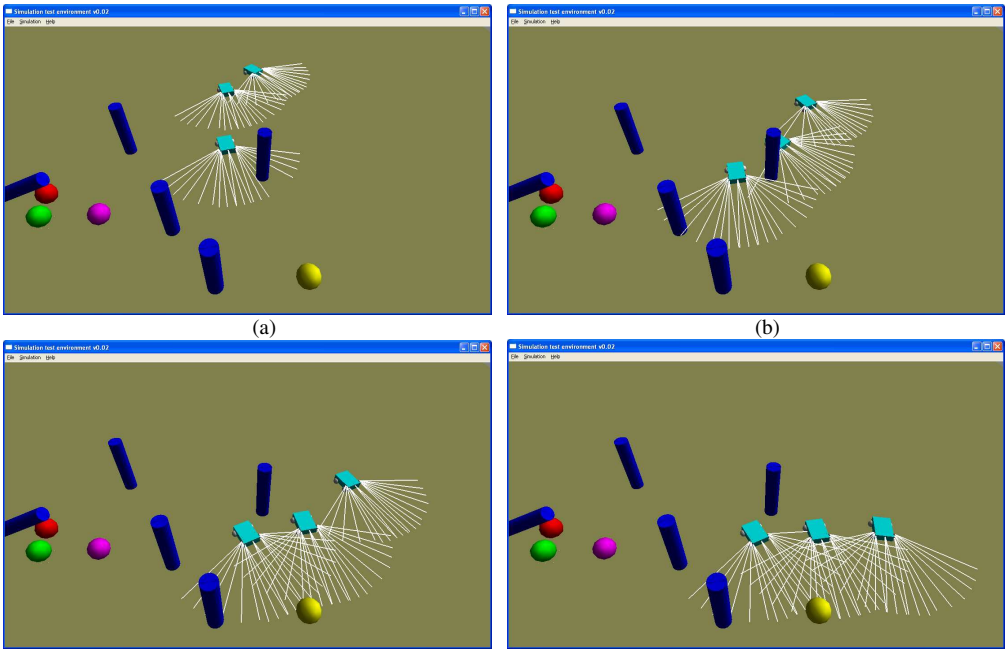
Tabela 2. Resultado das simulações com 10 cilindros.

Simulação	Quantidade de cilindros	Quantidade de chegadas satisfatórias	Quantidade de colisões	Quantidade de trajetos errados
1	10	3	0	0
2	10	3	0	0
3	10	2	0	1
4	10	3	0	0
5	10	3	0	0
6	10	3	0	0

Tabela 3. Resultado das simulações com 20 cilindros.

Simulação	Quantidade de cilindros	Quantidade de chegadas satisfatórias	Quantidade de colisões	Quantidade de trajetos errados
7	20	2	0	1
8	20	3	0	0
9	20	2	0	1
10	20	3	0	0
11	20	1	1	1
12	20	2	1	0

Nas duas colisões observadas, apenas o sensor frontal identificava o cilindro, e o veículo seguia em frente com velocidade reduzida. Esse tipo de desvio existia na base de treino, que devia priorizar giro para direita, porém, por ter produzido alguns erros na simulação, encoraja a construção de uma base de treino maior.



(a) (b) (c) (d)

Figura 5. Seqüências de uma simulação com navegação e desvio satisfatórios.

6. Conclusão

O objetivo deste trabalho foi detalhar o projeto e o desenvolvimento de um sistema multi-agente que opera em um ambiente virtual de simulação realística. Onde uma equipe heterogênea de agentes autônomos trabalha cooperativamente a fim de realizar com sucesso a identificação e o combate de incêndios em áreas florestais, sem intervenção humana. A abordagem implementada adota estratégia centralizada, mas com controle de ação distribuído, onde cada agente possui autonomia na execução das tarefas que lhe são passadas. Os robôs de combate são fisicamente simulados com a biblioteca ODE e as informações sensoriais de cada robô (*e.g.* GPS, bússola, sonar) servem de entrada para uma Rede Neural que controla os atuadores do veículo para assim realizar navegação com desvio de obstáculos de um ponto aleatório até um ponto de atuação no combate ao incêndio. O ambiente desenvolvido apresenta simulação de propagação de fogo baseado em tipos de vegetação, orientação e intensidade do vento. A comunicação através do uso de *blackboard* se mostrou bastante prática e simples de

implementar. Os resultados das simulações demonstram que a Rede Neural controla satisfatoriamente os robôs móveis e que o sistema proposto pode vir a ter um papel muito importante no planejamento e execução de operações reais de combate a incêndios florestais e possivelmente em outras tarefas similares, como acidentes nucleares ou desastres ambientais.

Referências

Antunes, M. A. H. (2000) "Uso de satélites para detecção de queimadas e para avaliação do risco de fogo". Ação Ambiental, 12:24-27.

Barros, L. M., Gonzaga, L., Raposo, A. B. (2007) "Open Scene Graph: conceitos básicos e aplicações em realidade virtual", Tutorial on IX Symposium on Virtual and Augmented Reality (SVR).

Batista, A. C. (2004) "Detecção de incêndios florestais por satélite", Revista Floresta 34, Mai/Ago, 237-241, Curitiba, Paraná.

Bekey, G. A. (2005) "Autonomous Robots: From Biological Inspiration to Implementation and Control". Cambridge, USA: The MIT Press, 2005. 577 p.

CE - Comissão Européia (2006) "O que faz a Europa? Incêndios florestais". http://ec.europa.eu/research/leaflets/disasters/pt/forest.html, setembro.

CPTEC/INPE (2006) "Centro de previsão do tempo e estudos climáticos - Instituto nacional de pesquisas espaciais", www.cptec.inpe.br/queimadas, outubro.

Dudek, G., Jenkin, M. (2000) "Computational Principles of Mobile Robotics" Cambridge, London, UK: The MIT Press, 280 p.

Go, J., Browning, B., Veloso, M. (2004) "Accurate and flexible simulation for dynamic, vision-centric robots". International Joint Conference on Autonomous Agents.

JPL/NASA (2007) "Jet Propulsion Laboratory/NASA" wwwrobotics.jpl.nasa.gov, maio.

Koproski, L.P. (2005) "O fogo e seus efeitos sobre a heperto e a mastofauna terrestre no parque nacional de Ilha Grande", Dissertação de mestrado, UFPR.

LIF - Laboratório de Incêndios Florestais (2006) "Pesquisas e projetos em prevenção e combate de incêndios florestais", UFPR, www.floresta.ufpr.br/~firelab, setembro.

Marchi, J. (2001) "Navegação de robôs móveis autônomos: estudo e implementação de abordagens", Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Osagie, P. (2006) "Distributed Control for Networked Autonomous Vehicles". Dissertação de Mestrado, KTH CSC, Royal Institute of Technology, Sweden.

Osório, F. S., Musse, S. R., Vieira, R., Heinen, M. R., Paiva, D. C. (2006) Increasing Reality in Virtual Reality Applications through Physical and Behavioural Simulation In: Proceedings of the Virtual Concept Conference. Springer Verlag, v.2, p. 1-45.

Pessin, G., et al. (2007) "Simulação Virtual de Agentes Autônomos para a Identificação e Controle de Incêndios em Reservas Naturais", IX Symposium on Virtual and Augmented Reality (SVR), v.1, p. 236-245.

Pfeifer, R., Scheier, C. (1999) "Understanding Intelligence". Cambridge, Massachusetts, USA: The MIT Press.

Remmel, T. K., Perera, A. H. (2001) "Fire mapping in a northern boreal forest: assessing AVHRR/NDVI methods of change detection", Forest Ecology and Management 152:119-129.

Rezende, S.O. (2003) "Sistemas inteligentes: fundamentos e aplicações", Ed. Manole, São Paulo.

Smith, R. (2006) "Open Dynamics Engine v0.5 User Guide".

Algoritmo para Recuperação de Falhas em Sistemas de Gerenciamento de *Workflow*

Adriano Fiad Farias[1,2], **Vinícius Cristiano de Almeida**[1,3], **Alexandre Fieno da Silva**[3]

[1]Universidade Federal de Uberlândia – Pós-graduação em Ciência da Computação
Campus Santa Mônica – 38400-902 – Uberlândia – MG – Brasil

[2]Centro Federal de Educação Tecnológica de Petrolina
Cx.Postal 178 – 56314-520 – Petrolina – PE – Brasil

[3]Faculdade do Noroeste de Minas – Tecnologia em Sistemas de Informação
Cx.Postal 201 – 38600-000 – Paracatu – MG – Brasil

{adrifiad,vinicius}@pos.facom.ufu.br

Resumo. *A recuperação de falhas em sistemas de workflow é um assunto muito discutido no meio científico e um grande desafio. Existem muitos trabalhos relacionados onde as abordagens do problema propõem soluções isoladas e desunificadas. Neste artigo temos como objetivos apresentar um estudo sobre recuperação de falhas em sistemas de gerenciamento de workflow (SGWF) e propor um algoritmo para tratamento do cancelamento das atividades correlacionadas por atividade com falha.*

1. Introdução

Atualmente sistemas de *workflow* se tornaram populares, pois auxiliam a resolução de problemas dos processos organizacionais. Essa popularização fez com que diversas empresas organizadas na forma de consórcios e associações sem fins lucrativos, conjuntamente com a comunidade acadêmica se esforçassem na padronização desses sistemas.

Workflow pode ser definido como qualquer tarefa executada em série ou em paralelo por dois ou mais membros de um grupo de trabalho, visando um objetivo comum. *Workflow* é usado como um sinônimo para "processo de negócio". Segundo a *WorkFlow Management Coalition* (WfMC), *workflow* é a automação total ou parcial de um processo de negócio, durante a qual, documentos, informações ou tarefas são passados de um participante para outro para a realização de alguma ação, de acordo com um conjunto de regras procedimentais. Aplicado dentro do contexto de uma estrutura organizacional, define atividades funcionais e relações. Essas atividades podem ser humanas (reuniões e entrevistas) ou automatizadas (a impressão de um documento).

A automatização do processo de negócio se faz através de três elementos básicos: papéis, regras e rotas, através desses elementos a tecnologia *workflow* possibilita o controle geral de um processo e suas atividades em termos de tempo e movimento, sendo assim possível rastrear "o quê" está atrasado e o "porquê" do atraso. Para que um *workflow* possa ser definido e executado, com participantes realizando atividades de negócios, é necessário um sistema de gerenciamento de *workflow* (SGWF).

A WfMC entende sistemas de gerenciamento de *workflow* como sistemas que definem, executam e gerenciam completamente *workflows* através da execução de um software, cuja ordem de execução é dirigida por uma representação lógica e computadorizada

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 31-38.

de um *workflow*. Uma das contribuições mais importantes do consórcio WfMC foi a padronização de diversos conceitos relativos a *workflows*. A Figura 1 apresenta estes principais conceitos e seus relacionamentos.

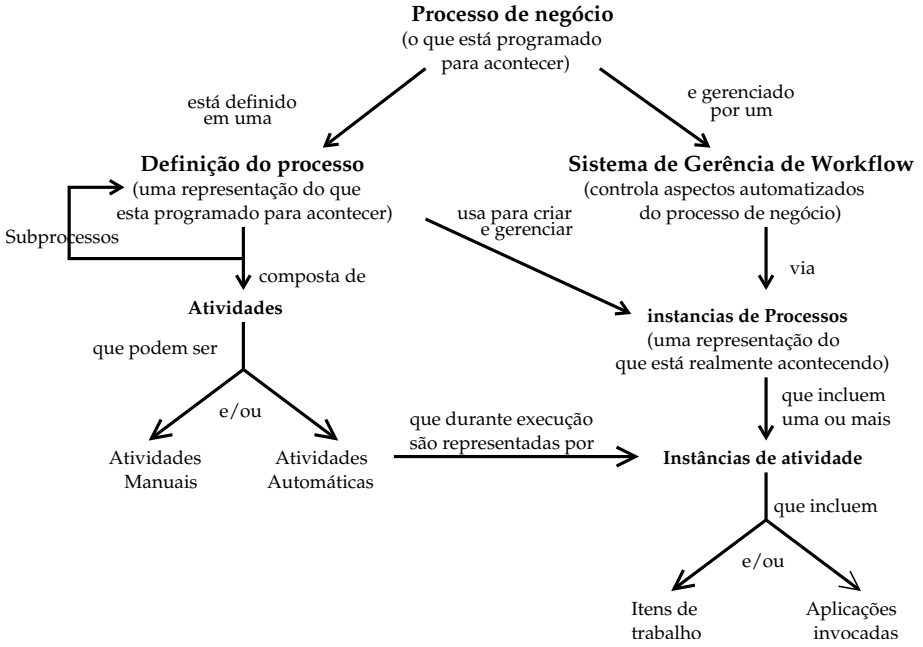


Figura 1. Conceitos relativos a sistemas de *workflow* padronizados pelo WfMC

Segundo [Georgakopoulos et al. 1995] a gerência de *workflow* envolve desde a modelagem dos processos até a sincronização das atividades e dos participantes que realizam os processos. São destacadas as seguintes etapas na gerência de *workflow*: (i) modelagem do processo e especificação; (ii) reengenharia do processo, e (iii) implementação e automação do *workflow*.

O importante é que os sistemas de *workflow* garantam que a execução nunca seja interrompida, independentemente da falha ocorrida. Falha é um evento que causa erros. De acordo com [Heinl et al. 1999] muitas falhas ou exceções, ocorrem nos SGWF, tornando impossível que o sistema identifique essas falhas. Nestas situações, apenas a intervenção humana, de alguém que tenha uma perspectiva global do sistema, pode identificar e definir o mecanismo de tratamento apropriado. Segundo [Worah and Sheth 1997] há três categorias de erros ocasionados por falhas associadas a sistemas de gerência de *workflow*: (i) erros de infra-estrutura; (ii) erros de sistema e (iii) erros de aplicações e usuários.

[Dayal et al. 1991] reconhece uma estrutura fixa de controle e políticas rígidas de compensação. Regras são desenvolvidas para desacoplar a detecção e o tratamento das exceções da execução do fluxo de trabalho aumentando a flexibilidade da abordagem. [Casati et al. 1999] descreve um sistema para tratamento de exceções esperadas baseado em regras desenvolvidas por Dayal, afirmando que as exceções não esperadas devem ser tratadas por humanos uma vez que não foram previstas durante a fase de modelagem.

Em [Fisteus 2005] foi proposto uma arquitetura para análise dos padrões de *workflow*, entre eles o padrão 19 responsável pelo cancelamento das atividades, sendo proposto uma codificação formal. Dentro desta perspectiva este trabalho apresenta um estudo sobre

recuperação de falhas em sistemas de gerenciamento de *workflow* e propõe um algoritmo para tratamento do cancelamento das atividades correlacionadas a atividade com falha.

Para tanto este artigo está dividido da seguinte forma: na seção 2 é apresenta uma visão dos modelos de *workflow*. O tratamento das falhas e exceções são abordados na seção 3. As formas de recuperação de *workflow* em caso de falhas são apresentadas na seção 4. Na seção 5 é apresentado o algoritmo proposto por este trabalho. As conclusões são apresentadas na seção 6 e por fim as referências bibliográficas.

2. Modelos de Workflow

O termo *workflow* transacional foi introduzido em 1993 enfatizando a relevância das propriedades transacionais nos *workflows*. Propriedades essas, necessárias para especificar critérios de coordenação, corretude, consistência de dados, confiabilidade e suporte a falhas [Worah and Sheth 1997].

Dentre as propriedades apresentadas acima, podem ser citadas aqueles relativas ao controle de concorrência sobre ítens de dados vinculados a diversas atividades ou *subworkflows* distintos, à recuperação em caso de falhas e a coordenação das atividades envolvidas em um determinado *workflow*.

O primeiro grande passo na evolução dos modelos transacionais foi exatamente a extensão da estrutura das transações planas para uma estrutura hierárquica, de múltiplos níveis, cujo modelo foi chamado de Modelo de Transações Aninhadas. Neste modelo, (a) uma transação-filha apenas começa depois que sua transação-pai tiver começado; (b) uma transação-pai apenas termina depois que todas as suas transações-filhas forem terminadas e (c) se uma transação-pai é abortada, todas as suas transações-filhas também são.

Modelar um *workflow* como uma transação estendida significa que as subtransações correspondem às tarefas do *workflow* ou a subworkflows e a estrutura de execução da transação estendida corresponde ao fluxo de controle do *workflow*. Transações aninhadas, Sagas, transações flexíveis, dentre outros, são alguns dos modelos estendidos encontrados na literatura. Um breve resumo destes modelos é apresentado em [Rusinkiewicz and Sheth 1995], [Worah and Sheth 1997].

3. Taxionomia de Tratamento de Exceções

As taxonomias existentes que abordam o tratamento de exceções na literatura podem ser vistas em duas distintas perspectivas: sistêmica e organizacional. A perspectiva sistêmica de [Eder and Liebhart 1998] caracteriza-se por falhas e exceções numa única dimensão, composta por dois tipos de falhas (falhas elementares e nas aplicações) e dois tipos de exceções (exceções esperadas e não esperadas).

As falhas elementares apresentam faltas em nível do sistema de suporte ao SGWF (sistema operacional, SGBD ou faltas na rede). As falhas nas aplicações, apresentam falhas na implementação das tarefas (entrada inesperada de dados);

As exceções esperadas são eventos ou situações que podem ser previstas durante a fase de modelagem mas, que não correspondem ao comportamento "normal" de um processo. Este tipo de exceção pode ocorrer com freqüência e originar uma quantidade de trabalho significativo em seu tratamento. Devem ser previstos mecanismos para o tratamento destas situações uma vez que podem ocorrer com freqüência [Eder and Liebhart 1998].

[Casati et al. 1999] identifica quatro classes de exceções esperadas de acordo com os eventos que as originam: (a) exceções de fluxos de trabalho, (b) exceções dos dados, (c) exceções temporais e (d) exceções externas.

As exceções não esperadas ocorrem quando a semântica do processo não é corretamente modelada no sistema. São exemplos, mudanças nas regras devido a alterações legislativas, mudanças estruturais que envolvem a organização ou alteração no processamento de uma compra efetuada por um cliente especial.

Ainda em [Casati et al. 1999], as exceções não esperadas resultam de inconsistências entre a modelagem do processo no fluxo de trabalho e a execução efetiva. Essas exceções resultam de falhas de projeto ou projeto incompleto, melhoramentos ou alterações no negócio ou a necessidades de satisfação dos clientes não previstas durante a fase de modelagem. Este tipo de exceções não esperadas é freqüente em ambientes dinâmicos ou complexos. Essas exceções podem obrigar a interrupção da execução automática do processo e a intervenção de um operador [Heinl et al. 1999].

Em situações onde este tipo de exceção não esperada ocorre com freqüência, deve ser considerada a reformulação do modelo do fluxo de trabalho, a adoção de outras tecnologias baseadas em trabalho cooperativo ou sistemas de fluxos de trabalho adaptativos.

4. Recuperação em Caso de Falhas

Existem várias classificações para falhas na literatura. Normalmente, essas classificações agrupam as falhas em físicas e humanas. Falhas físicas referem a falhas de componentes de hardware e falhas humanas compreendem falhas de projeto, decorrentes das fases de desenvolvimento do software, falhas de interação podem ser acidentais ou intencionais.

Caso o estado errôneo do sistema não seja tratado em um tempo determinado, ocorrerá um defeito, que se manifestará pela não execução ou mudança indesejada no serviço especificado. Como pode ser visto na Figura 2, além de indicar um estado errôneo, a ocorrência de um defeito realimenta o ciclo e pode gerar novas falhas. Esse fenômeno é conhecido como propagação da falha e deve ser contido através do seu confinamento.

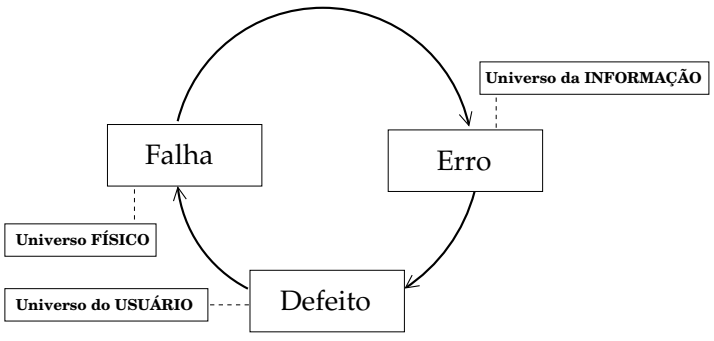


Figura 2. Ciclo entre a ocorrência de falhas, erros e defeitos (Brito:05)

As falhas também podem ser classificadas segundo a sua duração. Nessa perspectiva, as falhas são distribuídas em três grupos: (i) transientes, quando existe a chance das falhas ocorrerem mais de uma vez mas não obrigatoriamente, por exemplo, a invasão do sistema por um hacker; (ii) intermitente, quando a ocorrência da falha se repete, mas não necessariamente em períodos definidos, por exemplo, a manifestação de um vírus pre-

sente no sistema; e (iii) permanentes, que são caracterizadas pela sua presença constante no sistema, por exemplo, falhas de especificação e implementação [Brito 2005] .

Segundo [Worah and Sheth 1997] falhas associadas ocasionam três categorias de erros a SGWF: (i) erros de infra-estrutura - falhas de comunicação ou hardware; (ii) erros de sistema - falha no próprio software do sistema de *workflow* e (iii) erros de aplicações e usuários - decorrente da execução de determinada tarefa por parte dos mesmos.

Se o erro é de sistema e ocorre no banco de dados onde estão contidas as informações dos workflows, o mecanismo de replicação, se presente, pode resolver o problema. Neste caso, quando os dados necessários para a execução da tarefa estiverem locais ao dispositivo em que ela está sendo executada, a execução não é interrompida.

No contexto de recuperação em caso de falhas, algumas alternativas podem ser utilizadas com o intuito de evitar que a execução seja interrompida. Por exemplo, quando uma tarefa não puder ser executada, diversas tentativas podem ser feitas até que se consiga executá-la (como no caso da falta de recursos) ou tarefas alternativas podem ser executadas (quando um certo número de tentativas falhou) para evitar que a tarefa falhe. Se uma tarefa alternativa for executada no lugar da que tinha falhado, considera-se que a falha foi resolvida. No entanto, esta abordagem de utilização de tarefas alternativas ou novas tentativas não ocorre em grande parte dos sistemas de workflow. Para que a recuperação seja possível no nível de tarefas e não apenas no nível de workflows, é necessário que toda a informação de execução do workflow, inclusive aquelas relativas a cada tarefa individual, sejam armazenadas de modo persistente.

Se a execução de uma instância de *workflow* é de fato abortada, ações compensatórias devem ser executadas pela máquina de execução para desfazer os efeitos semânticos das operações, sempre que possível. Estas ações não levam, necessariamente, o banco de dados que controla a execução do *workflow* para o estado válido anterior à falha, mas sim o leva para um estado válido e semanticamente próximo. Isso porque muitas vezes os efeitos transacionais das tarefas de um *workflow* não podem ser desfeitos.

Existem basicamente duas maneiras possíveis de se recuperar uma instância de *workflow* que falhou: através de *backward recovery* e através de *forward recovery*. Na abordagem de *backward recovery* a instância de *workflow* é retornada, através de ações compensatórias, para o estado consistente que existia antes da transação que representa a instância de *workflow* abortada (que falhou). Na abordagem de *forward recovery* a instância que falhou volta para um estado válido e continua a sua execução a partir do ponto em que havia falhado.

Mecanismos de pontos de checagem podem ser úteis tornando o processo de recuperação eficiente, salvando dados de tempos em tempos. Mecanismos de recuperação são sempre intimamente ligados à infra-estrutura do sistema, à sua arquitetura, à natureza dos executores, aos tipos de tarefas e da natureza da própria aplicação de *workflow*. Assim um único mecanismo de recuperação de falhas poderá não ser capaz de resolver problemas de vários sistemas semanticamente distintos.

5. Proposta Algorítmica

No estudo de [Eder and Liebhart 1998], [Klein and Dellarocas 2000] e [Dayal et al. 1991], prevêem que as possíveis causas de falhas e exceções são difí-

ceis ou mesmo impossíveis de serem tratadas unificadamente. Tornando os sistemas mais complexos e difíceis de gerir. Assim, preparar sistemas para lidarem com estas situações durante a fase de execução é um fator crítico de sucesso na implementação de SGWF.

As organizações são ambientes complexos e a aproximação tradicional baseada apenas na integridade e consistência dos dados não constitui um suporte suficientemente sólido. Apesar das limitações reconhecidas, o sistema de suporte do SGWF deve ser suficientemente robusto e suportar a integridade e consistência dos dados relevantes.

Como mencionado anteriormente, um esforço significativo foi investido no tratamento das falhas e exceções utilizando técnicas de sistemas de SGBD. No entanto, a semântica das tarefas nos SGWF excede largamente os modelos transacionais de SGBD. Por exemplo, em um *processo_a* em execução, a *atividade_n* desse processo (i.e, telefonema para um cliente) falha (por este não atender) não é necessário fazer nada de momento; é esperado por um tempo pré-determinado, que não exceda o tempo de execução desta atividade, para repetir a *atividade_n* até que esta seja concluída.

A semântica do tratamento de erros em sistemas tradicionais é muito rígida para os SGWF [Worah and Sheth 1997]. A integridade do sistema é garantida assegurando que caso a *atividade_n* seja cancelada, as *atividades_n+m* que instanciarem as alterações provisórias sejam informadas e reajam em concordância. Por outro lado, políticas rígidas de compensação implementam as atividades que devem ser efetuadas para voltar a colocar o sistema em um estado coerente sempre que uma atividade é cancelada.

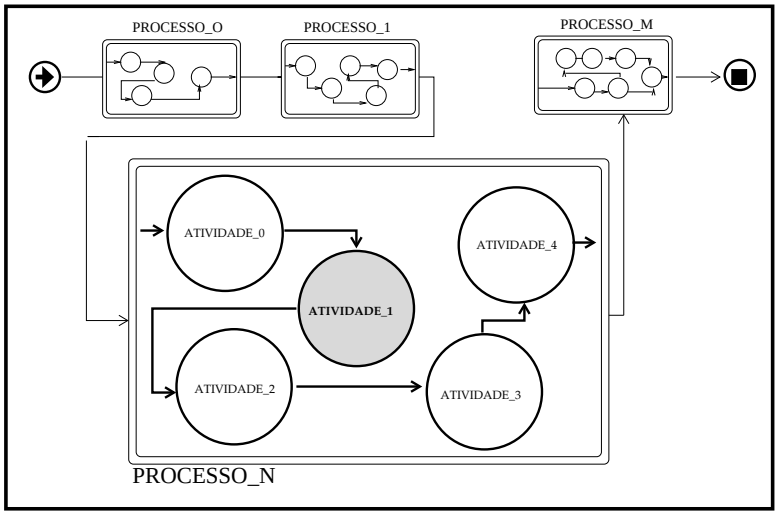


Figura 3. Processo de cancelamento de atividade.

Considere o *workflow* com ocorrência de falha na execução de atividades relacionadas a determinado processo. Se o processo possuir atividades correlacionadas entre si, como proceder? Nesse momento é necessário um algoritmo que trate o cancelamento das atividades correlacionadas a atividade com falha, implementando mecanismos que assegurem o sistema a retornar para um estado válido ou o mais próximo possível.

Por exemplo, uma *atividade_1* de um *Processo_N* falha. Todas as atividades relacionadas a este processo são canceladas. O sistema de workflow, deve implementar métodos para que a execução retorno a um estado válido, no caso o processo anterior (*processo_1* na Figura 3). Partindo dessa premissa, propomos algoritmicamente na Figura 4,

solução frente ao tratamento desse tipo de falha ocorrida.

O algoritmo está posicionado da seguinte forma *[Processo_0,..., Processo_N,..., Processo_M]* e a atividade do processo *[A_0,..., A_n,..., A_m]*. Considere a modelagem deste algoritmo mediante a utilização de um atributo adicional booleano, cujo valor inicial deve ser falso. O algoritmo como se pode observar, verifica se uma atividade anterior a atividade a ser executada foi cancelada ou não.

```
PROCESSO_N

1   Criar Tipo Cancel
2           cancelamento : booleano
3
4   Entidade A_n : cancel
5
6   VERIFICA_TRANSAÇÃO A_n−1
7      SE  ((estad.A_n != cancelado) e (estado.A_n−1 = cancelado))
8                  ENTÃO (A_n.cancel = true)
9
10  INÍCIO_TRANSAÇÃO A_n
11     SE ((estado.A_n = parado) e (estado.A_n−1 = finalizado)
12        e (A_n.cancel = false))
13              ENTÃO Início execução A_n
14                     SE término execução esperada (estado.A_n = finalizado)
15                     SE término execução inesperada (estado.A_n = cancelado)
16
17  CANCELAMENTO_TRANSAÇÃO A_n
18     SE ((estado.A_n != cancelado) e (estado.A_n = finalizado)
19       e (A_n.cancel = true))
20              ENTÃO Início cancelamento A_n
```

Figura 4. Algoritmo de tratamento de falhas das atividades.

Se o estado da transação atual for diferente de cancelado e o estado da atividade anterior foi cancelado, então a atividade atual é cancelada (Figura 4 - linhas 7-8). Após a *verificação_transação A_n-1*, o algoritmo executa o início da transação *A_n*.

Se o estado da transação *A_n* encontra-se parado e o estado da transação *A_n-1* finalizado e o valor booleano de *A_n*.cancel for igual a false, então se dá início a execução da atividade *A_n*. É monitorado se o término da execução é esperado ou inesperado, atribuindo ao estado da atividade *A_n*, finalizado ou cancelado, respectivamente.

Após o teste para início da execução da transação duas possibilidades se fazem possíveis. Se a condição das linhas 11-12 forem verdadeiras, é dado início a transação, podendo terminar de forma esperada ou inesperada. Se a condição for falsa, o algoritmo verifica o cancelamento da transação da atividade *A_n*.

Para a verificação do cancelamento da transação *A_n*, são observadas algumas variáveis como o estado da transação *A_n* e o valor booleano de *A_n*.cancel. Se o estado de *A_n* for igual a cancelado ou parado e o estado da variável *A_n*.cancel for true, então se dá início ao cancelamento da atividade. É interessante lembrar que está proposta foi baseada no motor de *workflow* YAWL, onde em sua arquitetura implementa métodos de recuperação de falhas *forward recovery*.

Apesar dos esforços para tratar automaticamente as exceções, parece impossível incluir semântica específica de uma tarefa numa plataforma de recuperação genérica. Uma vez que o comportamento das tarefas é ortogonal ao processo de fluxo de trabalho.

6. Conclusões

Este trabalho apresenta uma proposta para implementação de um algoritmo que trata o cancelamento das atividades subsequentes a atividade que ocorreu falha. Constatamos, que o assunto é amplo e muito discutido no meio científico. Existem vários trabalhos relacionados, onde as abordagens do problema propõem soluções isoladas e desunificadas, tornando o tratamento de falhas em SGWF algo desafiador. Esse algoritmo proposto é capaz de fazer o cancelamento de atividades que ocasionem falhas dentro de um processo em execução, adotando política preventiva de erros.

Como futuras extensões a este trabalhos será efetuado um aprofundamento no estudo, para implementação do algoritmo proposto junto ao motor *workflow* YAWL. Aplicação de técnicas de detecção de falhas em conjunto com o algoritmo proposto. A utilização de técnicas de mineração de dados e unificação de todas as possíveis soluções descritas na literatura para tratamento e recuperação de falhas em sistemas de workflow.

Referências

Brito, P. H. (2005). Um método para modelagem de exceções em desenvolvimento baseado em componentes. *Dissertação de mestrado - UNICAMP*.

Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G. (1999). Specification and implementation of exceptions in workflow management systems. *ACM Trans. Database Syst.*, 24(3):405–451.

Dayal, U., Hsu, M., and Ladin, R. (1991). A transactional model for long-running activities. In *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pages 113–122, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Eder, J. and Liebhart, W. (1998). Contributions to exception handler in workflow management. Int. Conf. on Extended Database Technology (EDBT'98).

Fisteus, J. A. (2005). Definición de un modelo para la verificación formal de procesos de negocio. Tesis Doctoral. Universidad Carlos III de Madrid. Departamento de Ingeniería Telemática. Leganés,Spain.

Georgakopoulos, D., Hornick, M. F., and Sheth, A. P. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153.

Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., and Teschke, M. (1999). A comprehensive approach to flexibility in workflow management systems. In *WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 79–88, New York, NY, USA. ACM Press.

Klein, M. and Dellarocas, C. (2000). A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work*, 9(3/4):399–412.

Rusinkiewicz, M. and Sheth, A. P. (1995). Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond.*, pages 592–620.

Worah, D. and Sheth, A. P. (1997). Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34.

Framework em Java para Desenvolvimento de Aplicações contendo Janelas Gráficas

Leandro Salvatti Piscke[1], Adilson Vahldick[1]

[1]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil
piscke@gmail.com, adilsonv77@gmail.com

Resumo. *Este artigo descreve a arquitetura de um framework desenvolvido para agilizar e simplificar o desenvolvimento de aplicações desktop utilizando objetos remotos com a tecnologia EJB3. Ao final do artigo é apresentado um exemplo de como utilizar o framework. Esse trabalho também serve como referência na exemplificação de uso dos padrões de projeto e é resultado de um projeto de conclusão de curso de graduação em Ciências da Computação.*

1. Introdução

O desenvolvimento de aplicações exige cada vez mais conhecimento técnico dos programadores, se comparado com as tecnologias usadas nas décadas passadas. Essa tendência vem em função da multiplicidade de ambientes que as aplicações precisam suportar, além de considerável mudança de requisitos por parte dos clientes, seja por ajuste ou alteração do foco nos negócios, seja pelas alterações de legislação.

Um bom ferramental na produção de software precisa considerar esse ciclo de vida de constantes alterações que os sistemas vivenciam. As ferramentas geradoras de código são uma alternativa, contudo elas impõem limitações quando uma aplicação possui muitas particularidades. O ideal é uma solução que ao mesmo tempo agilize o trabalho do desenvolvedor e permita que ele ajuste o grau de automatização, fazendo com que situações comuns utilizem soluções comuns, e problemas diferenciados possam ser tratados pelos programadores.

Os padrões ajudam na produção de software diminuindo o tempo de desenvolvimento, e principalmente no tempo de manutenção, que é considerada a fase mais custosa do ciclo de vida do software. Entretanto, a utilização de padrões exige um alto nível técnico da equipe, além de uma minuciosa documentação e catalogação das classes disponíveis.

Conclui-se que existe um paradoxo no uso de padrões: de um lado é inegável a relevância na sua utilização e do outro, o custo em manter uma equipe com elevado nível técnico e pela disseminação das informações sobre as bibliotecas de classes mantidas por ela.

Na busca de uma solução é proposto neste trabalho um *framework* que permite o desenvolvimento rápido de janelas nas aplicações, e que estes recursos exijam o conhecimento de poucos padrões permitindo que programadores com nível técnico não tão elevado possam fazer parte da equipe, e que ao mesmo tempo programadores mais experientes possam customizar as classes na arquitetura proposta.

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 39-50.

Nesse trabalho foram utilizados vários componentes e *frameworks* de terceiros, o que tornou motivador e norteador o desenvolvimento do próprio trabalho, pois um bom *framework* é aquele que pode ser utilizado e reutilizado várias vezes por pessoas e projetos diferentes, e que não tiveram influência nenhuma na produção dele.

2. Arquitetura do Framework

Uma das características desse *framework* é a disponibilização pela Internet das interfaces com o usuário. Por essa razão, ele foi dividido em duas versões: uma para execução no servidor e outra para execução nas máquinas clientes. A figura 1 apresenta as duas versões e a divisão em camadas dentro de cada uma delas.

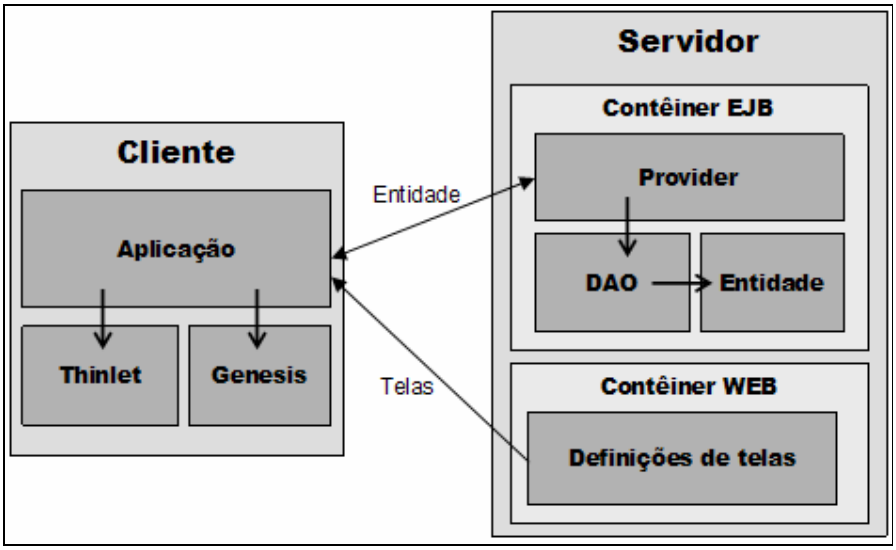


Figura 1. Arquitetura do framework

2.1. Aplicação Servidor

A versão do servidor usa os dois tipos de contêiner da especificação JEE. No contêiner EJB é onde se concentram as classes que o desenvolvedor utilizará e estenderá do *framework*. A figura 2 apresenta o diagrama com essas classes. Esse conjunto de classes foi estruturado seguindo uma arquitetura em camadas, conforme será descrito nos parágrafos abaixo.

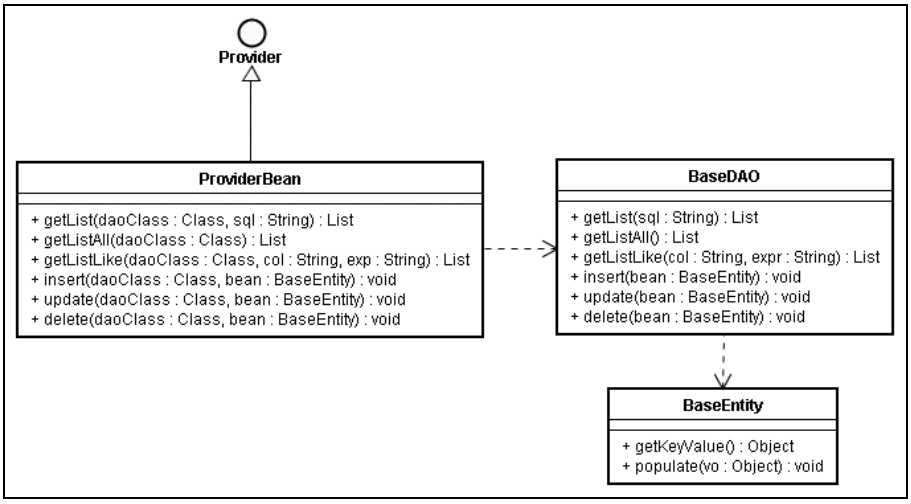


Figura 2. Classes da versão servidor

Toda classe que representa uma entidade (ou classe de modelo) precisa estender da classe `BaseEntity` e utilizar as anotações de entidades do EJB3. Os objetos dessa classe são transferidos entre a aplicação servidora e a aplicação cliente. Essa classe segue o padrão *Transfer Object* (ALUR, CRUPI E MALKS, 2004).

A persistência dos objetos entidade é gerenciada pelos objetos das classes que estendem de `BaseDAO`. A implementação de classes descendentes só se faz necessário quando os métodos oferecidos por essa classe base não atenderem. Por exemplo, a classe `BaseDAO` possui uma série de métodos para retornar uma lista de objetos (`getList`, `getListAll` e `getListLike`). Se o programador precisar de uma lista com um critério diferente, então ele precisa implementar esse método na classe descendente de `BaseDAO`. Essa classe segue o padrão *Data Access Object* (ALUR, CRUPI E MALKS, 2004).

A classe `ProviderBean` e a interface `Provider` foram definidas para a responsabilidade de desempenhar a tarefa de disponibilizar os serviços do *framework*. A aplicação cliente se comunicará com objetos de classes descendentes de `ProviderBean`, implementadas pelo programador usuário do *framework.* As classes descendentes precisam utilizar as anotações de objetos sem estado (*stateless*) do EJB3. A implementação dessas classes segue o padrão *Session Façade* (ALUR, CRUPI E MALKS, 2004; FOWLER, 2006) e *Business Object* (ALUR, CRUPI E MALKS, 2004).

No contêiner web foi desenvolvida uma aplicação para gerenciar as definições das telas da aplicação, armazenadas como arquivos XML. A utilização desses arquivos é feita pela aplicação cliente conforme descrita na próxima seção. Através do navegador o administrador adiciona, altera e remove os arquivos XML das telas da aplicação. A figura 3 apresenta a página principal dessa aplicação web.

Figura 3. Página principal da aplicação Web de Gerenciamento de Telas

Existem partes das janelas que se repetem, e poderiam ser agrupadas e consideradas como um único componente. Um exemplo é uma barra de botões para representar ações de navegação pelos registros de uma tabela, além de iniciar qualquer atividade de inclusão, alteração ou exclusão também conhecidas como padrão CRUD (YODER, JOHNSON E WILSON, 1998). Não só partes se repetem, mas depois que o desenvolvedor estabeleceu um padrão de interfaces com o usuário, ele acaba copiando o arquivo XML inteiro para criar uma nova janela.

Como uma alternativa para utilizar de componentização foram implementadas algumas *Custom Tags* (SUN..., 2007) de exemplo para inspirar o desenvolvedor na produção padronizada das janelas. Nesse caso, em vez de dispor de um arquivo XML, o programador monta uma página JSP que produzirá o XML ao cliente. Cada *Custom Tag* representa um conjunto de componentes do *Thinlet*. Foram implementadas as seguintes *Custom Tags*:

- `EditSearch`: concentra um painel com um rótulo, um campo de edição e um botão;
- `LabelLookup`: contém um rótulo seguido de um painel com um campo de edição e um botão;
- `LabelCombobox` e `LabelEditTag`: ambos correspondem a um rótulo seguido de uma caixa de seleção e campo de edição respectivamente.

2.2. Aplicação Cliente

A aplicação cliente foi desenvolvida para prover telas com interface gráfica. Para isso foi utilizado o *Thinlet*, que é um *toolkit* GUI leve para Java (BAJZAT, 2006). Em uma única classe Java processa a hierarquia e propriedades da interface gráfica, trata interação com usuário e permite invocar métodos de objetos específicos da aplicação.

Para desenvolver uma interface é necessário editar um arquivo XML que descreva os atributos dos componentes suportados pelo *Thinlet*. Essas telas também podem ser projetadas utilizando a ferramenta *ThinG*, que foi desenvolvida com o próprio *Thinlet* (MÖBIUS, 2007). A figura 4 apresenta a interface principal dessa ferramenta.

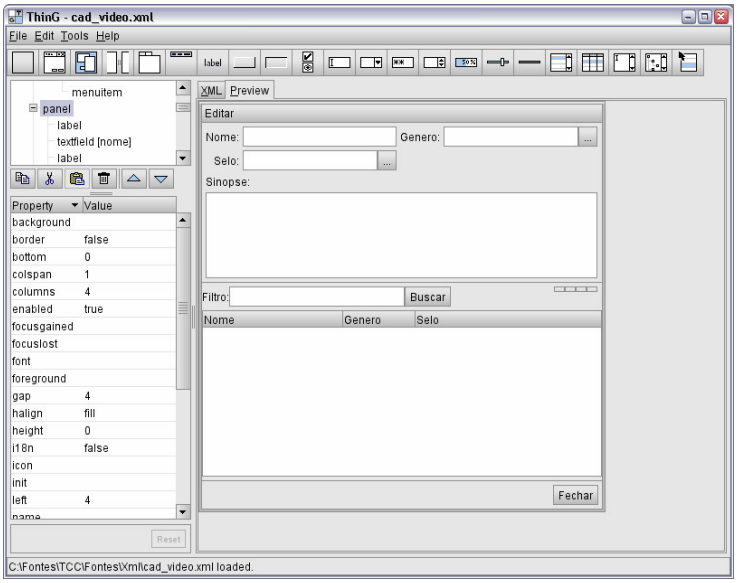


Figura 4. Tela principal do ThinG

As classes do *framework* para a aplicação cliente estão representadas na figura 5.

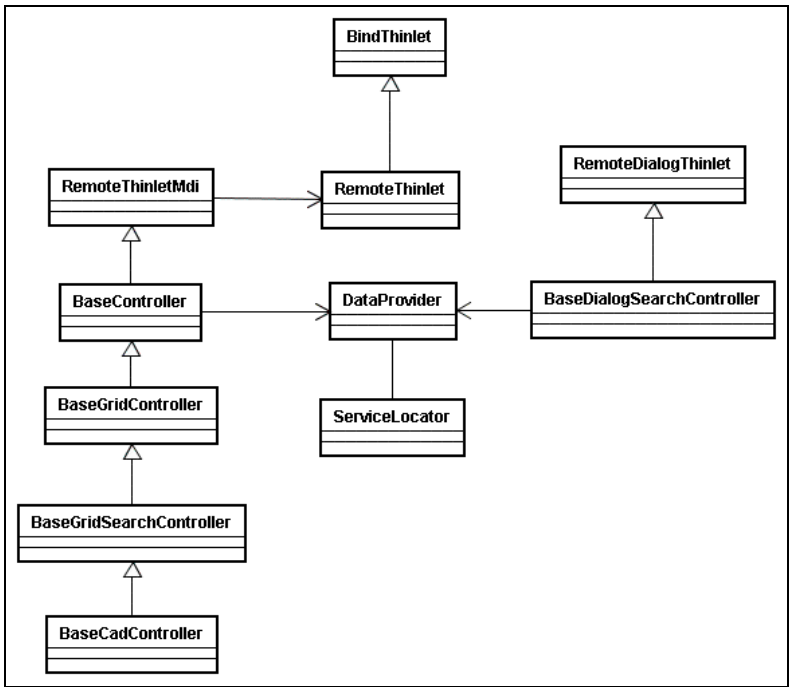


Figura 5. Classes da aplicação cliente

A classe `RemoteThinlet` é responsável por solicitar as definições das telas ao servidor e utilizar o *Thinlet* para a montagem das telas. A solicitação dos documentos XML foi simplificado com o uso do componente `HTTPClient` do projeto Jakarta (APACHE..., 2007b). Essa montagem está programada na classe `BindThinlet` que é ancestral de `RemoteThinlet`. A `BindThinlet` é descendente de uma classe de outro *framework* que é o *Genesis* (SUMMA..., 2006) que implementa o processo de ligação entre as telas *Thinlet* com objetos no padrão *JavaBeans*. A classe `BindThinlet` implementa o padrão *Adapter* (GAMMA et al, 2000), pois recebe um objeto de entidade e mapeia para o formato exigido da classe `BaseThinlet` do *Genesis*. Nesse mapeamento foi utilizado o componente `BeanUtils` do projeto Jakarta (APACHE..., 2007a).

Os objetos de controle da aplicação cliente são instâncias da classe `BaseController` ou descendentes. Essa classe estabelece uma ponte com os serviços implementados na aplicação servidora, delegando essas tarefas a objetos do tipo `DataProvider` e `ServiceLocator`. A classe `DataProvider` implementa o padrão *Business Delegate* (ALUR, CRUPI E MALKS, 2004) e é responsável em comunicar-se com os serviços remotos do *framework*. A classe `ServiceLocator` é que fornece os objetos remotos ao `DataProvider` comunicando-se com o servidor.

As classes descendentes de `BaseController`, como `BaseGridController`, `BaseGridSearchController` e `BaseCadController`, são padrões de tela com recursos como uma grade para apresentar a lista de registros, uma barra com configurações de pesquisa e funções básicas de inclusão, alteração e exclusão, também conhecido como modelo CRUD (YODER, JOHNSON E WILSON, 1998). O desenvolvedor pode simplesmente utilizar essas classes ou estendê-las quando a janela possuir algum recurso adicional.

3. Estudo de Caso

Para demonstrar a utilização do protótipo foi implementado um sistema de vídeo-locadora atendendo os casos de uso apresentados na figura 6. O modelo conceitual desse sistema está representado na figura 7.

Foram definidas cinco etapas para utilizar o *framework*, quatro delas na aplicação servidora que são (i) implementar as entidades, (ii) classes DAO, (iii) regras de negócios e (iv) criar definições de telas, e mais outra na aplicação cliente que é a implementação das classes de controle.

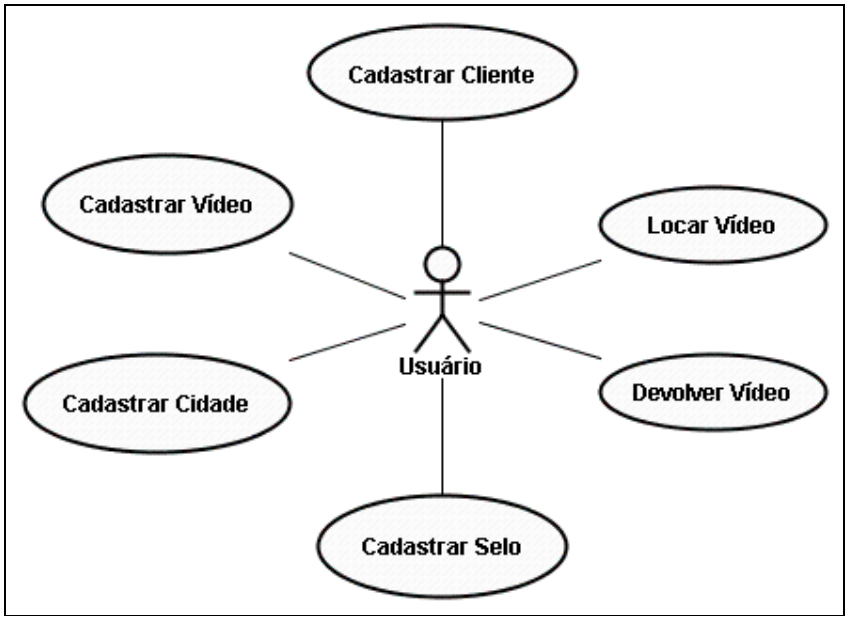


Figura 6. Diagrama de casos de uso do sistema de vídeo-locadora

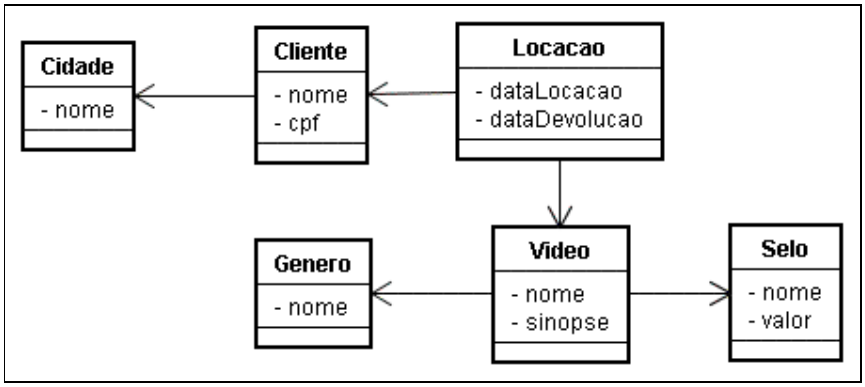


Figura 7. Modelo conceitual do sistema de vídeo-locadora

Na primeira etapa são implementadas as classes de entidades. Uma delas identificada no protótipo é a classe `Locacao`. A figura 8 apresenta parte do código dessa classe. Essa classe segue o padrão *JavaBeans* e são adicionadas anotações EJB3 de entidade para desempenhar o mapeamento objeto-relacional.

```
// imports...
@Entity
@Table(name = "locacao")
public class Locacao extends BaseEntity implements Serializable{

   private int codigo;
   private Cliente cliente;
   // demais atributos

   @Id
   @GeneratedValue
   public int getCodigo() { return codigo; }
   public void setCodigo(int codigo) { this.codigo = codigo; }

   @ManyToOne(optional = false)
```

```
   public Cliente getCliente() { return cliente; }
   public void setCliente(Cliente cliente) { this.cliente = cliente; }

   // ...
}
```

Figura 8. Entidade locação

A segunda etapa envolve a implementação de classes DAO. Basicamente basta estender da classe `BaseDAO`. No caso da locação, para retornar os vídeos locados e ainda não devolvidos de um cliente, foi necessário adicionar um novo método que é o `getVideos` (figura 9).

```
// imports
public class LocacaoDAO extends BaseDAO<Locacao> {

   public LocacaoDAO(EntityManager entityMnger) { super(entityMnger); }
   public LocacaoDAO() { super(); }

   public List<Locacao> getVideos(Cliente cli){
       Query q = getEM().createQuery("from Locacao where " +
                 "cliente.codigo=:codigo and dtDevolucao is null");
       q.setParameter("codigo", cli.getCodigo());
       return q.getResultList();
   }
}
```

Figura 9. Classe DAO para a entidade locação

Na terceira etapa acontece o desenvolvimento das regras de negócio, onde essas classes estendem da classe `ProviderBean` e implementam um EJB de sessão. Também é necessário definir uma interface com os serviços a serem disponibilizados à aplicação cliente, pois a referência será pela interface e não pela classe. Cada caso de uso poderia corresponder a uma classe de serviços. A figura 10 apresenta a classe de serviços de locação.

```
// imports
public @Stateless
class LocadoraProvider extends ProviderBean implements Provider,
       LocadoraInterface {

       public void devolucao(Locacao loc) {
          LocacaoDAO dao = new LocacaoDAO(getEM());
          loc.setDtDevolucao(new Date());
          dao.update(loc);
       }

       public List<Locacao> getVideos(Cliente cli) {
          LocacaoDAO dao = new LocacaoDAO(getEM());
          return dao.getVideos(cli);
       }
// outros métodos
}
```

Figura 10. Classe de serviços do cadastro de locação

A etapa seguinte é a definição das telas. Inicialmente as telas foram desenvolvidas com o *ThinG* e depois migradas para arquivos JSP, demonstrando a utilização das *Custom Tags*. A figura 11 apresenta o código da janela de locação e a figura 12 a representação gráfica dessa janela. Foi adicionado inclusive a *Custom Tag*

`labellookup` nessa janela. Os conteúdos dos atributos `action` correspondem aos métodos que a janela invocará dos objetos controladores.

```
<%@ page language="java" contentType="text/xml; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://piscke.server.org/tags" prefix="fk"%>
<dialog columns="1" gap="4" left="4" top="4" right="4" height="300"
                                resizable="true" scrollable="true" width="400">
    <panel columns="2" gap="4" left="4" right="4">
       <fk:labellookup name="cliente" buttonText="..." text="Cliente:"
                                      action="showCliente"></fk:labellookup>
    </panel>
    <separator/>
    <panel columns="2">
       <panel halign="left" right="4" weightx="1">
         <button action="locacao" text="Locação" tooltip="Locação" />
         <button action="devolucao" text="Devolução" tooltip="Devolução" />
       </panel>
       <panel halign="right" right="4" weightx="1">
         <button action="delete" alignment="right" halign="right"
                                   icon="img/excluir.gif" tooltip="Excluir" />
       </panel>
    </panel>
    <table name="grid" weighty="1" weightx="1">
        <header>
            <column name="video" text="Video"/>
            <column name="video.selo.valor" text="Valor"/>
        </header>
    </table>
    <separator/>
    <panel columns="2">
      <panel halign="right" right="4" weightx="1">
        <label alignment="right" text="Total:"/>
        <textfield editable="false" name="total"/>
      </panel>
    </panel>
    <jsp:include page="inc/fechar.jsp"></jsp:include>
</dialog>
```

Figura 11. Definição da tela de locação e devolução de vídeos

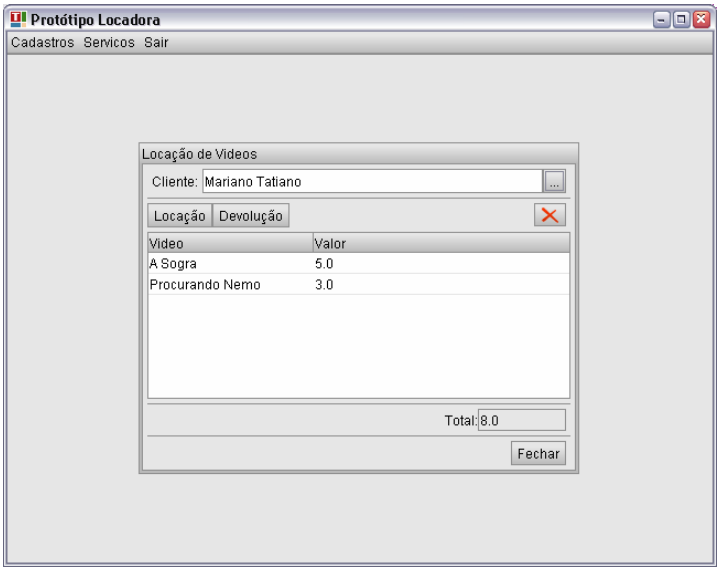


Figura 12. Representação gráfica da tela principal e da tela de locação e devolução de vídeos

A última etapa é a implementação das classes controladoras. Elas devem estender das classes descritas na seção 2.2. No caso da locação foi criada uma classe descendente de `BaseGridController`. A figura 13 apresenta a classe `LocacaoController` ilustrando dois métodos para os requisitos de locação e devolução. Deve-se observar que esses métodos foram relacionados na definição da tela citada na figura 11.

```
// imports
public class LocacaoController extends BaseGridController<Locacao> {

   public void devolucao() throws ScreenNotFoundException{
      Locacao obj = getSelected();
      if (obj != null) {
         locaServ.devolucao(obj);
         populateGrid(locaServ.getVideos(cliente));
      } else
         MessageDialog.show(getThinlet(), "Informação",
                                                  "Não há item selecionado!");
   }

   public void locacao() throws Exception{
      if (cliente == null){
         MessageDialog.show(getThinlet(),"Informação","Selecione um cliente!");
      }else{
         BaseDialogSearchController<Video> vid =
             new BaseDialogSearchController<Video>(getThinlet(), "lookup.jsp",
                          new DataProvider(VideoDAO.class), "grid", "nome");
         if (vid.showSearch()) {
            Locacao loc = new Locacao();
            loc.setCliente(cliente);
            loc.setVideo(vid.getObj());
            setText(find("video"), vid.getObj().getNome());
            setText(find("cliente"), cliente.getNome());
            locaServ.locacao(loc);
            populateGrid(locaServ.getVideos(cliente));
         }
         atualizaSoma();
      }
   }
   // outros métodos
}
```

Figura 13. Classe de controle do cadastro de locação

Como janela principal do sistema foi implementado um descendente de `RemoteThinlet` e também definido um documento XML contendo as opções do menu.

4. Conclusões

Na implementação do estudo de caso a preocupação foi voltada em atender o modelo conceitual e os casos de uso. Isso demonstrou que o foco do desenvolvedor foi quanto à lógica de negócio, deixando a infraestrutura por conta do *framework*. Esse sistema está pronto para execução em um ambiente multiusuário e ao mesmo tempo independente de plataforma.

Outro fato que se pode comprovar é que a quantidade de conhecimento para a utilização do *framework* é consideravelmente baixa, permitindo que programadores iniciantes possam fazer parte da equipe. Exige-se do programador o entendimento das

anotações de EJB de entidade e que classes do *framework* devem ser estendidas para implementar a aplicação. Ao mesmo tempo, programadores mais experientes podem customizar suas janelas estendendo das classes de controle e implementando novas *Custom Tags*.

Várias tecnologias foram imprescindíveis no sucesso desse *framework*. Pode-se destacar o *Thinlet*, o *Genesis* e o EJB3, que permitiram transparência quanto a disponibilização de serviços remotos e quanto ao mapeamento objeto-relacional. As classes `BaseEntity` e `ProviderBean` foram os EJBs implementados: o primeiro de entidade e o segundo de sessão. Nesse projeto também foi desenvolvida a classe `BindThinlet` é descendente de uma classe do *Genesis*. O *Thinlet* é utilizado indiretamente, visto que o `BindThinlet` faz a comunicação entre o documento XML que contém as definições de tela (no formato *Thinlet*) e o próprio *Thinlet* que monta as telas.

Comparando esse trabalho em relação ao *Genesis*, o *framework* aqui desenvolvido destaca-se por utilizar as especificações EJB3 e facilitar o desenvolvimento de telas no modelo CRUD. O *Genesis* é mais genérico e não implementa as facilidades para o desenvolvimento de telas no modelo CRUD. Porém, destaca-se por possuir suporte a diferentes bibliotecas gráficas como: *Thinlet*, SWT e *Swing*.

Esse trabalho possui algumas limitações como formatação de campos, onde não é possível definir máscaras para os campos, e notificação das telas *Thinlet* quando o modelo é alterado para que as telas possam ser recarregadas. Essa limitação pode ser resolvida aplicando o padrão *Observer* (GAMMA et al, 2000).

Referências

Alur, D., Crupi, J. and Malks, D. Core J2EE patterns: as melhores práticas e estratégias de design. Tradução Altair Dias Caldas de Moraes. Rio de Janeiro: Campus, 2004.

Apache Software Foundation. Commons BeanUtils. [S.l.], 2007a. Disponível em: <http://jakarta.apache.org/commons/beanutils/>. Acesso em: 14 set. 2006.

_____. HttpClient. [S.l.], 2007b. Disponível em: <http://jakarta.apache.org/commons/httpclient/>. Acesso em: 11 maio. 2007.

Bajzat, R. Thinlet. [S.l.], 2006. Disponível em: <http://thinlet.sourceforge.net/home.htmlThinlet>. Acesso em: 18 maio. 2007.

Fowler, M. Padrões de arquitetura de aplicações corporativas. Tradução Acauan Fernandes. Porto Alegre: Artmed, 2006.

Gamma, E. et al. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Tradução Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

Möbius, D. ThinG - a GUI Editor for Thinlet. [S.l.], 2007. Disponível em: <http://thing.sourceforge.net/ >. Acesso em: 15 maio. 2007.

Summa Tecnologies do Brasil. Genesis. [S.l.], 2006. Disponível em: <http://genesis.dev.java.net/nonav/3.0-EA3/maven-site/pt-BR/index.html>. Acesso em: 27 ago. 2006.

Sun Developer Network. Custom Tags in JSP Pages. [S.l.], 2007. Disponível em: <http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPTags.html>. Acesso em: 12 maio. 2007.

Yoder, J., Johnson, R. and Wilson, Q. Connecting business objects to relational database. In: CONFERENCE ON PATTERNS LANGUAGES OF PROGRAMS, 5th, 1998, Monticello. Proceedings**...** Illinois: Dept. of Computer Science, 1998. p. 9-11. Disponível em: <http://www.joeyoder.com/papers/patterns/PersistentObject/Persista.pdf>. Acesso em: 27 out. 2006.

Fundamentos para um Método Unificado para Avaliação de Processos de *Software* e mapeamento com ISO/IEC 15504 e FAA-FAM

Cristiano Schwening

Engsoft - Engenharia de *Software*
Rua Quinze de Novembro, 321/503-2 – Ijuí – RS – Brasil

IPTEC – Pólo Tecnológico do Noroeste Gaúcho
Rua Quinze de Novembro, 321/503 - Ijuí – RS - Brasil

crsch@engsoft.com.br

Abstract. *This work presents an unified method for software process assessment with the unification of ISO/IEC 15504-2 and ISO/IEC 15504-3 requirements and examplar method and FAA-FAM Method in an unified process, considering their specific characteristics and limitations. The unified assessment method allows a process assessment in an organization as a conformant process with the requirements of both original methods.*

Resumo. *Este trabalho apresenta um método unificado para a avaliação de processos de software que reúne os requisitos e método exemplo da ISO/IEC 15504-2 e ISO/IEC 15504-3 e do método FAA-FAM em um único processo, considerando suas características específicas e limitações. O método unificado de avaliação ao ser aplicado permite que uma organização realize uma avaliação em conformidade com os requisitos de ambos os métodos citados acima.*

Palavras Chaves: ISO/IEC 15504-2, ISO/IEC 15504, FAA-FAM, Avaliação de processos

1. Introdução

Cada vez mais a tecnologia da informação vem destacando-se como sendo uma das mais importantes ferramentas para auxiliar a vida de uma grande parte da população moderna. Sem notarmos, com o passar do tempo, o *software* passou a fazer parte das atividades do nosso dia-a-dia, estando hoje presente em diversos equipamentos como celulares, eletrodomésticos, automação industrial, medicina, automóveis, entre outros.

Neste contexto, a qualidade do *software* e a produtividade do processo de desenvolvimento são de suma importância para a competitividade das organizações de *software*. Porém, na maioria das vezes as organizações contam com um processo deficiente no que se refere a produtividade e qualidade.

Para CURTIS (2000), a indústria de *software* está neste novo século, preocupada em ajustar seus processos para produzir *software* de qualidade, dentro dos prazos e com orçamentos confiáveis. Logo, as organizações serão pressionadas por seus concorrentes a reduzir substancialmente os prazos para a entrega de produtos. Portanto, organizações que

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 51-61.

sejam capazes de integrar, harmonizar e acelerar seus processos de desenvolvimento e manutenção de *software* terão a primazia do mercado.

Em vista disto, a indústria de *software* mundial deu início a um amplo processo de melhoria, buscando suprir ao menos os fatores críticos relacionados com a qualidade, a produtividade e a agilidade. Neste contexto foram desenvolvidas abordagens para apoiar a melhoria dos processos de *software* (MPS), de modo a torná-las viáveis, eficazes e eficientes para apoiar iniciativas de melhoria em empresas desenvolvedoras de *software*, permitindo assim a superação dos fatores críticos o mais rápido possível.

Porém, para tornar um processo de desenvolvimento, condizente com um processo de qualidade (além de produtivo e ágil) é necessário torna-lo sistemático e passível de repetição, independente de quem o execute. Buscando atender este requisito, foram desenvolvidos diversos modelos de processo que sistematizam, representam as melhores práticas e definem uma métrica para a avaliação da capacidade dos processos. SALVIANO (2006) destaca que diversos destes modelos estão bem difundidos na comunidade e descritos em diversas publicações.

Para auxiliar na implementação destes modelos nas organizações, diversas abordagens para a melhoria de processo foram produzidas, destacando-se, a IDEAL, o Guia para Melhoria da ISO/IEC 15504 e o brasileiro PRO2PI [SALVIANO (2006)]. Estas abordagens de melhorias baseadas em modelos podem ser definidas como sendo:

> *Uma abordagem para melhoria das organizações intensivas em software, baseada em modelos de capacidade de processo, que orienta ações para alteração dos processos utilizados para aquisição, fornecimento, desenvolvimento, manutenção e/ou suporte de sistemas de software, com o objetivo de estabelecer processos que satisfaçam de forma mais eficiente e eficaz os objetivos e necessidades de negócio da organização [SALVIANO (2003)]*.

1.1. A avaliação de processos

Um aspecto importante a ser observado, além da implementação de um projeto de melhoria de processos, é realizar a avaliação dos resultados obtidos com a implantação, permitindo assim estabelecer uma base sólida quanto a evolução do processo. A literatura descreve esta avaliação de processos como sendo um exame disciplinado dos processos utilizados pela organização em relação a um modelo de referência, visando determinar a capacidade dos processos ou a maturidade de uma organização.

Mas não somente para avaliar ganhos obtidos deve-se utilizar uma avaliação de processos. Ela também pode auxiliar na implementação de um projeto de melhoria de processos de software através de uma avaliação inicial dos processos organizacionais, com a finalidade de apontar os pontos fortes e fracos que serão a base para a definição de um plano de melhoria. Portanto, conforme WANGENHEIM, *"o primeiro passo para a melhoria de processos é identificar as forças*[1] *e as fraquezas do processo de software da organização para determinar uma efetiva ação de melhoria"*.

Nesta linha, diversos autores destacam a importância da avaliação de processos, visando identificar a situação do processo de desenvolvimento de *software* da organização e a relevante contribuição para direcionar uma abordagem para a melhoria de processos. Para JONES (2000), a avaliação de processo de *software* é uma forma de abordagem que contribui para obter melhores produtos. Outro autor, ZAHRAN (1998), acrescenta que o processo de

[1] Tradução livre para a palavra em inglês: *strengths*

avaliação fornece os parâmetros básicos do estado atual das práticas predominantes de *software* numa organização. Isto deve ser utilizado como a base para a implementação de melhorias no processo.

Assim, a avaliação em geral atua como um catalisador de um plano de melhorias e no caso de um modelo de processo, como uma ferramenta para analisar e auferir um grau de capacitação ou maturidade de uma organização.

1.2. Motivação do trabalho

Este trabalho procura estudar dois métodos de avaliação disponíveis para a comunidade de *software* e que são mais usados no momento de se avaliar o nível de capacidade ou maturidade de uma organização que implementou um plano de melhoria baseado nos modelo de qualidade de *software*. O resultado do estudo destes métodos é a definição de um novo método que pretende unificar os processos estudados.

Esta motivação fundamentou-se no crescente aumento da demanda por realização de avaliações, em pelo menos dois modelos, em organizações de *software* de diversos portes no Brasil e também pela constatação da viabilidade de que um avaliador credenciado em vários modelos poderá em uma mesma oportunidade realizar auditoria em um ou mais modelos.

Esta justificativa é válida sempre ao avaliar que o custo para a realização de uma avaliação é de certa forma onerosa para a organização, principalmente quando se tratando de modelos internacionais. Assim, havendo oportunidade de em um mesmo momento realizar-se atividades de avaliação em mais de um modelo certamente tornara os custos menores e o emprego de menos tempo da equipe alocada para o processo de avaliação.

Portanto, este trabalho pretende apresentar um mapeamento das atividades definidas nos principais modelos de avaliação disponíveis e por fim demonstrar um método unificado que possa contemplar todas as atividades que serão necessárias para que um avaliador capacitado possa proceder mais de uma avaliação em um programa de melhoria de processos de *software* da organização.

2. Aspectos Conceituais

Para uma análise mais criteriosa dos métodos estudados é necessário antes apresentar alguns conceitos relacionados com a avaliação de processos de *software*. O primeiro conceito a ser estabelecido é a própria palavra avaliação.

Em seus trabalhos, HUMPHREY (1989) escreve que uma avaliação de processo de *software* não é uma auditoria, mas uma revisão da organização de *software* que visa recomendar à gerência e a seus profissionais ações de melhoria da operação. Desta forma, neste trabalho se adotará esta mesma definição de avaliação, que em inglês significa *assessment*.

2.1. Tipos de avaliação

A norma ISO/IEC 15504-2 (2003), entre outras referências classifica as avaliações, em três tipos, dependendo de quem executa o papel principal na avaliação:

- Auto avaliação: avaliação executada internamente à organização de *software*, em geral pela própria equipe. O objetivo principal é identificar a capacitação do próprio processo de *software* da organização e iniciar um plano de ação para melhoria de processo.

- Avaliação de segunda parte: executada por avaliadores externos. O objetivo principal é avaliar a capacitação da organização para atender os requisitos especificados em contrato e geralmente feita pelo cliente para avaliação do seu fornecedor.
- Avaliação de terceira parte: é executada por uma organização independente de terceira parte. O objetivo principal é avaliar a habilidade da organização em responder por contratos ou produzir produtos de *software*.

Além desta classificação, ZAHRAN (1998) propõem que estes tipos de avaliações ainda possam ser classificados em auto avaliações e avaliações independentes, ou como avaliações internas e externas, conforme PERSSE (2001). Desta forma, as auto avaliações de ZAHRAN (1998) ou avaliações internas de PERSSE (2001), são executadas pela própria organização, com seus próprios recursos e métodos para obter uma fotografia do processo de desenvolvimento. Já as avaliações independentes de ZAHRAN (1998) ou externas, conforme PERSSE (2001) são conduzidas por avaliadores independentes da unidade organizacional sendo avaliada, e são assim chamadas de avaliações de segunda ou de terceira parte.

2.3. Processo de avaliação

Toda avaliação deve ser realizada por um avaliador capacitado e guiado por um processo de avaliação composto de um plano de execução que descreve os passos a serem realizados durante o desempenho das atividades. O processo de avaliação deverá ao seu final apontar à organização um nível de capacidade ou maturidade de seus processos.

Conforme SALVIANO (2006), dentro do contexto de melhoria de processos, a avaliação significa a caracterização das práticas correntes de uma unidade organizacional em termos de capacidade dos processos selecionados. Os resultados são analisados em relação às necessidades de negócio da organização, identificando os aspectos positivos e negativos, e os riscos associados aos processos.

Ao ser conduzida uma avaliação baseada em um processo documentado, esta deve ser capaz de atender a diversos propósitos. Conforme a ISO/IEC 15504, estes propósitos devem cobrir no mínimo as atividades de planejamento, coleta de dados, validação destes dados, pontuação dos atributos de processos (notas) e a representação dos resultados. Na visão de HUMPHREY (1989), uma avaliação deve contemplar ao menos três fases:

- A preparação da avaliação inicia com a identificação da organização a ser avaliada, da equipe que a realizará e a obtenção do comprometimento da gerência para a realização da avaliação.
- A avaliação subdivide nas etapas de formação e treinamento da equipe, preparação do cronograma e realização da avaliação na organização.
- Para as recomendações, os passos previstos são: preparação do relatório com as conclusões, preparação do plano de ação, as reavaliações de acompanhamento, fornecimento de futuros marcos do plano e o estabelecimento de novas prioridades para a melhoria contínua.

2.4. Norma ISO/IEC 15504-2

De acordo com SALVIANO (2006), o termo "ISO[2]/IEC[3] 15504" designa a Norma Internacional ISO/IEC 15504:2003 para Avaliação de Processos desenvolvida pela ISO/IEC

[2] ISO é a sigla da *International Organization for Standardization* (Organização Internacional para a Normalização).

(através do grupo de trabalho ISO/IEC SC1 JTC1 WG10) e com o apoio do projeto SPICE (*Software Process Improvement and Capability dEtermination*).

A ISO/IEC 15504 define um *framework* para modelos de avaliação de processo. Na prática, a norma também pode ser utilizada como referência para a melhoria de processo. A ISO/IEC 15504 é composta por 5 partes conforme a tabela 1.

Parte	Descrição
1	Conceitos e Vocabulário.
2	Executando uma Avaliação (normativo).
3	Guia sobre como executar uma avaliação.
4	Guia sobre como utilizar os resultados de uma avaliação.
5	Exemplo de modelo de avaliação de processo.

Tabela 1: Partes da ISO/IEC 15504

Para a ISO/IEC 15504-2, uma avaliação deve ser conduzida de acordo com um processo de avaliação documentado capaz de atender aos propósitos da avaliação. O processo documentado de avaliação tem que conter no mínimo as atividades de:

1. Planejamento;
2. Coleta de dados;
3. Validação dos dados;
4. Pontuação dos atributos de processo;
5. Representação dos resultados

A parte 3 da ISO/IEC 15504-3:2004 é um guia (não normativo) para a interpretação dos requisitos mínimos para executar uma avaliação baseada na norma ISO/IEC 15504-2. Ele apresenta uma visão geral de um processo de avaliação e interpreta os requisitos apresentando os seguintes itens:

a) A execução de uma avaliação;
b) Um *framework* para medição da capacidade do processo;
c) Um modelo de referência de processo e modelos de avaliação de processo;
d) A seleção e uso de ferramentas de avaliação;
e) A competência da equipe de avaliação; e
f) A verificação da conformidade.

2.5. O FAM

O FAM[4] é o método de avaliação desenvolvido pela Administração Federal de Aviação dos Estados Unidos (*Federal Aviation Administration* - FAA) desde 1999, para ser utilizado em avaliações no modelo FAA iCMM (Integrated Capability Maturity Model, Modelo Integrado da Maturidade da Capacidade) [IBRAHIM (1999)]. A versão, utilizada neste trabalho, do FAM é a 1.0 que foi publicada em 1999 e integra o ISO/IEC 15504 e o método SCAMPI.

Ele tem por finalidade:

---

[3] IEC é a sigla da International Electrochnical Commision (Comissão Internacional de Eletroeletrônica).

[4] FAA-iCMM Appraisal Method

- Motivar, direcionar e iniciar uma melhoria em uma organização ou projeto;
- Diagnosticar e determinar o status de uma melhoria comparado com um modelo ou padrão, ou para acompanhar o progresso;
- Estabelecer uma linha base ou uma avaliação das atuais práticas, obtidas através de uma auto-melhoria.

O método pode ser adaptado às necessidades do projeto ou organização através de adequações que podem ser consultadas no documento do modelo.

O processo de avaliação definido pelo método FAM é composto de três fases:

1. Planejar e preparar para a avaliação;
2. Conduzir a avaliação;
3. Reportar resultados.

3. Desenvolvimento do Tema

3.1. Análise dos métodos

Inicialmente realizou-se uma revisão bibliográfica sobre avaliação de processo, engenharia de processos, sobre os métodos de avaliação pesquisados e seus modelos através de consultas aos documentos oficiais dos métodos disponibilizados, além da leitura de teses, dissertações e artigos científicos que abordassem temas importantes para a elaboração deste trabalho. A partir deste estudo foram identificados os principais processos de avaliação dos métodos e posteriormente realizou-se um levantamento de todas as atividades e tarefas de cada método, com o propósito de compreender claramente estas atividades.

A análise dos métodos permitiu a geração de duas tabelas. Cada tabela apresenta uma lista ordenada das atividades e das subatividades de cada método. Estas tabelas posteriormente foram utilizadas como base de consulta para a realização do mapeamento das atividades dos processos de avaliação.

3.2. Mapeamento dos métodos

Inicialmente, para definir o método unificado foi realizado um minucioso mapeamento que objetivou relacionar as atividades de cada método com a sua correspondente no outro método. Esta análise criteriosa buscou interpretar a descrição e as observações apresentadas no método, para cada uma das atividades, sempre garantindo a real correlação entre ambos os métodos.

O mapeamento foi elaborado com base no método FAM que é utilizado para a obtenção de nível no modelo iCMM e na parte 3 da norma ISO/IEC 15504, um guia para a interpretação dos requisitos mínimos para executar uma avaliação baseada na norma ISO/IEC 15504-2.

Para produzir o mapeamento foram relacionadas todas as atividades do método FAM que estivessem referenciando uma atividade do ISO/IEC 15504-3. O resultado desse relacionamento indicou uma grande correlação entre as atividades dos dois métodos e os requisitos, o que viabilizou a construção do método unificado.

Ao analisar o mapeamento nota-se um alto grau de relacionamento entre as atividades dos métodos mapeados e, portanto, basicamente, verifica-se que todas as atividades definidas na ISO/IEC 15504-3 são apresentadas no FAM, através de uma ou um mais atividades

relacionadas. Além disso, o FAM foi produzido para suportar diversas atividades presentes no método SCAMPI[5].

Com relação ao grau de subdivisão dos processos, pode-se constatar em ambos os métodos isto é muito semelhantes quando analisados pelo ângulo da ISO/IEC 15504-3. A maior quantidade de subprocessos e atividades encontrados no FAM, deve-se ao motivo de que a norma ISO/IEC 15504-3 é apenas um guia exemplar de um processo que atende aos requisitos da norma ISO/IEC 15504-3.

3.3. O Método Unificado

O método unificado é composto de um conjunto de processos e atividades que incorpora o método FAM e a parte 3 da ISO/IEC 15504. Ele se adapta a ambos os métodos permitindo assim a estruturação de um processo único que atende todos os requisitos obrigatórios para que um avaliador possa executar em um mesmo período uma avaliação para obtenção de um nível no iCMM ou em algum método construído a partir da norma ISO/IEC 15504, além disso, ele considera as características específicas e as limitações de cada um dos métodos.

Como visto, os métodos aqui estudados, não possuem grandes diferenças na maioria das suas atividades, executando geralmente as mesmas tarefas apenas com alguma diferenciação com relação a nomenclatura ou sua granularidade. A granularidade é o principal destaque que difere os métodos, desta forma observa-se que o FAM é composto de um conjunto mais detalhado de atividades comparando-se com a norma ISO/IEC 15504-3. O principal motivo para esta menor granulidade do método exemplo da norma ISO/IEC 15504 é em virtude deste não constituir-se em um método oficial de avaliação e sim um modelo mínimo que atenda a parte 2 da mesma norma. Salienta-se, porém esta particularidade destacada não descredencia o método unificado, apenas o enriquece em atividades.

Uma necessidade constatada, após o mapeamento e a definição do método (tabela 2) foi a elaboração de um guia de adequação, que permita aplicar as atividades definidas para o método unificado no contexto do modelo de referência específico. Desta forma, quando uma atividade definida no método unificado referenciar mais de uma no método escolhido este guia permitirá ao avaliador saber quais atividades estão sendo executadas naquele momento.

Com relação a atribuição de níveis de capacidade ou maturidade dos processos, o FAM permite apenas a atribuição por níveis de maturidade, diferente da norma ISO que permite apenas a atribuição de níveis de capacidade. Esta situação também deverá ser especificada no guia de adequação para permitir que o avaliador possa definir o tipo de atribuição de níveis conforme o modelo de referência utilizado

3.3.1. As fases do Método Unificado

O método unificado, apresentado na tabela 2, é composto de fases e atividades que cumprem os requisitos para um processo de avaliação e tem por objetivo permitir que o avaliador possa desempenhar as atribuições necessárias para a determinação de um nível de maturidade ou capacidade em uma organização em mais de um modelo de qualidade. O método proposto é dividido em três fases principais:

1. Preparar para a realização da avaliação;
2. Realizar a avaliação;

[5] Standard CMMI Appraisal Method for Process Improvement – Classe A

3. Documentar os resultados da avaliação.

Na seqüência são descritas as principais características das três fases que compõem o método unificado de avaliação proposto neste trabalho.

Preparar para a realização da avaliação

A fase de Preparar para a realização da avaliação tem a finalidade de planejar uma avaliação, preparando toda a documentação necessária, definição do patrocinador, a seleção e preparação da equipe de avaliação (treinada e orientada), obtenção de dados preliminares que possam diagnosticar se a organização está preparada para receber uma avaliação e os recursos necessários. A organização também deverá estar preparada para as atividades de avaliação.

Realizar a avaliação

A fase de Realizar a Avaliação tem o propósito de conduzir uma avaliação no modelo desejado e por fim comunicar seus resultados à empresa avaliada. Nesta fase, a equipe de avaliadores tem o foco direcionado à coleta e validação de dados, com o propósito de avaliar a extensão em que o modelo desejado é executado, permitindo assim contar com dados suficientes e colher uma amostra representativa dos processos.

Documentar os resultados da avaliação

O propósito da fase de Documentar os resultados da avaliação é de elaborar um relatório da avaliação que será entregue ao patrocinador da avaliação e também encaminhado à instituição responsável pelo método que insere os resultados da avaliação em sua base de dados e divulga o resultado, caso seja possível. Nesta fase, a equipe de avaliação fornece as considerações (lições aprendidas) na avaliação e o resultado ao patrocinador.

A tabela 2 busca apresentar todas as atividades que compõem as três fases propostas para o método unificado. Neste momento foram preservadas as descrições em inglês das atividades do método, pois alguns termos ao serem traduzidos poderiam modificar o seu significado original.

1. Preparar para a realização da avaliação	2. Realizar a avaliação	3. Documentar os resultados da avaliação
1.1 Obtain sponson commitment	2.1 Conduct Opening meeting	3.1 Prepare and deliver appraisal report
1.1.1 Understand the customer's problem	2.1.1 Invite sponsor comments	3.1.1 Review final briefing
1.1.2 Propose appraisal solution	2.1.2 Present appraisal process	3.1.2 Plan the preparation of the report
1.1.3 Agree on basics	2.1.3 Review schedule	3.1.3 Write the report
1.1.4 Record commitments	2.1.4 Invite and answer questions	3.1.4 Review/revise the report
1.2 Select appraisal scope	2.2 Conduct Interviews	
1.2.1 Administer project and/or organization questionnaire	2.2.1 Introduce participants	
1.2.2 Identify appraisal scope	2.2.2 Describe technique	
1.2.3 Identify organization scope	2.2.3 Ask and answer questions	
1.2.4 Make preliminary	2.2.4 Record requests	

arrangements	
1.2.5 Document preliminary plan	2.2.5 Close session
1.3 Select Appraisal Team	2.3 Review documentation
1.3.1 Appraisal advocate and/or sponsor and/or organizational appraisal representative	2.3.1 Review documentation
1.3.2 Appraisal team leader	2.3.2 Identify additional documents
1.3.3 Site coordinator	2.3.3 Collect and manage document
1.3.4 Introduce to method	2.4 Consolidate data
	2.4.1 Review notes
	2.4.2 Record observations
	2.4.3 Identify missing information
	2.4.4 Check for accuracy
	2.4.5 Check for validity
	2.4.6 Check for sufficiency
	2.5 Develop draft findings
	2.5.1 Develop draft findings
	2.5.2 Priorize findings
	2.5.3 Form consensus
	2.5.4 Prepare Briefing charts
	2.6 Present Draft findings
	2.6.1 Describe purpose of session
	2.6.2 Present finding
	2.6.3 Validate findings
	2.6.4 Adjourn meeting
	2.6.5 Review suggested change to findings
	2.7 Develop ratings
	2.7.1 Determine classification of Process goal
	2.7.2 Determine classification of Process Implementation
	2.7.3 Determine classification of capability level goals for Process
	2.7.4 Determine institutionalization classification (Capability level) for Process
	2.7.5 Check for consistency
	2.7.6 Develop process capability profile
	2.7.7 Determine maturity rating
	2.8 Develop final briefing
	2.8.1 Prepare final briefing
	2.8.2 Ensure consistency of final briefing

2.8.3 Produce final briefing
2.9 Brief sponsor
2.9.1 Brief sponsor
2.9.2 Ask for feedback
2.9.3 Record Feedback
2.9.4 Discuss next steps
2.10 Present final briefing
2.10.1 Present final briefing
2.10.2 Conduct open discussion
2.10.3 Thanks sponsor and participants
2.11 Conduct wrap-up
2.11.1 Collect lesion learned
2.11.2 Review comments
2.11.3 Answer last-minute questions
2.11.4 Return material and clean up
2.9.3 Record Feedback
2.9.4 Discuss next steps
2.10 Present final briefing
2.10.1 Present final briefing
2.10.2 Conduct open discussion
2.10.3 Thanks sponsor and participants
2.11 Conduct wrap-up
2.11.1 Collect lesion learned
2.11.2 Review comments
2.11.3 Answer last-minute questions
2.11.4 Return material and clean up
2.10.2 Conduct open discussion
2.10.3 Thanks sponsor and participants
2.11 Conduct wrap-up
2.11.1 Collect lesion learned
2.11.2 Review comments
2.11.3 Answer last-minute questions
2.11.4 Return material and clean up

Tabela 2 - O método unificado

Quanto a compatibilidade na atribuição de níveis, constatou-se que será necessário a definição de um item específico no guia de adequação, visto que o formato utilizado para a atribuição dos níveis nos dois métodos estudados é diferente. Acredita-se, porém que a próxima versão do FAA-FAM, deverá suprir esta deficiência atual.

4. Conclusão

Neste trabalho é apresentado um método unificado que tem por objetivo descrever as atividades necessárias para a realização de uma avaliação dos processos de desenvolvimento

de software. Além disso, o seu principal foco é permitir que um avaliador possa realizar mais de uma avaliação em organizações em uma mesma oportunidade, permitindo assim a obtenção de um nível de maturidade ou de capacidade desejado e também a redução de custos, permitindo que a avaliação esteja adaptada ao contexto da maioria das empresas de software brasileira.

Este método unificado foi obtido através de um mapeamento dos processos e atividades dos métodos de avaliação FAA-FAM e ISO/IEC 15504-3. Como resultado deste mapeamento gerou-se um conjunto de atividades que quando aplicadas satisfazem as exigências e particularidades de cada um dos métodos, inclusive sugerindo um guia de adequação que permitirá que a atribuição de níveis seja orientada conforme o modelo escolhido (estagiado ou contínuo). O autor pensa que este método unificado pode ainda agregar mapeamentos para outros métodos de avaliação conhecidos pela comunidade, como o MARES [WANGENHEIM (2006)] e o QUICKLOCUS [KOHAN (2003)], por exemplo.

Analisando o método apresentado percebe-se que ele pode ser aplicado por avaliadores credenciados nos métodos FAA-FAM e em métodos construídos com base na ISO/IEC 15504-2 em organizações de qualquer porte, portanto conclui-se que o trabalho proposto alcançou o seu objetivo de apresentar uma solução de avaliação compatível com mais de um método.

Como próximos passos deste trabalho, sugere-se desenvolver algumas ações visando enriquecer a descrição, validar, automatizar e definir um guia de adequação para o método unificado:

- Realizar um detalhamento maior dos processos e atividades do método unificado nos moldes do FAA-FAM, inclusive definindo a tradução correta para as atividades;
- Validar o método unificado, com sua aplicação em algumas avaliações pilotos em organizações reais;
- Projetar e desenvolver uma ferramenta de apoio que possa auxiliar o avaliador na definição automática das atividades a serem realizadas pela avaliação, inclusive com a possibilidade de apoiar a coleta de indicadores diretos e indiretos e anotações do avaliador;
- Elaborar o guia de adequação detalhando as atividades do método no contexto do modelo de referência específico.

5. Referências Bibliográficas

CHRISSIS, M. B., KONRAD, M., SHRUM, S. ***CMMI: Guidelines*** for ***Process Integration and Product Improvement***, *Addison-Wesley*, 2004.

CURTIS, B. ***The global pursuit of process maturity.*** IEEE Software, jul./ago. 2000.

HUMPHREY, W.S. ***Managing the Software Process.*** Estados Unidos: Addison- Wesley Publishing Company. 1989.

IBRAHIM, Linda, Bill Bradford, David Cole, Larry LaBruyere, Heidi Leinneweber, Dave Piszczek, Natalie Reed, Mike Rymond, Dennis Smith, Michael Virga and Curt Wells, ***The Federal Aviation Administration Integrated Capability Maturity Model® (FAA-iCMM®) Appraisal Method (FAM)***, Version 1.0, Published by the Federal Aviation Administration, 1999. (Disponível em http://www.faa.gov/about/office%5Forg/headquarters%5Foffices/aio/documents/iCMM/media/faafamv1.pdf e acessado em 26/10/2006).

ISO/IEC 15504-2. ISO/IEC JTC1/SC7 SOFTWARE & SYSTEM ENGINEERING SECRETARIAT. ***FDIS 15504-2 – Software Engineering – Process Assessment – Part 2: Performing an Assessment.*** Canada: ISO/IEC JTC1/SC7 Secretariat: ANSI. 2002.

ISO/IEC 15504-3. ISO/IEC JTC1/SC7 SOFTWARE & SYSTEM ENGINEERING SECRETARIAT. ***FDIS 15504-3 – System and Software Engineering – Process Assessment – Part 3: Guidance on Performing an Assessment.*** Canada: ISO/IEC JTC/SC7 Secretariat: ANSI. 2003.

JONES, C. ***Software Assessments, Benchmarks, and Best Practices.*** Estados Unidos: Addison-Wesley. 2000.

ZAHRAN, S. ***Software Process Improvement.*** Estados Unidos: Addison-Wesley. 1998.

KOHAN, S. **QuickLocus: proposta de um método de avaliação de processo de desenvolvimento de software em pequenas organizações.** Trabalho Final de Mestrado Profissional em Engenharia da Computação, Instituto de Pesquisas Tecnológicas do Estado de São Paulo. 2003.

MEMBERS OF THE ASSESSMENT METHOD INTEGRATED TEAM. ***Standard CMMI Appraisal Method for Process Improvement (SCAMPI), Version 1.1 Method Definition Document***. Estados Unidos: Carnegie Mellon University – Software Engineering Institute. 2001. (Disponível em www.sei.cmu.edu/pub/documents/01.reports/pdf/01hb001.pdf e acessado em 26/10/2006).

PESSÔA, M.S.P. ***Introdução ao CMMI – Modelo Integrado de Maturidade da Capacidade de Processo***, publicação do curso de pós-graduação "Lato Sensu" / (Especialização) a distância em melhoria de processo de software, UFLA/FAEPE. 2005.

PERSSE, J. R. ***Implementing the Capability Maturity Model***. Estados Unidos: John Wiley & Sons, Inc. 2001.

PRESSMAN, R. S. **Engenharia de Software.** 5ed. Rio de Janeiro: McGraw-Hill, 2002.

SALVIANO, C.F. **Uma proposta orientada a perfis de capacidade de processo para evolução da melhoria de processo de software.** Tese de doutorado pela Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. 2006.

SALVIANO, C.F. **Melhoria e Avaliação de Processo com ISO/IEC 15504 (SPICE) e CMMI**, publicação do curso de pós-graduação "Lato Sensu" / (Especialização) a distância em melhoria d de processo de software, UFLA/FAEPE. 2003.

SCHWENING, C., SALVIANO, C. **Fundamentos para um método unificado para avaliação de processos de software e mapeamento com SCAMPI e MA-MPS**, Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Lato Sensu Modelo de Maturidade e Capacidade do Processo com CMMI e MPS.BR, para obtenção do título de especialização, 2006.

WANGENHEIM, V., ANACLETO, A., SALVIANO, C.F. ***Helping Small Companies Assess Software Processes.*** IEEE Software, jan./fev. 2006.

WEBER, K., ARAUJO E., MACHADO, C.A.F., SCALET D., SALVIANO C. F., e ROCHA, A. R. C. **Método de Avaliação para Melhoria de Processo de Software - versão 1.0 (MA-MPS)**, Brasil: Softex. 2005. (Disponível em www.softex.br/portal/mpsbr/_guias/MPS.BR_GUIA_AVALIACAO.pdf e acessado em 26/10/2006).

Testware: ferramenta de planejamento e execução de casos de teste

Fabiane Barreto Vavassori Benitti[1, 2], Ana Paula Zimmermann[1]

[1]Centro de Ciências Tecnológico da Terra e do Mar
Universidade do Vale do Itajaí, (UNIVALI) – Itajaí, SC – Brazil

[2]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

fabiane.benitti@univali.br, anapzi@gmail.com

Resumo. *Teste de software é um dos processos do ciclo de desenvolvimento do software que possui o objetivo de exercitar um programa a fim de descobrir erros, contribuindo para a qualidade do produto final. Por ser um processo complexo consome bastante recursos. Sendo assim, as ferramentas para automação de teste de software, permitem reduzir o tempo gasto e conseqüentemente os custos para a organização que as incorpora. Testware é uma ferramenta para automação de testes de caixa preta, que visa auxiliar na definição de casos de teste e sua execução.*

1. Introdução

Teste é uma das áreas da Engenharia de Software e uma das etapas do ciclo de desenvolvimento de software, sendo definida por Pressman (2005) como o processo de executar um programa com a intenção de descobrir um erro. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto. O teste auxilia na produtividade e fornece evidências da confiabilidade e da qualidade do software. Uma organização gasta 40% do esforço do projeto total somente na etapa de teste, e se os defeitos forem descobertos na fase de manutenção, o custo por correção pode ser de 60 a 100 vezes maior [Pressman 2005].

Assim, surgem no mercado ferramentas para automatizar as diversas atividades inerentes a um processo de testes de software. Segundo Mats (2001) um dos principais problemas nas atividades de testes de software é a ausência de critérios para seleção dos casos de teste, definição da sua completude e estabelecimento de um ponto de parada, dificultando a revelação de falhas no produto. A ferramenta Testware, apresentada neste artigo, tem seu foco de atuação no problema relatado por Mats, auxiliando na definição e automação de casos de teste baseada nos conceitos da técnica de caixa preta que, segundo Bartié (2002), concentram-se nos requisitos funcionais do sistema.

Sendo assim, este artigo está organizado da seguinte forma: na seção 2 são apresentados os conceitos que embasam a ferramenta. Na seção 3 consta a especificação da ferramenta e descrição de sua funcionalidade. A seção 4 aborda ferramentas de automação de teste visando uma análise comparativa. Por fim, a seção 5 expõe as considerações finais e melhorias previstas para a ferramenta.

2. Teste de Software: caixa preta

Segundo Bartié (2002), os testes "têm por objetivo identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo".

Desta forma, tem-se duas abordagens para os testes: testes de caixa branca e os testes de caixa preta.

Os testes de caixa branca são considerados “testes em pequeno porte” (*testing in the small*), ou seja, são tipicamente aplicados a componentes de programa pequenos (por exemplo, uma classe ou um método). Os testes de caixa preta, por outro lado, ampliam o foco e poderiam ser denominados “testes em grande porte” (*testing in the large*) [Pressman 2005]. Utilizando técnicas de caixa branca pode-se derivar casos de teste que: (1) garantam que todos os caminhos do código tenham sido exercitados ao menos uma vez; (2) exercitam todas as condições lógicas, verdadeiro e falso; (3) exercitam os ciclos nos seus limites e dentro de seus intervalos operacionais; e (4) exercitam as estruturas de dados internas para garantir sua validade. Os métodos mais conhecidos para a abordagem de caixa branca são: Teste de Caminho Básico e Teste de Estrutura de Controle.

Os testes de caixa preta são projetados para validar os requisitos funcionais, sem se preocupar com o funcionamento interno de um programa. As técnicas de teste de caixa preta concentram-se no domínio de informações do software, derivando os casos de teste ao dividir a entrada e a saída de uma maneira que proporcione uma satisfatória cobertura de teste. Existem quatro principais métodos utilizados para detectar casos de teste de caixa preta, que procuram aumentar a cobertura dos testes: particionamento por equivalência, análise de valor limite, técnicas de grafo de causa efeito e testes de comparação. A ferramenta Testware aborda as duas primeiras técnicas.

“A partição de equivalência é um método que divide o domínio de entrada de dados em classes (grupos de valores). Cada classe representa um possível erro a ser identificado, permitindo que os casos de testes redundantes de cada classe identificada sejam eliminados sem que a cobertura dos cenários existentes seja prejudicada.” [Bartié 2002].

As classes devem ser definidas de acordo com as seguintes diretrizes:

1. Se uma condição de entrada especifica um intervalo, uma classe de equivalência válida e duas inválidas são definidas;

2. Se uma condição de entrada exige um valor específico, uma classe de equivalência válida e duas inválidas são definidas;

3. Se uma condição de entrada especifica um membro de um conjunto, uma classe de equivalência válida e uma inválida são definidas; e

4. Se uma condição de entrada é booleana, uma classe de equivalência válida e uma inválida são definidas.

A análise de valor limite complementa o método de particionamento por equivalência, porém foca os casos de testes nas fronteiras de cada classe, por exemplo, o software de digitação das médias finais dos alunos de graduação, o campo nota permite a entrada de valores na seguinte faixa “x >= 0 and x <=10”. Desta forma, este método sugere os casos de teste com o valor –1, 11, 0 e 10.

Este princípio foi criado, pois um grande número de erros é encontrado nas fronteiras do domínio de entrada. Além de focalizar somente as condições de entrada, a

análise de valor limite também deriva casos de teste para o domínio de saída, apresentando as seguintes diretrizes para a criação de casos de teste:

1. Se uma condição de entrada especifica um intervalo limitado pelos valores a e b, devem-se criar casos de teste com valores imediatamente acima e abaixo de a e b;

2. Se uma condição de entrada especifica vários valores, casos de teste deverão ser realizados para exercitar os números mínimos e máximos permitidos, também valores imediatamente acima e abaixo do mínimo e máximo selecionado;

3. As diretrizes 1 e 2 também são aplicadas às condições de saída. Por exemplo, uma tabela de temperatura *versus* pressão é esperada como saída de um programa de análise de engenharia. Devem ser projetados casos de testes para criar um relatório de saída que produza o número máximo e mínimo aceitável de entradas na tabela; e

4. Se o programa contém uma estrutura interna que existem limites prescritos, deve realizar testes a fim de exercitar esta estrutura de dados no seu limite, por exemplo, um vetor de 100 entradas.

3. Especificação e Apresentação da Ferramenta Testware

Para a execução de testes de software com a ferramenta Testware é necessário seguir as etapas ilustradas na Figura 1. Na etapa 1 é preciso configurar a ferramenta descrevendo o roteiro de execução da aplicação em teste. Na etapa 2, de acordo com as diretrizes dos métodos de caixa preta (particionamento por equivalência e análise de valor limite), o testador fornece os valores que a ferramenta irá utilizar para testar a aplicação. Após, na etapa 3, através do roteiro de execução e dos valores definidos anteriormente a ferramenta gera os casos de teste. Na etapa 4 a ferramenta executa automaticamente a aplicação em teste. E, por fim, o testador informa o resultado do teste executado, ficando os casos de teste disponíveis na ferramenta para reprodução.

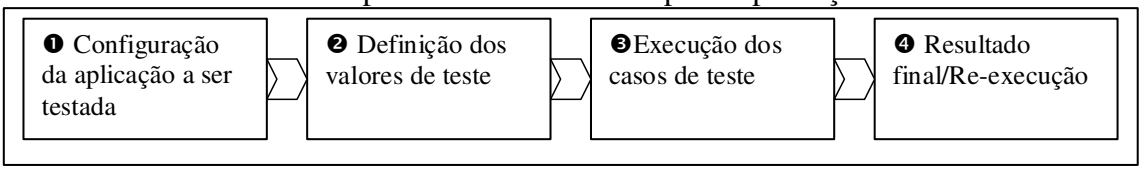


Figura 1. Etapas da ferramenta

Considerando o processo descrito, as principais funcionalidades da ferramenta desenvolvida são demonstradas através do diagrama de casos de uso ilustrado na Figura 2.

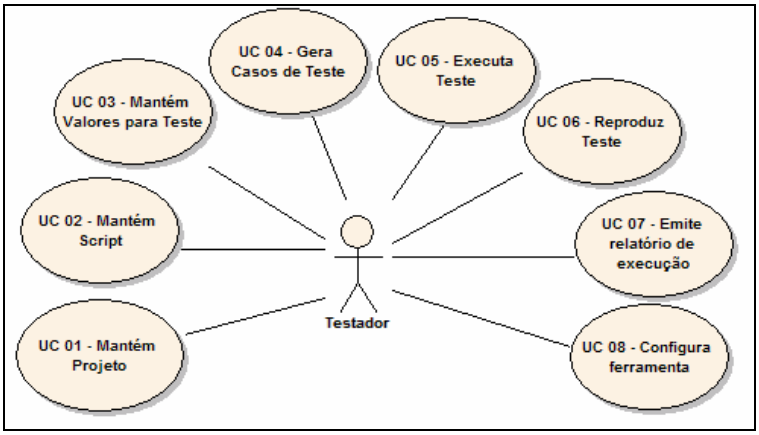


Figura 2. Modelo de Casos de Uso

Sucintamente, os casos de uso possuem os seguintes comportamentos:

- ✓ O caso de uso 01 refere-se ao cadastramento do projeto testado, mantendo informações como o nome do projeto e indicação do caminho da aplicação a testar (*path* do executável).
- ✓ No caso de uso 02 é onde ocorre a edição e a validação do *script*. O *script* é o roteiro de execução da aplicação em teste, e possui uma sintaxe própria (conforme detalhado na seção 3.1). A Figura 3a, apresenta um exemplo de *script*.
- ✓ O caso de uso 03 prevê que o testador poderá informar valores de teste para cada campo de entrada definido no *script*. Conforme o tipo do campo: valor, intervalo, conjunto ou booleano - são exibidas dicas, instruindo o testador a cadastrar valores de teste válidos e inválidos, conforme as diretrizes dos métodos particionamento por equivalência e análise de valor limite (conforme pode ser observado na Figura 3b).

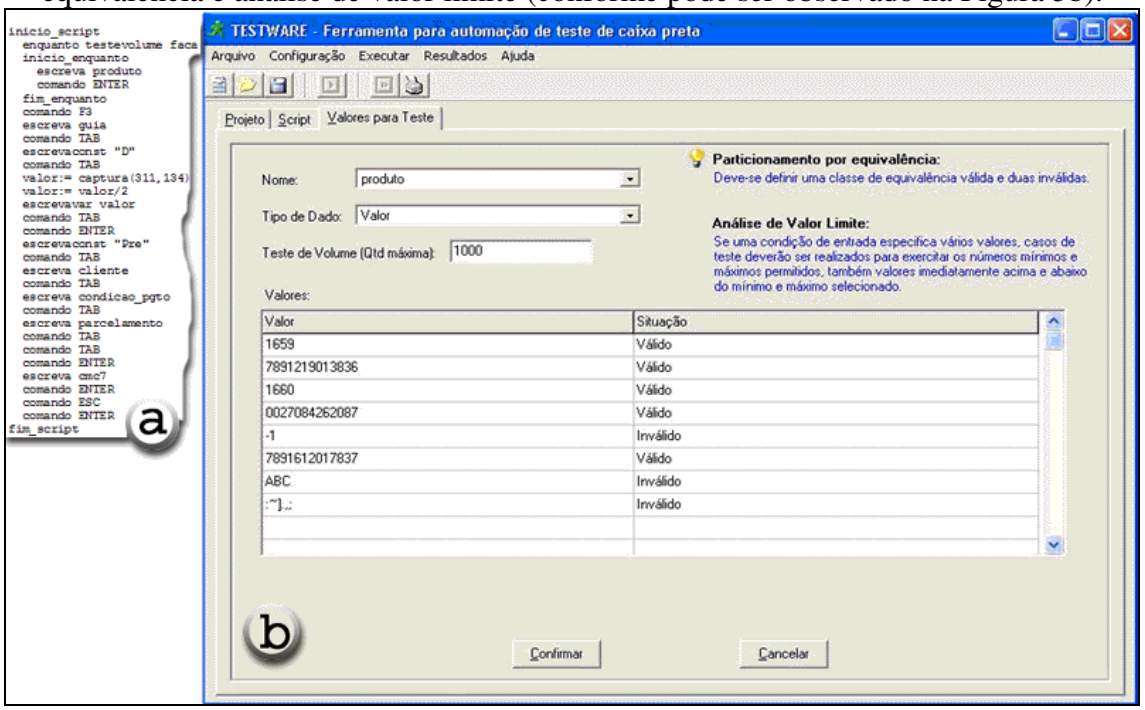


Figura 3. Interface para cadastro de valores de teste por campo

Além disso, os aplicativos testados que possuam campos iterativos (por exemplo, funcionalidade de vendas, na qual se pode relacionar vários produtos em uma venda) poderá ser aplicado o teste de volume. Conforme Bartié (2002), o teste de volume determina o limite de processamento de um aplicativo através do incremento do volume de operações. Para isto, os campos iterativos devem ser identificados no *script* e também informados o valor máximo de iterações a ser aplicado (campo "Teste de volume" na interface da Figura 3b). Desta forma, será criado um caso de teste separadamente para aplicar o teste de volume.

- ✓ O caso de uso 04, através do *script* e dos valores cadastrados para teste, realiza a geração automática dos casos de teste. Os casos são gerados pela ferramenta observando a seguinte regra: no primeiro caso de teste em cada campo de entrada será inserido um valor válido, após serão criados casos de testes visando exercitar as opções inválidas de cada campo, sendo para os demais campos utilizados sempre valores válidos. A ferramenta irá gerar casos de teste para todos os valores

cadastrados, sempre exercitando um campo de cada vez e para os demais aplicando valores válidos.

- ✓ O caso de uso 05 descreve a possibilidade de execução dos casos de testes permitindo ao testador acompanhar a execução e, ao finalizar a execução, informar o resultado do teste. A Figura 4 ilustra a interface para este procedimento.

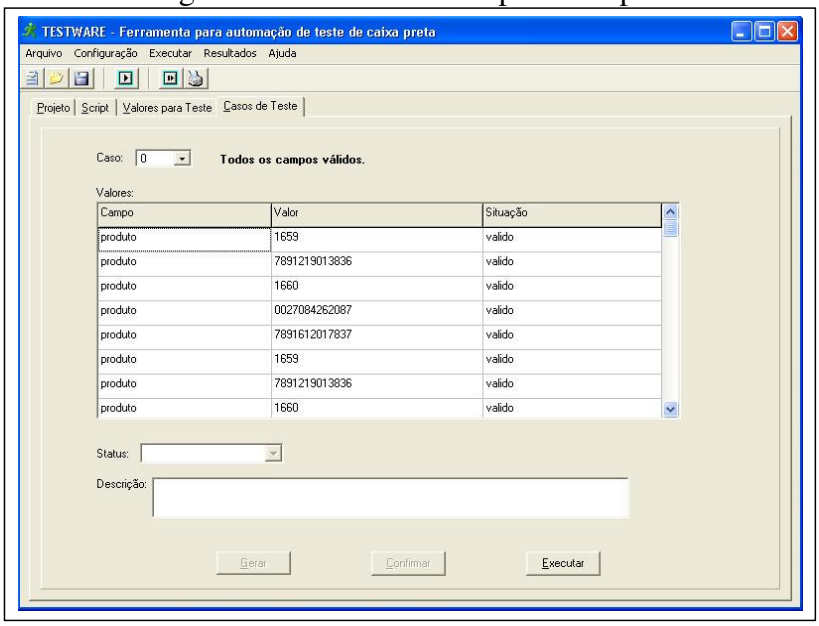


Figura 4. Interface para execução de um caso de teste

- ✓ O caso de uso 06 apresentará o resultado dos testes realizados, permitindo ao testador, a qualquer momento, reproduzir um caso de teste executado anteriormente (visando abranger os testes de regressão[1]). A Figura 5 retrata a interface de acompanhamento e os resultados possíveis de um caso de teste:
    - Sucesso: o caso de teste foi concluído com sucesso, considerando valores válidos para todos os campos de entrada; ou o caso de teste não foi concluído, pois se utilizou valores inválidos em algum campo.
    - Erro: o caso de teste foi concluído, tendo utilizado valor inválido; ou o caso de teste não concluiu, utilizando todos os valores válidos.
    - *Timeout*: a aplicação travou.

[1] "O teste de regressão é a reexecução de algum subconjunto de testes que já foram conduzidos para garantir que as modificações não propagaram efeitos colaterais indesejáveis" [Pressman 2005].

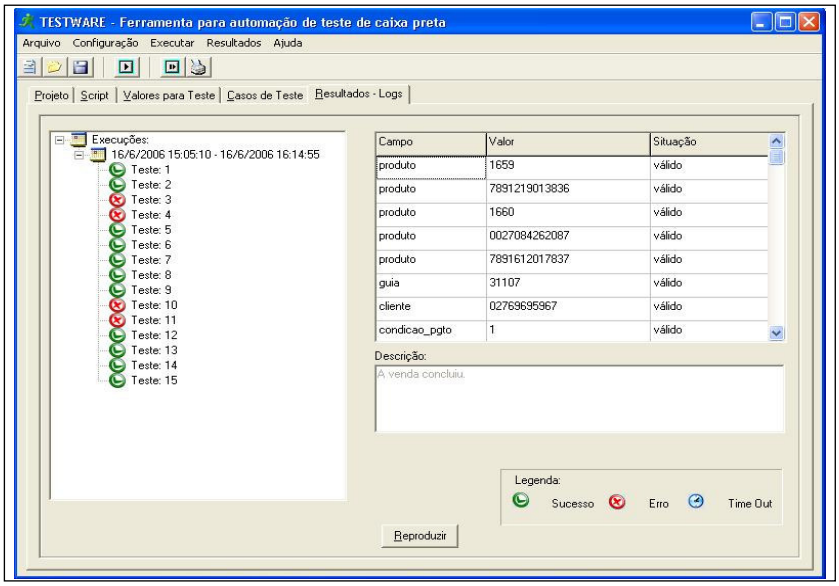


Figura 5. Interface de acompanhamento dos testes executados

- ✓ Através do caso de uso 07 é emitido um relatório de execução informando os casos de teste e o seu resultado. Este relatório deverá ser repassado aos programadores para efetuar correções no sistema.
- ✓ O caso de uso 8 permite ao testador configurar: (i) o conjunto de teclas utilizadas e seus devidos códigos ASCII (*American Standard Code for Information Interchange*); (ii) o tempo de espera para a aplicação em teste carregar e; (iii) o intervalo entre o envio de dados de um campo para outro.

A Figura 6 retrata o diagrama de classes de domínio da ferramenta Testware.

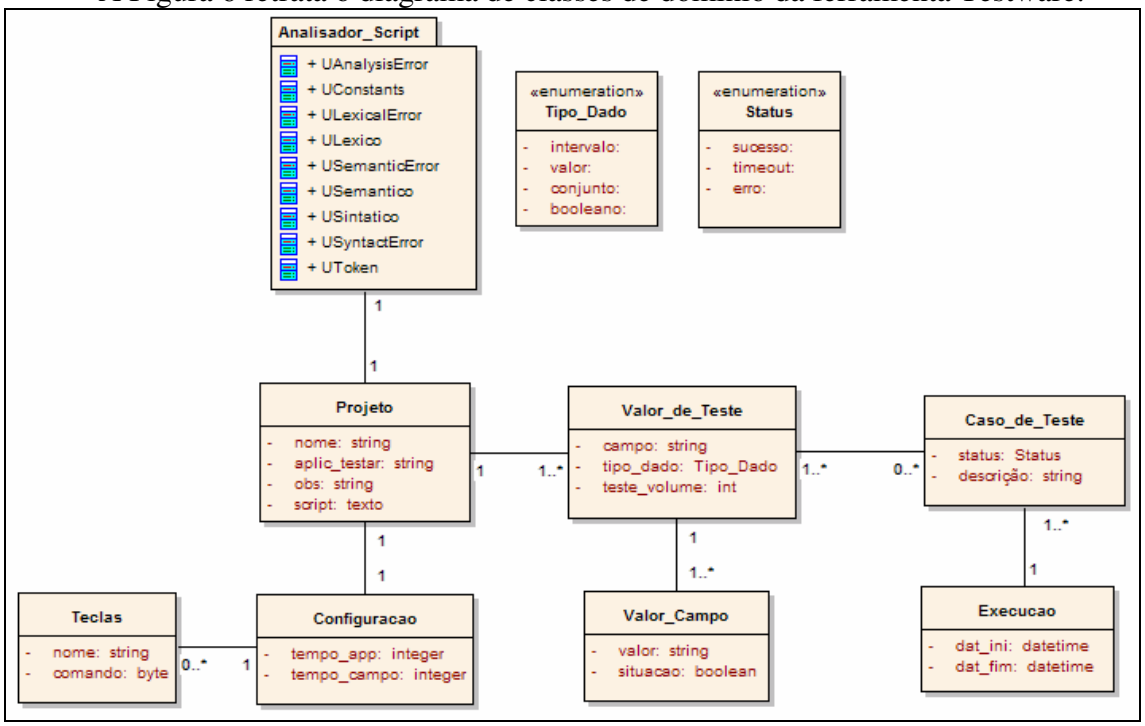


Figura 6. Diagrama de Classes

O diagrama de classes da ferramenta possui sete classes principais relacionadas entre si, conforme detalhamento:

- ✓ Projeto: é a classe principal da ferramenta, contendo informações gerais referente ao projeto;
- ✓ Valor de Teste: esta classe possui informação referente aos campos de entrada da aplicação em teste;
- ✓ Valor_Campo: valores a serem testados para cada campo de entrada da aplicação;
- ✓ Execução: é a classe que possui dados sobre a execução do teste;
- ✓ Caso_de_Teste: esta classe contém o status e considerações referente a execução do teste, bem como a referência aos valores testados no caso;
- ✓ Configuração: possui as configurações gerais da ferramenta, como: tempo para a aplicação em teste carregar e intervalo de tempo para envio de dados de um campo para outro;
- ✓ Teclas: classe com o código ASCII correspondente aos comandos utilizados no script, por exemplo: enter, F1, F2, tab, etc.

3.1 Estrutura do *Script*

O *script* tem o objetivo de apresentar o roteiro de execução da ferramenta a se testar. Sua gramática foi escrita e validada na ferramenta Gals [Gesser 2003], que gerou as classes ULexico e USintatico (pacote Analisador_Script da Figura 6) com a implementação do analisador léxico e sintático, respectivamente, que por sua vez foram incorporados a ferramenta. A Tabela 1 exibe as principais instruções para construção do *script*.

Tabela 1. Instruções da gramática do *script*

Instrução	Descrição
inicio_script	Primeira instrução do *script* e indica que ele foi inicializado, esta instrução é particularmente importante para permitir a implementação de diversos *scripts* em um mesmo projeto.
enquanto testevolume faca	Esta instrução deve ser utilizada para campos iterativos, por exemplo, o campo código de produto, em um sistema de caixa, na validação do *script* serão identificados os campos de entrada que estiverem dentro deste laço e solicitado o cadastrado da quantidade máxima de iterações desejadas para a aplicação do teste de volume, e para os demais casos de teste será realizado um *random* de dez.
inicio_enquanto	Identifica que o laço "enquanto" foi iniciado.
escreva	Declara os campos de entrada existentes no aplicativo em teste.
comando	Define a tecla utilizada para navegação entre os campos, por exemplo: "tab", "enter". Os códigos ASCII destas teclas devem estar devidamente cadastrados na ferramenta.
fim_enquanto	Marca o final do laço enquanto.
Escrevaconst	Instrução utilizada para enviar à aplicação valores/texto pré-definidos.
captura(x,y)	Função que retorna uma string contendo o valor / texto de determinada posição da tela.
:=	Atribui um valor a uma determinada variável.
escrevavar	Envia a aplicação o valor de uma variável interna do *script*.
ALT#A	Envia uma combinação de teclas para a aplicação.
fim_script	Identifica o término do *script*.

4. Estado da Arte

O aumento das exigências dos usuários e concorrência cada vez maior na área de desenvolvimento de software vem pressionando as empresas a aperfeiçoarem seus processos. No entanto, considerando a complexidade envolvida nos sistemas atuais, para se testar adequadamente um software, uma organização gasta 40% do esforço de projeto total em teste (Pressman, 2005). Para diminuir o tempo e custo do processo de teste de software algumas ferramentas propõem automatizar as atividades relacionadas com este processo, sendo algumas destas ferramentas descritas nesta seção.

4.1 TestComplete 3.11

A TestComplete é uma ferramenta de automação de testes, aplica testes de unidade, funcional e de regressão. É desenvolvida pela empresa AutomatedQA Corporation, fundada em 1999 com matriz nos Estados Unidos [AutomatedQA, 2005].

As principais funcionalidades da ferramenta são:

- ✓ Grava a interação do usuário com a interface do software, através dos eventos do mouse e teclado, gerando *scripts* em Delphi, Visual Basic e C;
- ✓ Permite criar ou alterar um *script* de teste;
- ✓ Reconhece variáveis e componentes da interface;
- ✓ Permite realizar teste de carga e de escalabilidade de aplicações web;
- ✓ Permite gerar o resultado dos testes em arquivo com extensão HTML (*Hyper Text Markup Language*) e XML (*Extensible Markup Language*), podendo assim incluir na documentação do software;
- ✓ Permite depurar o *script* incluindo *breakpoints*; e
- ✓ Realiza testes em interfaces *desktop* e *web*.

4.2 QuickTest Professional 8.2

Esta ferramenta pertence a empresa Mercury Intercative Corporation, fundada em 1989 [Mercury 2005]. A QuickTest é um módulo do suíte de testes que contém mais dois módulos:

- ✓ TestDirector: é o módulo gerenciador de testes e defeitos, controla o processo de testes de "ponta-a-ponta", garantindo que todos os requisitos dos usuários sejam cobertos pelos casos de testes construídos; e
- ✓ LoadRunner: responsável pela automação de testes de desempenho, atendem aos testes de carga e volume, a ferramenta permite a criação de um cenário para a execução dos testes e monitora *on-line* os elementos envolvidos, como: servidores, banco de dados e rede.

A QuickTest é uma ferramenta de automação de testes funcionais, ela grava a ação desejada na aplicação, parametriza e em seguida executa o teste. Suas principais características e funcionalidades são:

- ✓ Gera script da interação do usuário em VBScript;
- ✓ Permite incluir ou alterar o *script*;

- ✓ Realiza testes em interfaces *web* e *desktop*;
- ✓ Possibilita a depuração do *script*;
- ✓ Reconhece variáveis e componentes da interface;
- ✓ Após a execução apresenta um *log* com o resultado do teste;
- ✓ Permite cadastrar o executável ou *browser* a ser testado, desta forma o executa automaticamente;
- ✓ Permite cadastrar parâmetros de entrada e saída; e
- ✓ Criptografa as senhas digitadas nas interfaces.

4.3 J-Fut

Esta é uma ferramenta acadêmica, apresentada no 19º Congresso Brasileiro de Engenharia de Software. A J-Fut possui o objetivo de apoiar o teste funcional de programas desenvolvidos em Java, possuindo as seguintes características e funcionalidades [Rocha et al. 2005]:

- ✓ Oferece suporte as técnicas de particionamento de equivalência, análise de valor limite e este funcional sistemático ;
- ✓ Permite que pré e pós condições do sistema sejam avaliadas;
- ✓ Permite efetivar as atividades básicas para a aplicação de critérios: instrumentação, seleção, execução de casos de testes e análise de cobertura;
- ✓ Realiza teste funcional de unidade;
- ✓ Analisa o comportamento interno do programa identificando classes de equivalência de uma operação;
- ✓ Não há acesso direto ao programa em teste e sua execução sempre é feita a partir dos casos de teste implementados; e
- ✓ Os testes são feitos a partir de um projeto de teste, que armazena todos os dados relativos ao teste, incluindo seu nome, classes e métodos em teste, bibliotecas necessárias, requisitos de teste, condições de entrada e registro de execução.

4.4 Análise Comparativa

A Tabela 2 mostra uma análise comparativa entre as três ferramentas estudadas: TestComplete, QuickTest e J-Fut, e a ferramenta desenvolvida neste projeto, a TestWare.

Tabela 2. Análise comparativa entre ferramentas similares

Características	TestComplete	QuickTest	J-Fut	TestWare
Realiza testes de quais fases (Unidade, Integração, Sistema e Aceitação)	- Teste de unidade - Teste de integração	- Teste de unidade - Teste de integração	- Teste de unidade	- Teste de integração
Tipos de Teste (Caixa Branca e Caixa Preta)	Caixa Preta	Caixa Preta	- Caixa Preta e Branca	- Caixa Preta
Método de caixa preta abordado	- nenhum	- nenhum	- Particiona-mento de equivalência - Teste funcional sistemático - Análise de valor limite	- Particionamento de equivalência - Análise de valor limite
Categorias de Testes	- funcional - regressão - carga - escalabilidade	- funcional - volume - carga	- funcional	- funcional - volume - regressão
Classificação da ferramenta	- execução dos testes	- execução dos testes	-desenvolvimento dos testes - execução dos testes	- desenvolvimento dos testes - execução dos testes
Tipo de aplicações testadas	-web - *desktop*	-web - *desktop*	- *desktop* escritas em Java	- *desktop* indepentente da linguagem
Formato de entrada dos casos de testes	- *script*	- *script*	- base de casos	- base de casos - *script*
Resultado ao usuário	- interface e gera arquivo de *log* em HTML e XML	- interface e gera arquivo de *log*	- interface	- interface

A partir da comparação percebe-se que a TestComplete e a QuickTest apresentam praticamente as mesmas características. As duas gravam a iteração do usuário com a interface em *script*, após executa o teste, e apresenta relatórios de *logs*. Estas ferramentas são comerciais e apresentam um custo elevado. Possuem versão *trial* para *download*, para a realização desta análise comparativa foram aplicados diversos testes, com aplicações *web* e *desktop*.

Já a ferramenta acadêmica J-Fut implementa dois métodos de caixa preta, o método de particionamento de equivalência e o método de análise de valor limite, através destes métodos gera uma base de casos de testes e os executa. Uma grande desvantagem da J-Fut é testar somente aplicações implementadas na linguagem Java.

Com a pesquisa das ferramentas de automação de testes, percebe-se que existem algumas opções já em comercialização ou em desenvolvimento através de projetos de pesquisa, porém, a Testware apresenta um diferencial em relação aos produtos em comercialização: uma vez descrito o roteiro de interação (*script*) o testador informa os valores e os casos de testes são gerados automaticamente, ou seja, a partir do mesmo *script* tem-se diversos casos de testes (situação não permitida nas ferramentas analisadas). Outro aspecto a destacar refere-se ao relatório (já formatado para ser

encaminhado aos programadores), pois nas demais ferramentas comerciais o testador gera o relatório a partir da análise dos *logs* da ferramenta.

5. Considerações Finais

Percebe-se que a área de teste de software vem sendo explorada cada vez mais, tendo em vista os seguintes aspectos: os sistemas tornam-se cada vez mais complexos; a concorrência entre empresas da área de desenvolvimento de software é cada vez maior; e os clientes tornam-se mais exigentes.

Dias Neto e Travassos (2006) ressaltam que "apesar da importância das atividades de planejamento e controle de testes de software, observa-se na literatura técnica da área de testes pouca importância dada a elas. Poucas abordagens que apóiam a essas atividades são propostas, e normalmente para um contexto específico, e um número mais reduzido ainda é aplicado na indústria, caracterizando o estado da arte das atividades de planejamento e controle dos testes." Tendo em vista este relato, a ferramenta Testware apresenta uma abordagem para este problema, atuando na definição e execução de casos de testes.

A validação da ferramenta Testware[2] foi realizada através do módulo de caixa do sistema ItlSys, pertencente a empresa Intelidata Ltda, que constitui um sistema de informação ERP (*Enterprise Resource Planning*) de grande porte, destinado a área comercial. A Testware também foi validada através do aplicativo SAPRO (Sistema de Acompanhamento de Processos), desenvolvido pela empresa Four-M Comercial e Serviços de Informática Ltda, tendo como objetivo automatizar as rotinas de escritórios de advocacia. Alguns aspectos detectados para melhoria dos aplicativos testados foram:

- ✓ no cadastro de clientes do SAPRO o campo relativo a data de cadastro aceita, sem nenhuma mensagem de confirmação, datas maiores que a data atual;
- ✓ no registro de venda do ItlSys foram aplicados valores inválidos para o campo de entrada "guia" e em dois casos a venda finalizou;
- ✓ foram inseridos valores inválidos para o campo "parcelamento" do ItlSys, porém, cada um obteve um erro diferente. No primeiro, foi enviado a aplicação o valor "02566985", o caixa exibiu o seguinte erro "*Run-time error 6 Overflow*", significando que o valor inserido contém mais caracteres do que foi definido no banco, sendo que após o erro o aplicativo do caixa fechou deixando a operação da venda não concluída. No segundo caso, foi enviado ao campo "parcelamento" o valor "abc", e a aplicação aceitou normalmente, finalizando a venda como cheque a vista (enquanto o comportamento esperado seria um aviso de parcelamento inválido).

Através da validação foi possível relatar algumas inconformidades nos aplicativos testados, bem como identificar melhorias para aperfeiçoar a ferramenta Testware, como a inclusão de gravação de macros, controle de sub-projetos, automatização do resultado da execução, dentre outros.

Referências

AutomatedQA. (2005) TestComplete. Disponível em: <http://www.automatedqa.com/products/testcomplete/>. Acesso em: 08 set. 2005.

[2] Em http://www.inf.furb.br/~fabiane está disponível um vídeo demonstrativo das principais funcionalidades da ferramenta.

Bartié, A. (2002) “Garantia da qualidade de software” , Campus.

Dias Neto, A.C. e Travassos, G.H. (2006) “Maraká: infra-estrutura computacional para apoiar o planejamento e controle dos testes de software.” In *Simpósio Brasileiro de Qualidade de Software*, p. 250 – 264.

Gesser, C. E. (2003) “GALS: Gerador de analisadores léxicos e sintáticos.” Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)–Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

Mats, L. (2001) “The top five software-testing problems and how to avoid them”, EDN Europe, Fevereiro, Vol. 46 Issue 2, p 37.

Mercury. (2005) **QuickTest**. Disponível em: <http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/>. Acesso em: 10 set. 2005.

Pressman, R. S. (2005) “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 6th ed, New York.

Rocha, A. D.; Simão, A. S.; Maldonado, J. C.; Masiero, P. C. (2005) “Uma ferramenta baseada em aspectos para o teste funcional de programas Java”. In: *Congresso Brasileiro de Engenharia de Software*. Minas Gerais. Disponível em: < http://www.sbbd-sbes2005.ufu.br/arquivos/17-%209610.pdf>. Acesso em: 5 set. 2005.

Qualifica: Uma Ferramenta para Apoio a Construção de Algoritmos Estruturados

Mauro Marcelo Mattos, Jean Fábio Fuchs

Departamento de Sistemas e Computação – Universidade Regional de Blumenau (FURB)
CEP –89035-160– Blumenau – SC – Brasil

mattos@inf.furb.br, jeanfabiofuchs@hotmail.com

Resumo. *O presente trabalho apresenta um método de desenvolvimento de algoritmos baseado na premissa de que um enunciado bem elaborado e uma análise detalhada deste enunciado por parte do aluno pode contribuir de forma importante no aprendizado de construção de algoritmos. O trabalho descreve também uma ferramenta computacional que foi construída para facilitar o processo de análise dos enunciados. O resultado final é um pseudocódigo desenvolvido pelo aluno..*

1 Introdução

De acordo com Casas (1999), a pedagogia em ciências de educação está baseada em dois princípios: (a) a instrução pode desenvolver as habilidades do aprendiz para que compreenda intuitivamente como funciona o mundo natural em vez de inculcar-lhe a representação formal e as habilidades de raciocínio que os cientistas usam; (b) a instrução que pode ajudar o aprendiz a desenvolver o seu modelo mental (existente) para uma concepção mais exata da realidade.

Segundo as diretrizes curriculares do MEC (MINISTÉRIO DA EDUCAÇÃO, 1999, p.6), "A programação, entendida como programação de computadores, é uma atividade voltada à solução de problemas. Nesse sentido ela está relacionada com uma variada gama de outras atividades como especificação, projeto, validação, modelagem e estruturação de programas e dados, utilizando-se das linguagens de programação propriamente ditas, como ferramentas. Ao contrário do que se apregoava há alguns anos atrás, a atividade de programação deixou de ser uma 'arte' para se tornar uma ciência, envolvendo um conjunto de princípios, técnicas e formalismos que visam à produção de software bem estruturado e confiável. Portanto o estudo de programação não se restringe ao estudo de linguagens de programação. As linguagens de programação constituem-se em uma ferramenta de concretização de software, que representa o resultado da aplicação de uma série de conhecimentos que transformam a especificação da solução de um problema em um programa de computador que efetivamente resolve aquele problema".

Neste contexto Mattos, Fernandes e Lopez (1999) afirmam que, "os estudantes que iniciam um curso de Graduação em Informática, normalmente encontram uma primeira dificuldade relacionada com a disciplina de Introdução a Programação (ou com nome similar), cujo principal objetivo é o de introduzir os conceitos básicos de lógica de programação. Esta dificuldade é, na maioria das vezes, decorrente da falta de experiência com os aspectos relacionados a ambientes industriais e/ou comerciais, pois é

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 75-87.

a partir destes ambientes que são caracterizados os exercícios propostos".

Analisando-se o perfil dos alunos, verifica-se que em sua maioria, são oriundos do 2º Grau e, portanto, possuem conhecimentos abstratos sobre áreas científicas (matemática, física, biologia, e outros.). Porém, quando se deparam com a descrição textual dos enunciados dos problemas apresentados nesta disciplina introdutória, geralmente encontram dificuldades em identificar como extrair as informações necessárias para iniciar a solução destes problemas. (MATTOS, 2002; KOVACIC, 2003, p. 794).

Conforme Carvalho e Chiossi (2001, p. 19), quando os requisitos não são totalmente compreendidos, registrados e comunicados para a equipe de desenvolvimento, pode haver discrepância entre o que o sistema construído faz e o que deveria fazer. Estas discrepâncias no contexto de introdução à programação conduzem ao desenvolvimento de soluções erradas (na melhor das hipóteses) ou mesmo à dificuldade no desenvolvimento de uma solução (mesmo que incorreta).

Assim, neste artigo é apresentada uma metodologia de ensino/aprendizagem e uma ferramenta que procura superar as dificuldades apontadas anteriormente estando o texto organizado da seguinte forma: a seção 2 apresenta a contextualização do trabalho; a seção 3 descreve o método de desenvolvimento de algoritmos utilizado; a seção 4 descreve a ferramenta desenvolvida para automatizar o método. Por fim, são apresentados alguns trabalhos correlatos e os resultados obtidos e as limitações da ferramenta proposta.

2 Contextualização do trabalho

O objetivo desse trabalho é apresentar uma metodologia de ensino/aprendizagem para disciplina de introdução a programação que permita ao aluno desenvolver as habilidades necessárias para a interpretação dos enunciados de problemas característicos desta disciplina e também conhecer uma técnica que facilita o processo de construção de soluções algorítmicas. Esta metodologia foi concebida a partir de experiências práticas de sala de aula, e vem sendo desenvolvida na forma de projeto de pesquisa e projetos de conclusão de curso desde 1998 (MATTOS, FERNANDES e LÓPEZ, 1999; MATTOS, 2000; GUBLER, 2002; HEIZEN, 2002; MATTOS, 2002; FREITAS; 2003; FUCHS, 2006).

A intenção principal desse artigo é divulgar essa experiência sem querer afirmar que essa seja uma metodologia ideal para a disciplina, mas sim, uma proposta que foi colocada em experiência e que trouxe bons resultados. Foi possível constatar uma melhora no aproveitamento e um aumento no grau de interesse e satisfação dos alunos no decorrer do curso.

3 O método

Nesta seção é apresentado o método de ensino de introdução a programação desenvolvido pelo Prof. Mauro Mattos (Mattos, 2005). O método está baseado em duas premissas: a construção de enunciados detalhados contendo exemplos de entradas e saídas esperadas e, um processo de análise detalhada dos dados de entrada e de saída. Tendo em vista facilitar a explicação, será utilizado um dos enunciados de problemas de introdução a programação utilizados em aula (Quadro 1). A partir deste enunciado o

aluno deve construir uma tabela conforme apresentado na Figura 1. Nesta tabela o aluno registra, da esquerda para a direita e, de cima para baixo, a seqüência com que os dados de exemplo são entrados no sistema e qual a saída esperada para aquela entrada.

Observe-se que, cada entrada é registrada em uma linha específica (na coluna das entradas) e as saídas são registradas uma linha abaixo da última entrada (na coluna correspondente às saídas).

- Enunciado: Ler e imprimir um conjunto de dados contendo nome e uma nota referente a um grupo de n alunos. Parar a leitura quando o nome digitado for = "fim".
  - **Exemplo de dados:**

aluno:Daniel Ana Elvis Joao Maria fim
Nota : 7 6 5 10 6 *

Quadro 1 - Exemplo de enunciado de problema

A partir do momento em que o aluno informou todas as entradas e saídas, o próximo passo é qualificar as informações fornecidas identificando um nome para a variável que será utilizada para armazenar aquele valor entrado ou saído (quando for o caso) e um tipo (inteiro, caractere, lógico, string). A Figura 2 representa esta operação.

Figura 1 – Relacionando saídas após as entradas

Tendo sido qualificadas as variáveis de entrada e saída o próximo passo é identificar se os nomes das variáveis estão repetidos. Por exemplo, na Figura 2 os nomes de variáveis "nome" e "nota" repetem-se várias vezes. Este exemplo permite algumas reflexões, quais sejam:

a) Se o aluno está aprendendo a utilizar estruturas de repetição (*while*, *repeat*, *for*), geralmente no início do semestre, então, a constatação de que os nomes das variáveis repetem-se indicam uma situação objetiva da necessidade do emprego deste tipo de construção;

b) Se o aluno já conhece as estruturas de repetição, então a constatação da repetição pode facilitar a introdução do conceito de estruturas de armazenamento do tipo matrizes;

c) Uma vez que o aluno tenha assimilado o conceito de matrizes para armazenamento de dados homogêneos, o próximo passo é a possibilidade da introdução do conceito de registros, pelo agrupamento de nomes de variáveis repetidas.

Entradas	Processamento	Saídas
Nome: Daniel		
Nota: 7		
		Nome: Daniel **Nota**: 7
Nome: Ana		
Nota: 6		
		Nome: Ana **Nota**: 6
Nome: Elvis		
Nota: 5		
		Nome: Elvis **Nota**: 5
Nome: João		
Nota: 10		
		Nome: João **Nota**: 10
Nome: Maria		
Nota: 6		
		Nome: Maria **Nota**: 6
Nome: fim		

Figura 2 – Identificação dos nomes das variáveis

O próximo passo consiste em identificar como os dados de saída são produzidos a partir do fornecimento dos dados de entrada. Supondo-se que o enunciado propusesse a emissão de um relatório contendo o nome e a nota dos alunos cuja nota é maior que 6, a coluna processamento seria utilizada para registrar a necessidade de uma operação de teste para filtrar a saída. A Figura 3 indica que, quando foram informados os dados: "nome: Elvis" e "nota: 5", não houve saída porque a nota ficou abaixo do limiar estabelecido no enunciado. Portanto, não foi produzida uma saída para aquela entrada.

Entradas	Processamento	Saídas
Nome: String: Daniel		
Nota: inteiro: 7		
	> 6, produz saída	
		Aluno: Daniel Nota: 7
Nome: String: Ana		
Nota: inteiro: 6		
	> 6, produz saída	
		Aluno:Ana Nota: 6
Nome: String: Elvis		
Nota: inteiro: 5		
	< 6, não produz saída	
Nome: String: João		
Nota: inteiro: 10		
	> 6, produz saída	
		Aluno:João Nota: 10
Nome: String: Maria		
Nota: inteiro: 6		
	6 não é maior que 6, portanto, não produz saída	
Nome: String: fim		

Figura 3 – Anotação de necessidade de manipulação dos dados de entrada

Outro aspecto que pode ser identificado a partir da análise dos dados de entrada é a condição de que a nota de “Maria” é “6” e que, o primeiro esboço de solução pode fazer com que o nome “Maria” não seja impresso tendo em vista que não foi considerada a condição de “maior ou igual” (embora este possa ser um requisito da especificação do problema). Ou seja, a partir da análise dos dados, o aluno é conduzido a refinar as suas proposições lógicas no sentido de equacionar o problema proposto.

Numa análise mais detalhada, o aluno pode perceber que o processo de repetição das entradas produzindo as saídas implicaria na repetição do conjunto de linhas apresentado na Figura 4a. Isto não seria uma solução aceitável do ponto de vista computacional. Além disso, o destaque para a condição de parada (nome do aluno = “fim”) poderia levar o aluno a produzir uma solução conforme apresentado na Figura 4b.

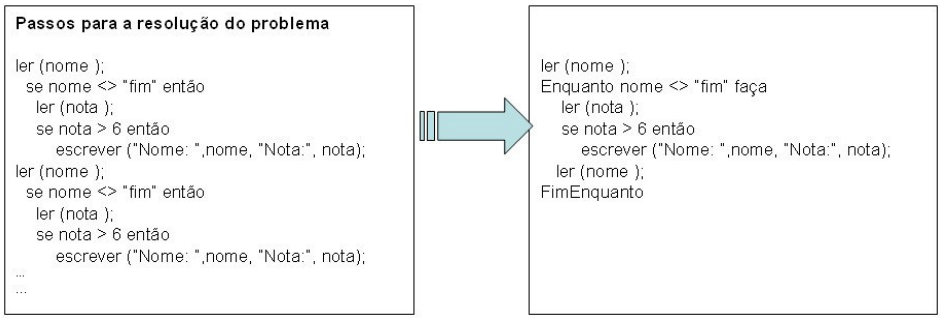


Figura 4– a) exemplo de um pseudocódigo extraído a partir do detalhamento da planilha; b) exemplo de um refinamento de um bloco de repetição.

Este processo de fazer com que o aluno detalhe as entradas e saídas, posicionando-as de forma deslocada nas linhas da tabela conduzem-no a pensar em termos temporais – requisito implícito na construção de soluções algorítmicas. Com isto o aluno tem facilidade em identificar:

a) O que ocorre antes do que (exemplo entra-se primeiro o nome depois a nota);

b) O que ocorre depois do que (exemplo: entra-se primeiro o nome e a nota e depois se exibe no nome e nota).

4 A ferramenta desenvolvida

A partir da aplicação prática do método acima descrito, propôs-se a construção de uma ferramenta que automatizasse o processo. O sistema foi concebido na forma de dois módulos: um módulo professor e um módulo aluno. O módulo professor permite que o professor configure um exercício enquanto o módulo aluno permite ao aluno solucionar o problema proposto (FUCHS, 2006).

A Figura 5 mostra um exemplo da tela principal do programa com o editor do enunciado do problema e a definição das estruturas da base de dados, com suas tabelas, atributos e registros. Nesta figura é possível identificar: (1) configuração do nível de complexidade do exercício, no caso aqui é intermediário; (2), exibe a lista de atributos da tabela atual (Curso); (3), exibe a lista de tabelas do exercício, ficando a tabela atual (Curso) em destaque, exibindo detalhadamente o seu *alias* e a quantidade de atributos e registros; (4) mostra a fase atual da criação do exercício, que no caso aqui o professor

está Definindo a Base de Dados e Registros; (5) um botão que permite o professor ir para a próxima fase de criação do exercício que é a Formatação de Saída; (6) editor de textos aonde o professor deverá descrever o enunciado do exercício; (7) mostra a lista de registros de entrada da tabela atual (Curso) do exercício.

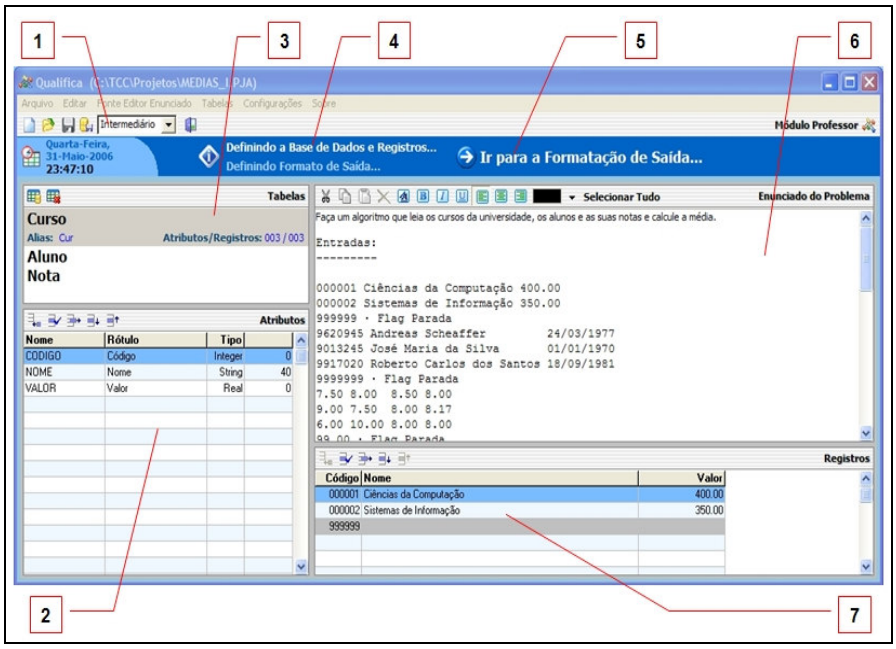


Figura 5– Tela principal do módulo professor.

A Figura 6 apresenta alguns dos recursos disponíveis ao professor na configuração das tabelas de exemplos a serem utilizados pelos alunos na solução do problema proposto.

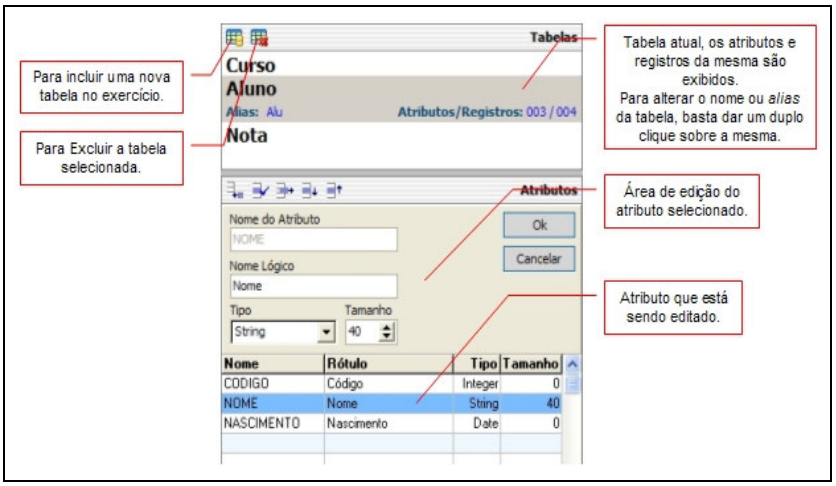


Figura 6 – Recursos para criação das tabelas de exemplos.

A Figura 7 apresenta um diagrama de atividades do módulo professor e a Figura 8 apresenta a tela do módulo aluno. Nesta figura é possível identificar: (1) botões para avançar e retornar as fases da solução do exercício, o aluno só poderá ir adiante na

solução depois de concluir a fase atual; (2) descreve a fase em que o exercício se encontra no momento: “Classificando as Entradas e Saídas...”, é fase em que o aluno terá que descobrir a seqüência de leitura das entradas e as saídas; (3) os registros de entradas que foram selecionados, tem a aparência de um botão pressionado; (4) um registro de entrada que está em destaque, quando o *mouse* é posicionado, a mesma se torna intermitente; (5) as entradas que ainda não foram selecionadas; (6) o botão para visualizar o formato de saída em formato texto; (7) os itens do formato de saída, nas cores iguais a grade significam que já foram selecionados; (8) os itens do formato de saída que ainda não foram selecionados; (9) o tempo total que o aluno utilizou para resolver o exercício, a partir da primeira alteração em qualquer parte da tela o tempo é iniciado ou reiniciado caso o aluno esteja dando continuidade a um exercício que foi parcialmente resolvido; (10) uma mensagem exibindo qual o próximo passo que o aluno deve tomar, no caso, qual o próximo registro de entrada que deve ser selecionado; (11) exibe o nível do exercício, no caso aqui é intermediário; (12) mensagens informando quais fases da solução do exercício que o aluno já completou; (13) exibe a grade de classificação das entradas, aqui especificamente uma entrada que possui saídas vinculadas a ela; (14) a grade de classificação das saídas, caso a entrada tenha alguma saída vinculada é então exibida.

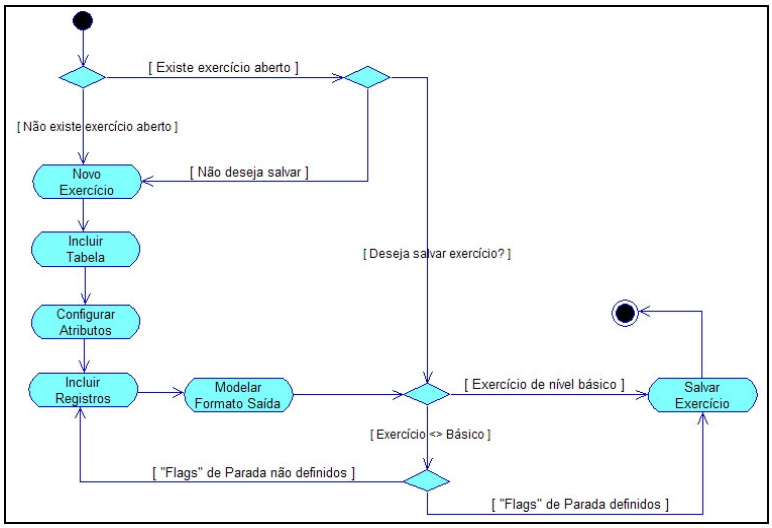


Figura 7 – Diagrama de atividades do módulo professor.

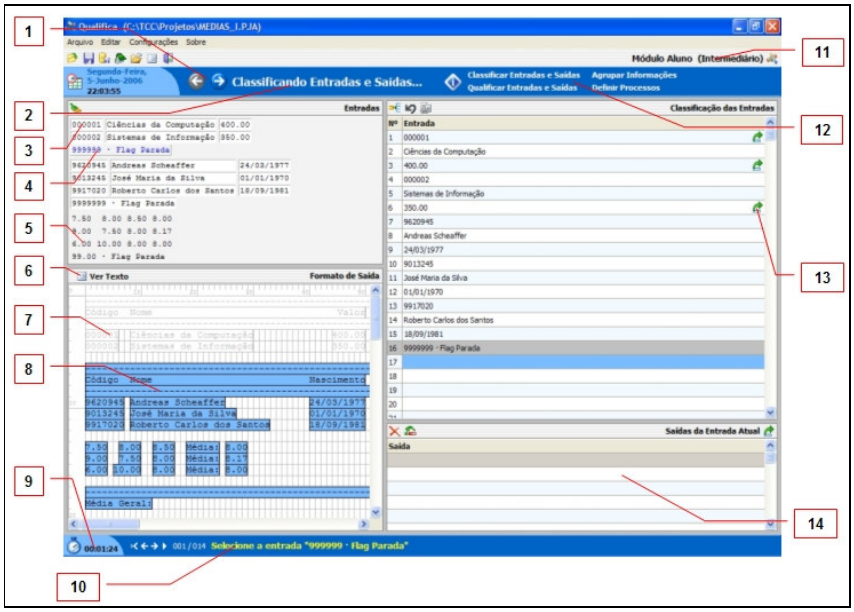


Figura 8 – Tela do modulo do aluno.

O conceito de agrupamento de entradas diz respeito ao processo repetitivo de entrada de dados em um programa que é associado a um campo (ou a vários campos de um registro ou de um *array*) até que uma condição de fim seja detectada.

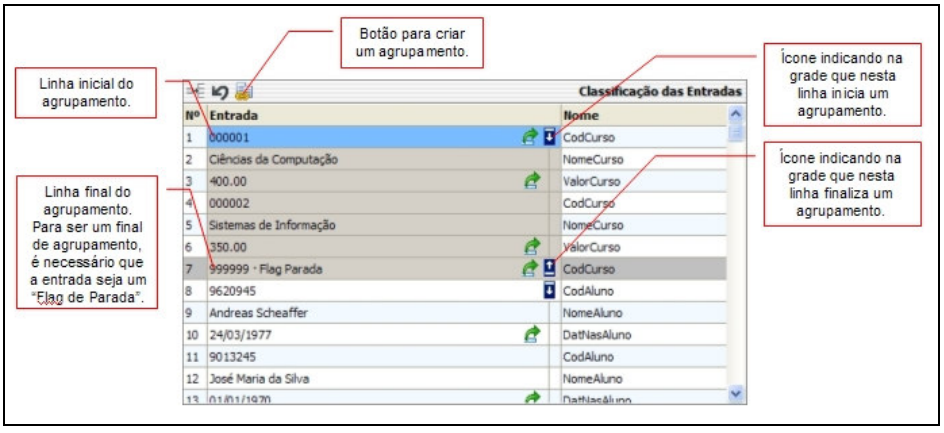


Figura 9 – Agrupamento informações

Quando o processo de associação entre entradas e saídas é encerrado, a janela que contém estas informações passa a ficar oculta e é desabilitada a possibilidade de qualquer alteração nas mesmas. Contudo, o aluno pode consultá-las clicando no botão localizado na parte superior da barra lateral. O processo de agrupamento (Figura 9) consiste em clicar na linha da entrada inicial e arrastar o mouse até a linha da entrada final (que deve ser uma entrada definida no módulo professor como "Flag de Parada").

Depois de marcado o bloco, basta clicar no botão "Agrupar" (Figura 9) e a tela de cadastramento do agrupamento (Figura 10) irá aparecer para que o aluno informe o nome do agrupamento, tipo do agrupamento – "procedimento" ou "laço de repetição", tipo do laço de repetição e uma dica ou descrição sobre o que este agrupamento trata,

esta descrição irá aparecer no código fonte no Portugol. Além disso, o sistema apresenta o intervalo das linhas inicial e final que foram selecionadas na grade de entradas (Figura 9) e que resultarão no novo agrupamento.

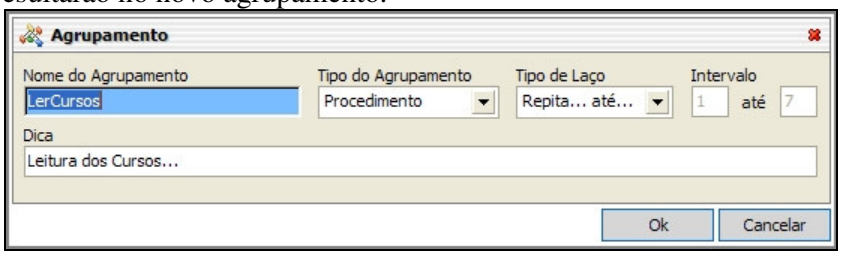


Figura 10 – Tela de cadastramento de agrupamentos

A Figura 11 apresenta o diagrama de atividades que estão envolvidas no processo de solução de um exercício no ambiente da ferramenta construída.

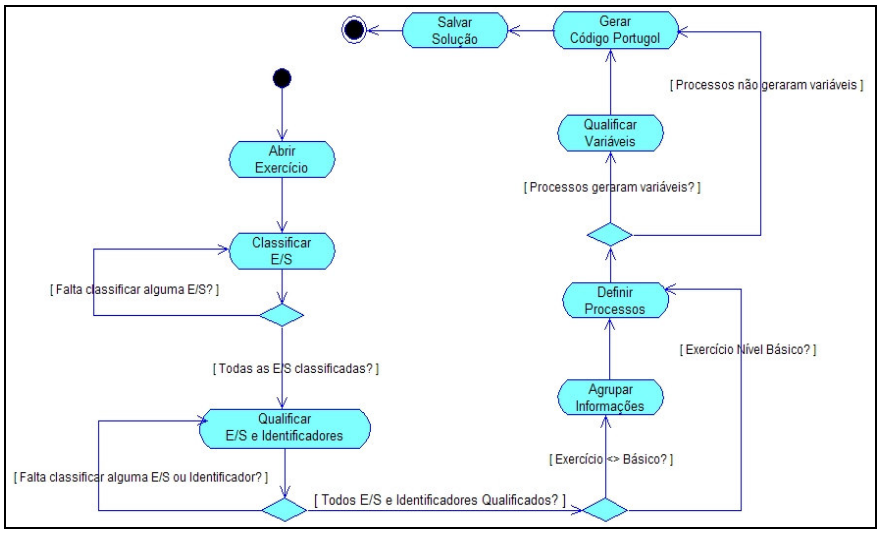


Figura 11 – Diagrama de atividades do processo de solução de um exercício pelo aluno

Tendo em vista conduzir o aluno no processo de aprendizagem, existe uma barra de informações que mantém o aluno posicionado em que fase ele está (Figura 12). Um aspecto importante a destacar é a possibilidade que o aluno tem de navegar entre as fases voltando atrás ou avançando até o ponto em que ele conseguiu chegar na solução do problema.

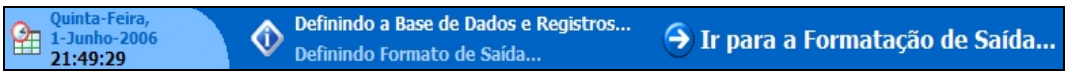


Figura 12 – Barra de informações para o aluno.

Uma funcionalidade importante é que na medida em que o aluno vai "consumindo" os dados de exemplo cadastrados pelo professor, os mesmos vão sendo marcados como já utilizados. A Figura 13 representa a situação em que o aluno utilizou todos os dados de exemplo do exercício.

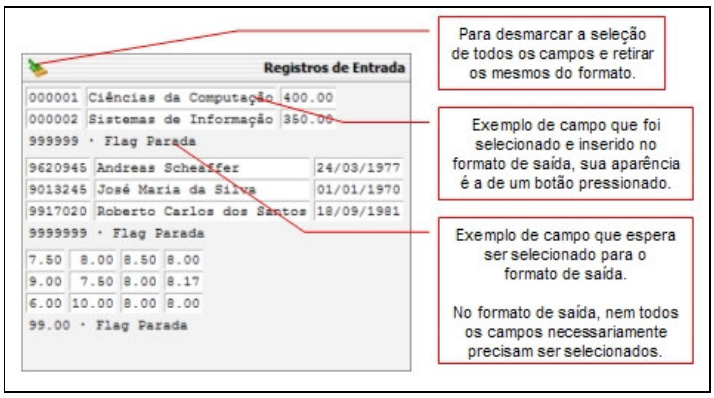


Figura 13 – Dados de entrada totalmente utilizados pelo aluno.

Da mesma forma, os dados do relatório de saída devem ser completamente "consumidos" pelo aluno à medida que as entradas vão sendo informadas (Figura 14)

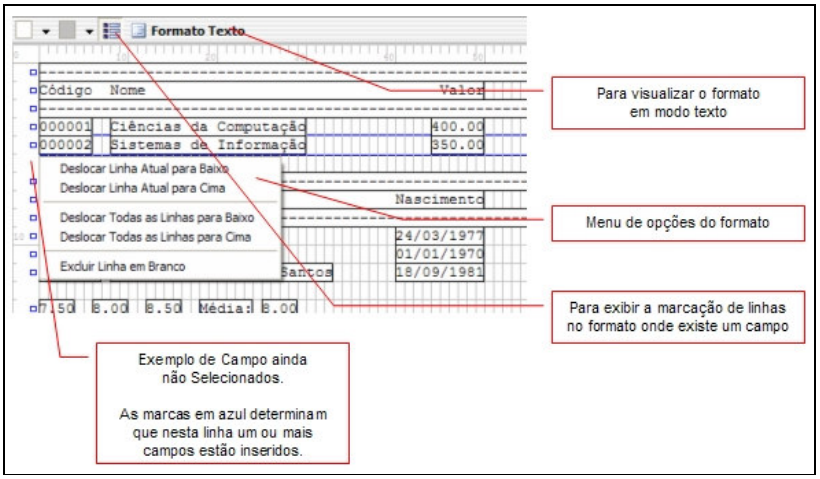


Figura 14 – Dados de saída totalmente utilizados pelo aluno.

A Figura 15 apresenta um apanhado de telas do sistema caracterizando o módulo professor sendo utilizado na construção de uma especificação detalhada do problema e telas do módulo aluno caracterizando a resolução do mesmo até a geração de um pseudocódigo da solução proposta. Maiores detalhes sobre a arquitetura do sistema podem ser obtidos em Fuchs (2006).

5 Trabalhos correlatos

A questão de ensino de algoritmos e lógica de programação é um tema recorrente. De acordo com Santiago e Dazzi (2003), relacionam, entre outros os trabalhos de Brown (1987,1988), Stubbs e Webre (1988), Stasko (1990), Brown (1991), Amorim e Rezende (1993), Szwarcfiter e Markenson (1994), Cares (2002), Medeiros (2001) e Mendes e Gomes (2000). Além disso, podemos citar: Gloor (1992), Esmin (1998), Colleen,Stasko e Ashley (1999), Heinzen (2002), Gubler (2002), Silveira e Esmin (2003), Santiago e Dazzi (2003) e Pereira Júnior e Rapkiewicz (2006), entre outros.

Segundo Pereira Júnior e Rapkiewicz (2004) apud Ferrandin e Stephani(2005), a análise de 105 artigos sobre o tema mostra que há três vertentes na busca de soluções

para os problemas do processo: Ferramentas, Estratégias e a união de ambas. Esta análise parece sugerir que a união de ferramentas computacionais e estratégias têm se demonstrado como melhor proposta. O presente trabalho associa uma estratégia (refinamentos sucessivos) a uma ferramenta computacional como forma de auxiliar no processo de ensino-aprendizagem.

6 Resultados e limitações

O presente trabalho apresentou os fundamentos para a utilização de técnicas de desenvolvimento estruturado de programas na construção de uma ferramenta de apoio ao ensino de desenvolvimento de algoritmos. Como referido no texto, o projeto atual está inserido no contexto de um projeto mais amplo de desenvolvimento de uma ferramenta para ensino de algoritmos Conforme apresentado, a filosofia sobre a qual o sistema foi construído está baseada em dois conceitos principais: (i) uma especificação detalhada do problema por parte do professor e (ii) uma análise detalhada da especificação por parte do aluno. Esta análise detalhada é conduzida pela ferramenta de tal forma que, após verificar todos os elementos da especificação o aluno obtém um pseudocódigo da solução algorítmica do problema a ser desenvolvido.

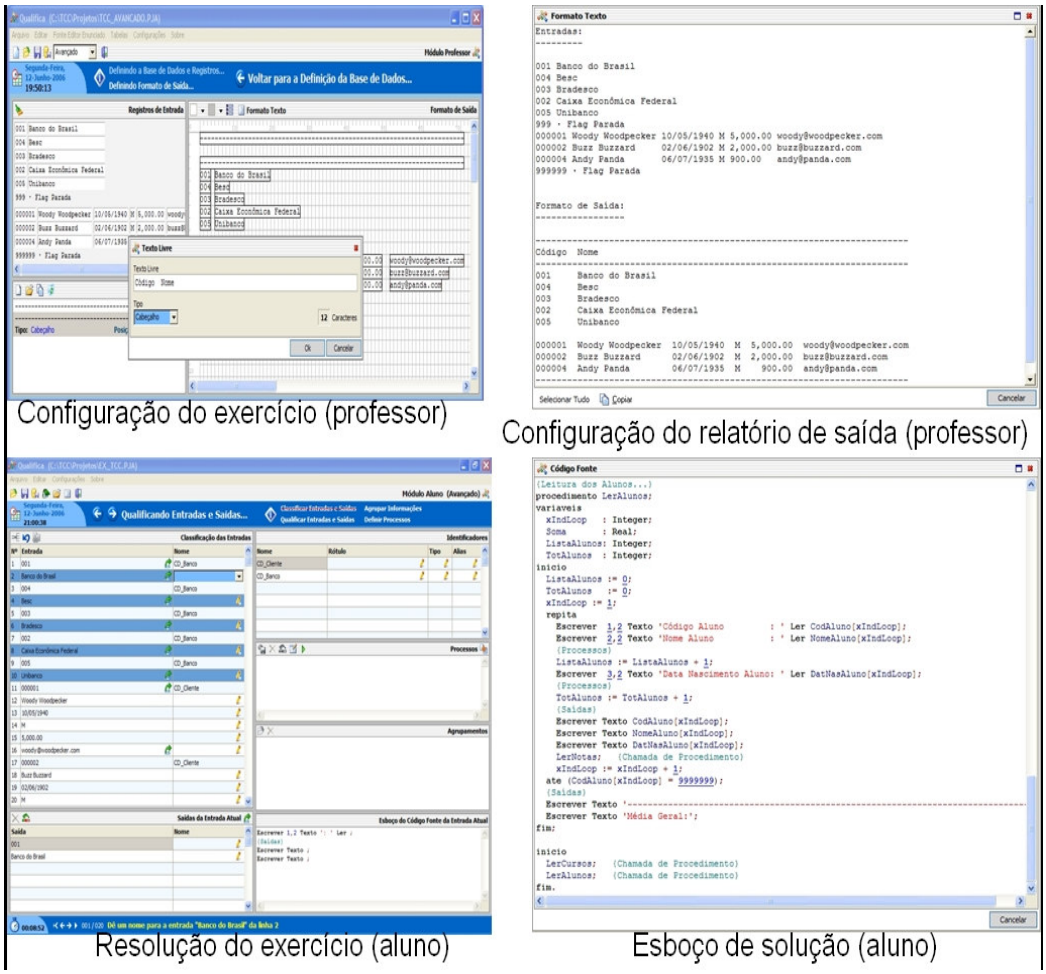


Figura 15 – Telas do modo professor e aluno

O método apresentado foi aplicado em turmas de primeiro e segundo semestre durante o período de 2004/1 e 2004/2. A avaliação dos acadêmicos foi que a estratégia de refinamento atuou como um facilitador no processo de compreensão do enunciado de um problema na medida em que auxiliou os acadêmicos na organização das idéias de como construir uma solução algorítmica para os problemas propostos. A ferramenta descrita foi finalizada em jul/2006 sendo esperada a sua utilização para validação no segundo semestre de 2007.

Referências

AMORIM, R. V.; REZENDE, P. J. Compreensão de Algoritmos através de Ambientes Dedicados a Animação. In: SEMISH, 10., 1993.

BROWN, M. H. Algorithm Animation. The MIT Press, 1987.

BROWN, M. H. Exploring Algorithms Using Balsa-II. Computer, maio 1988. p. 14-36.

BROWN, M. H. Zeus: A System for Algorithm Animation and Multi-View Editing. Procedings...IEEE Workshop on Visual Languages, 1991.

CARES, P. L. L. Ambiente para teste de mesa utilizando fluxograma. TCC(Graduação)– Faculdade de Ciência da Computação, Universidade do Vale do Itajaí, Itajaí, 2002.

CARVALHO, A.M.B.R.; CHIOSSI, T.C.S. Introdução à engenharia de software. São Paulo: Unicamp, 2001.

CASAS, A.A. Contribuições para modelagem de um ambiente inteligente de educação baseado em realidade virtual. Florianópolis, 1999. Paginação irregular. Disponível em: <http://www.eps.ufsc.br/testes99/casas/>. Acesso em: 01 jun. 2004.

COLLEEN, K, STASKO, J.;ASHLEY, T. Rethinking the evaluation of algorithm animations as learning aids: an observational study. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT -GVU-99-10, March 1999.

ESMIN, A. A. A. . Portugol/Plus : Uma Ferramenta de Apoio ao Ensino da Lógica de Programação baseado no Portugol. In: IV Congresso Ibero-americano de Informática Educativa. Brasília, 1998, Brasilia/DF. Anais do IV Congresso Ibero-americano de Informática Educativa. Brasília, 1998.

FERRANDIN, M.;STEPHANI, S.L. Ferramenta para o ensino de Programação via Internet1 . In: SULCOMP - I Congresso Sul Catarinense de Computação, 2005, Criciúma. Anais do Sulcomp.

FREITAS, G. Protótipo de um sistema de animação de algoritmos. 2003. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FUCHS, J.F. Qualifica: uma ferramenta de suporte ao desenvolvimento de algoritmos. 2006. 107 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GLOOR, P.A. AACE – algorithm animation for computer science education. In Proceedings of the 1992 IEEE Workshop on Visual Languages, pages 25-31,Seattle, WA, September 1992.

GUBLER, A.I. Protótipo de um sistema especialista para auxiliar no ensino de algoritmos. 2002. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HEINZEN, L.A. Módulo de raciocínio baseado em casos em uma ferramenta de apoio ao ensino de lógica de programação. 2002. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

KOVACIC, Z. J. A comparison of learning and teaching styles. In: InSITE 2004 – Informing Science and Information Technology Education 4., 2004, Rockhampton, Australia. Proceedings… Santa Rosa, California, 2004. Paginação irregular. Disponível em: <http://proceedings.informingscience.org/InSITE2004/105kovac.pdf>. Acesso em: 1 abr. 2006.

MATTOS, M.M.; FERNANDES, A.; LÓPEZ, O.C. Sistema especialista para apoio ao aprendizado de lógica de programação. In: Congresso Ibero-americano de Educação Superior em Computação, 7., 1999, Florianópolis. Anais... Assunção: Universidad Autônoma de Asunción, 1999. p. 1-12.

MATTOS, Mauro. M. Construção de abstrações em lógica de programação. In: SBC 2000, 10., 2000, Curitiba. Anais... Curitiba: Editora Universitária Champagnat, 2000a. p. 60-60.

MATTOS, M.M. Learning how to build abstractions in programming logics classes. In: IE 2002 – CONGRESSO IBEROAMERICANO DE INFORMATICA, 6., 2002, Vigo, Espanha. Proceedings... Vigo, Espanha, 2002. Paginação irregular.

MATTOS, M.M. Linguagens para programação de sistemas. 2005. Paginação irregular. Notas de aula (Disciplina de Linguagens para Programação de Sistemas, Curso de Ciências da Computação). Depto de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

MEDEIROS, C.L.; DAZZI, R.L.S. Aprendendo Algoritmos com Auxílio da Web. II CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 2., 2002, Itajaí. Anais... Itajaí: UNIVALI, CTTMar, 2002.

MENDES, A. J. N.; GOMES, A. J. Suporte a aprendizagem de programação com o ambiente SICAS. In: Congresso Ibero Americano de Informática Educativa - RIBIE,5., 2000, Viña del Mar-Chile. Anais... Viña del Mar-Chile: Universidad de Chile, 2000.

MINISTÉRIO DA EDUCAÇÃO. Diretrizes curriculares de cursos da área de computação e informática. Brasília, 1999. Disponível em: <http://www.mec.gov.br/sesu/ftp/curdiretriz /computacao/co_diretriz.rtf>. Acesso em: 1 jun. 2005.

PEREIRA JR, J.C.R., RAPKIEWICZ, C. E. “Um Ambiente Virtual para apoio a uma Metodologia para Ensino de Algoritmos e Programação”. RENOTE. Revista Novas Tecnologias na Educação, v. 3, 2005.

SILVEIRA, I.J.; ESMIN, A. A. A. . AVA - Um Ambinete Visual para a Construção de Algoritmos. In: ICECE'2003 International Conference on Engineering and Computer Education, 2003, , São Vicente / Santos. Anais ICECE, 2003.

SANTIAGO, R.; DAZZI, R. L. S. . Ferramentas que auxiliam o desenvolvimento da lógica de programação.. In: XII SEMINCO - Seminário de Computação, 2003, Blumenau. Anais do XII SEMINCO. Blumenau : Furb, 2003. p. 113-120.

STASKO, J. T. Tango: A Framework and System for Algorithm Animation. Computer, setembro 1990. p. 14-36.

STUBBS, D. F.; WEBRE, N. W. Data Structures with Abstract Data Types and Pascal, Pacific Grove, Brooks/Cole, 2 ed., 1988.

SZWARCFITER, J.; MARKENSON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

Um Template Destinado à Construção de Sistemas Operacionais Para o VXt

Mauro M. Mattos, Antonio C. Tavares[1], Jorge S.Farias[2],Daniel S. Estrázulas[3]

Universidade Regional de Blumenau (FURB)
CEP – 89035-160 – Blumenau – SC – Brasil

União Metropolitana de Educação (UNIME)
CEP - 42700-000 - Lauro de Freitas - BA - Brasil

[1,3]{mattos,Tavares,estrazulas}@inf.furb.br, [2]jorgefarias@unime.com.br

Resumo. *Este artigo descreve o estágio atual do desenvolvimento de uma ferramenta de apoio ao ensino de conceitos básicos de sistemas operacionais e arquitetura de computadores. O VXt – virtual XT é um simulador de uma máquina baseada no processador Intel 8086. O presente trabalho descreve a estrutura de um template que serve como base para a construção de sistemas operacionais para o ambiente didático VXt. É descrita a arquitetura do hardware simulado bem como a funcionalidade do template.*

1 Introdução

Conforme Gagné, Brigss e Wagner (1992 apud CASAS, 1999), instrução, no contexto da escola, é um processo oferecido por instrutores humanos e envolve o professor e o aprendiz. Os méritos da instrução proporcionada por professores humanos são os aspectos dinâmicos e interativos do ensino.

Segundo Fagundes (1998) apud Veiga (2001), aprender por projetos é uma forma inovadora de romper com as tradições educacionais, dando um formato mais ágil e participativo ao trabalho de professores e educadores. Trata-se mais do que uma estratégia fundamental de aprendizagem, sendo um modo de ver o ser humano construir, aprendendo pela experimentação ativa do mundo. Ao elaborar seus projetos, o professor conduzirá seus alunos a um conjunto de interrogações, quer sobre si mesmos, quer sobre o mundo à sua volta, levando o aluno a interagir com o desconhecido ou com novas situações, buscando soluções para os problemas.

Conforme Maziero (2002), "uma das principais características da disciplina de Sistemas Operacionais é a relativa dificuldade em definir um seqüenciamento didático claro entre seus diferentes tópicos".

Conforme Machado e Maia (2004), "diversos pesquisadores como Ramakrishman e Lancaster (1993), Pérez-Dávila (1995), Fekete e Greening (1996), Wagner e Ressler (1997) e Downey (1999) propõem "*closed labs*", utilizando sistemas reais, na maioria dos casos, alguma versão do sistema operacional Unix".

Neste contexto, o presente trabalho descreve o estágio atual do projeto VXt – Virtual XT. O projeto, de caráter didático, consiste na implementação em software de uma máquina baseada no processador 8086 e vem sendo desenvolvido através de atividades de sala de aula, trabalhos de conclusão de curso

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 88-99.

(LINZMEIER,1999;GOEDERT,2006) e através de projetos de iniciação científica (MATTOSet al.,2004). A versão atual caracteriza-se pela facilidade de propagação da interface com o usuário para uma classe de tal forma a permitir o acompanhamento passo a passo da execução de um programa. O software vem sendo utilizado em atividades de apoio ao ensino de sistemas operacionais e tem se mostrado uma ferramenta bastante útil.

A próxima seção descreve as principais características. A seção 3 descreve a arquitetura do sistema e as técnicas utilizadas. Na seção 4 são apresentados os resultados e discussões.

2 Arquitetura do sistema

A arquitetura do VXt caracteriza-se por adotar uma solução híbrida envolvendo componentes escritos em C, Delphi e Java. A justificativa para este modelo de desenvolvimento é apresentada a seguir.

O VXt, desde a sua primeira versão, foi implementado em Delphi. Em 2003, substituiu-se a implementação do conjunto de instruções que até então era implementado em Delphi pela biblioteca SoftX86 (implementada em C na forma de DLL). Esta decisão levou em consideração as principais características da biblioteca (SOURCEFORGE, 2003): (i) precisão na emulação da CPU; (ii) tamanho da memória ajustável; (iii) projeto que modela o comportamento dos seguintes sinais de hardware (na forma de *callbacks* no programa hospedeiro): *clock*, acesso ao espaço de E/S, acesso ao espaço de endereçamento da memória, interrupções por software, interrupções de hardware. Segundo (SOURCEFORGE, 2003) a biblioteca Softx86 foi concebida de forma a viabilizar a construção de programas de emulação de hardware. A biblioteca emula somente a CPU (figura 2). A simulação dos demais componentes de hardware deve ser realizada por um programa externo – neste caso, o VXt é responsável pela construção de toda a infra-estrutura adicional (*motherboard*, memória, espaço de endereçamento de Entrada/Saída, periféricos, etc.).

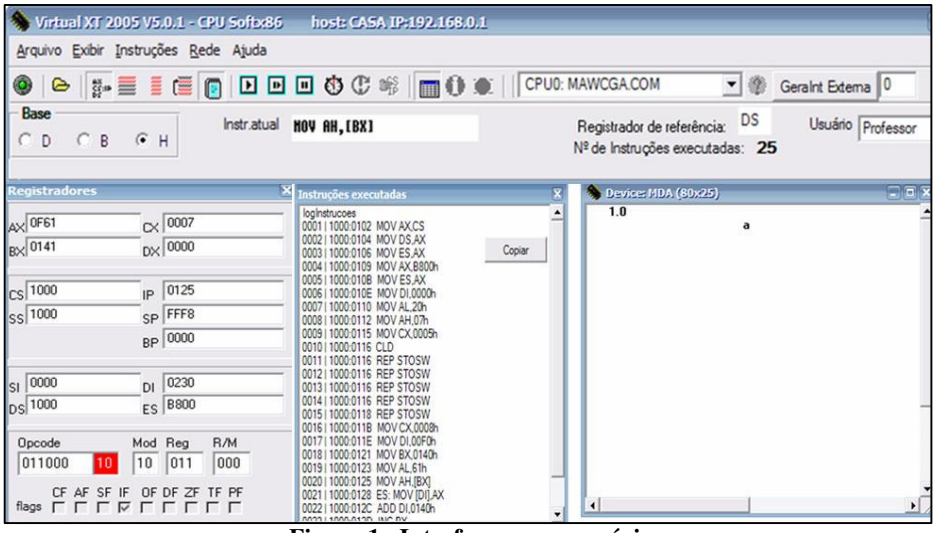


Figura 1. Interface com o usuário

O sistema foi concebido segundo o padrão *model-view-controller* (MVC) o qual oferece algumas características que fazem dele uma boa escolha de padrão a ser utilizado no desenvolvimento de software, dentre as quais, tem-se (SILVEIRA, 2006, p. 29):

a) Separar dados (*Model*) da interface do usuário (*View*) e do fluxo da aplicação (*Controller*);
b) Permitir que uma mesma lógica de negócio possa ser acessada e visualizada através de várias interfaces;
c) A lógica de negócio é independente da camada de interface com o usuário (*View*).
d) No contexto do projeto VXt, o padrão MVC foi adotado na seguinte perspectiva:
e) A versão em Delphi implementa o modelo do emulador do PC-XT e boa parte do modelo do controlador (uma vez que, o controle da aplicação ainda permanece na aplicação Delphi);
f) O *middleware* implementa parte do controlador da aplicação (uma vez que é o módulo responsável pela propagação das informações para os clientes e também por propagar os comandos de abertura/fechamento de janelas);
g) O módulo de interface do aluno implementa a camada de visão.

3 O middleware

Segundo Carvalho (2005), *middleware* é o neologismo criado para designar camadas de software que não constituem diretamente aplicações, mas que facilitam o uso de ambientes ricos em tecnologia da informação. A camada de *middleware* concentra serviços como identificação, autenticação, autorização, diretórios e outras ferramentas para segurança. Aplicações tradicionais implementam vários destes serviços, tratados de forma independente por cada uma delas. As aplicações modernas, no entanto, delegam e centralizam estes serviços na camada de *middleware*. Ou seja, o *middleware* serve como elemento que aglutina e dá coerência a um conjunto de aplicações e ambientes (CARVALHO, 2005).

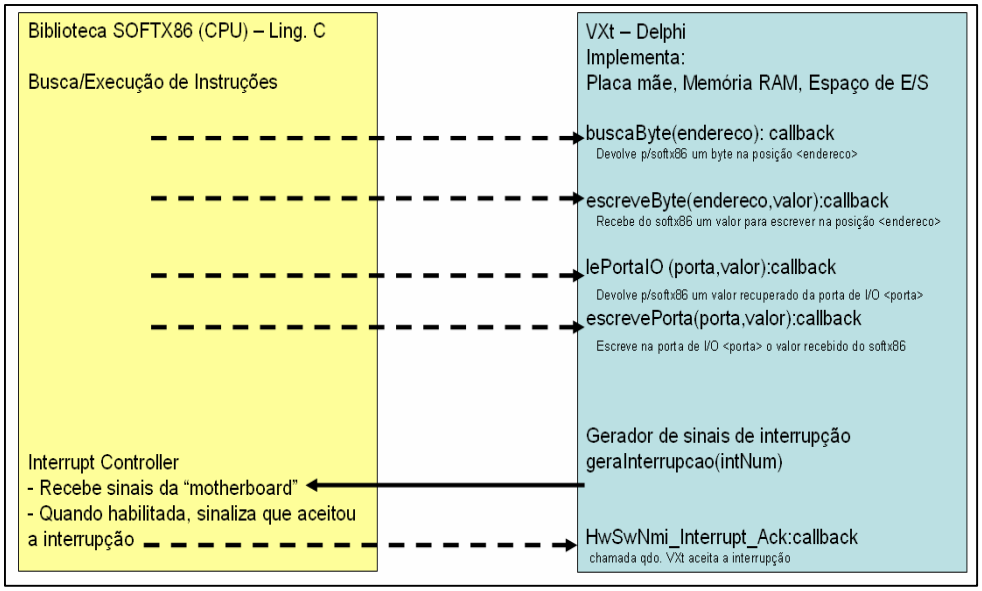


Figura 2 – Arquitetura do VXt utilizando a biblioteca Softx86.

Internamente, o *middleware* está estruturado na forma de um clássico modelo produtor/consumidor (CARISSIMI; OLIVEIRA; TOSCANI, 2003, p. 93). Ao receber uma mensagem do VXt, o servidor insere-a em um *buffer* global. Para cada cliente conectado é criada uma instância de um consumidor (*thread*) o qual, por sua vez, comunica-se exclusivamente com o seu cliente VXt Java. A figura 3 apresenta um diagrama de classes envolvendo os diversos componentes da camada *middleware*.

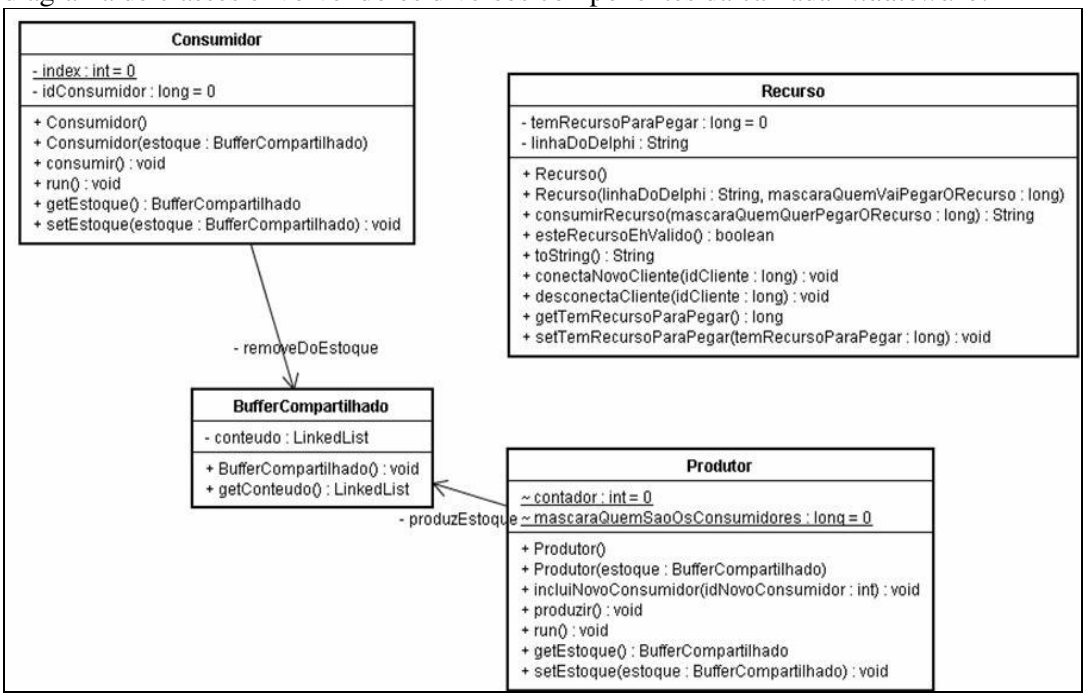


Figura 3 - Diagrama de classes do módulo middleware

A estratégia adotada para a concepção da arquitetura do *middleware* é apresentada na figura 4. Como se pode observar, o *middleware* atua como um cliente que requisita informações que o VXt em Delphi (atuando como servidor) fornece.

A estratégia baseada em produtor/consumidor viabilizou que questões de comunicação específicas de algum cliente fiquem isoladas naquele cliente não comprometendo a execução dos demais. Para operacionalizar esta estratégia, lançou-se mão de um atributo na classe do servidor chamado `mascaraQuemSaoOsConsumidores`.

Ao ser incluído um novo consumidor através de uma conexão (tardia ou não) de algum cliente Java, o servidor seleciona um identificador de cliente (a partir da variável `idNovoConsumidor`) e recupera a máscara de *bits* que o identifica realizando uma operação OU com a mascara que identifica os indivíduos atualmente conectados. Quando um cliente vai consumir um recurso do *buffer* compartilhado, ele apresenta a sua máscara de identificação e, se já não consumiu aquele recurso que se encontra no *buffer*, tem permissão de fazê-lo (Figura 5).

Inicialmente o método consumir verifica se existem itens no estoque, ou seja, se o VXt enviou uma informação para o servidor. A função deste método então, é de copiar do buffer compartilhado do servidor para o buffer da *thread* consumidora de cada cliente, que está executando o método consumir.

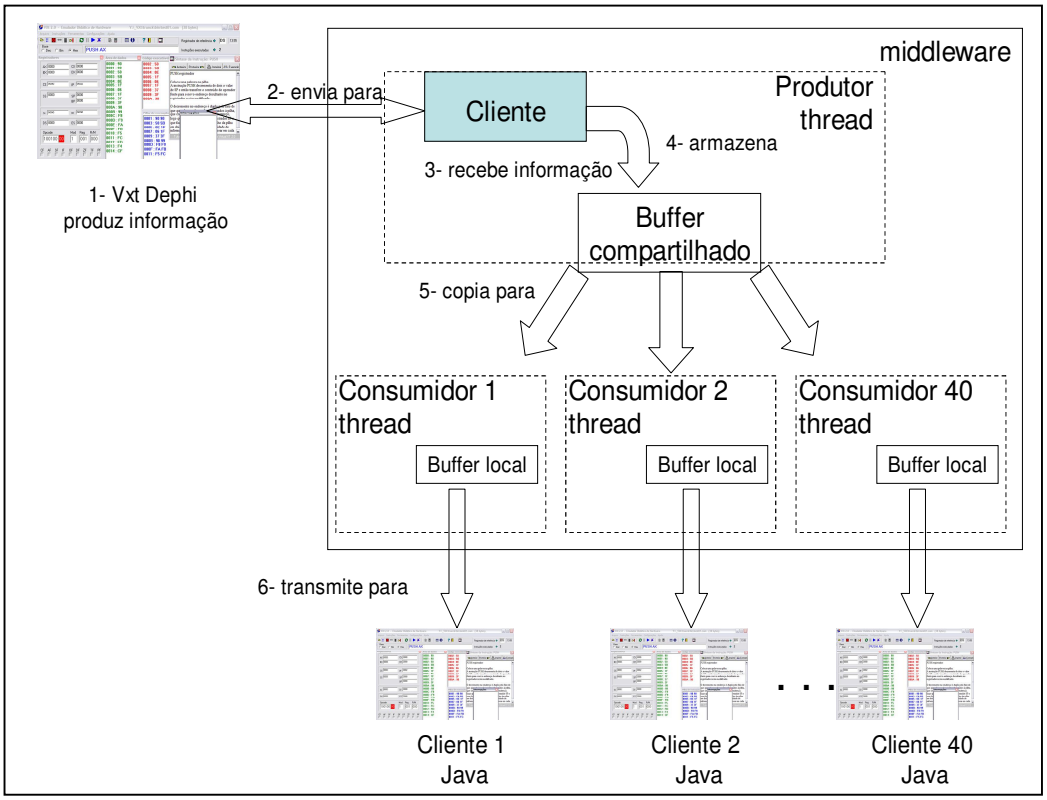


Figura 4 - Arquitetura do middleware

Para isto, executa-se um laço para permitir a este consumidor resgatar todos os recursos produzidos pelo VXt Delphi e copiá-los para o seu *buffer* local. Para isto, varre-se a estrutura de dados do buffer compartilhado, procurando por recursos que possuam na máscara de identificação um bit ligado correspondente ao identificador do consumidor atual. Se o *flag* `temRecursoParaPegar == 0`, significa que todos os consumidores já consumiram o recurso. Então, a função do último consumidor a varrer a lista, é remover aquele recurso do buffer compartilhado.

```
public interface IdentificaClientes {
    public static long codIdConsumidor[] = {
            1,                  //  00000000000000000000000000000001
            2,                  //  00000000000000000000000000000010
            4,                  //  00000000000000000000000000000100
            8,                  //  00000000000000000000000000001000
            16,                 //  00000000000000000000000000010000
            32,                 //  00000000000000000000000000100000
                                . . . . . .
            68719476736l,       //  00010000000000000000000000000000
            1374389534721,      //  00100000000000000000000000000000
            274877906944l,      //  01000000000000000000000000000000
            549755813888l       //  10000000000000000000000000000000
            };
}
```

Quadro1 - Especificação da máscara de bits de identificação das threads consumidores.

Se não houver recurso em estoque, executa uma chamada `estoque.wait()` a qual bloqueia a *thread* até que o servidor informe que há recursos para consumir no buffer compartilhado.

Todo recurso ao ser criado recebe uma máscara de *bits* do servidor

`(mascaraQuemVaiPegarORecurso)` identificando todos os clientes habilitados naquele momento a consumi-lo. Um aspecto a destacar é que, quando as *threads* compartilham acesso a um objeto comum, elas podem entrar em conflito umas com as outras (HORSTMANN, 2004, p. 777). Essa situação é denominada de condição de competição (ou *race condition*) (HORSTMANN, 2004, p. 779).

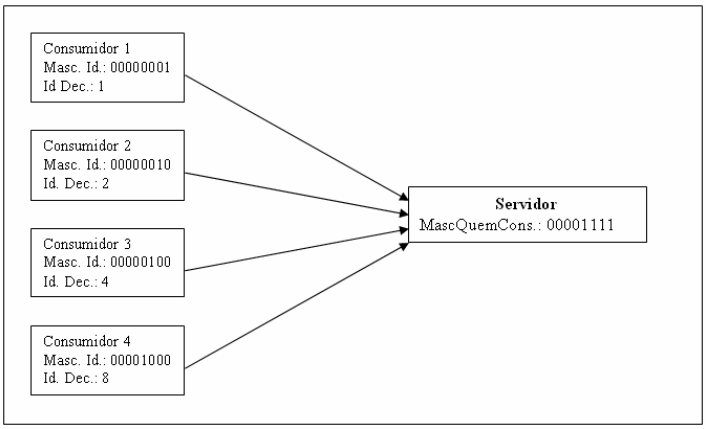


Figura 5 – Organização dos consumidores identificados

Para solucionar o problema, cada *thread* deve ser capaz de bloquear um objeto temporariamente. Enquanto a *thread* bloqueia o objeto, nenhuma outra deve ser capaz de modificar o estado deste objeto. Na linguagem de programação Java utilizam-se métodos sincronizados para bloquear objetos (HORSTMANN, 2004, p. 784). Quando uma *thread* consumiu todos os seus recursos e não tem mais o que fazer, ela chama o método `wait()` liberando o processador para que outras *threads* executem. Quando um novo recurso é adicionado ao *buffer* compartilhado, o método do servidor chama `notifyAll` para desbloquear todas as *threads* consumidoras para que elas resgatem o recurso do buffer global e transmitam para seus respectivos clientes VXt.

3.1 Os meta-forms

Sabendo-se que a aplicação VXt é desenvolvida em Delphi e que as interfaces cliente foram desenvolvidas em Java, havia a necessidade de um protocolo que viabilizasse que alterações nos formulários em Delphi fossem observadas nas interfaces cliente. Assim, desenvolveu-se o conceito de meta-forms - classes que possuem uma cópia dos campos de um formulário e que quando são notificadas da alteração de algum campo, automaticamente enviam para o servidor uma mensagem com a respectiva alteração.

Esta estratégia reduziu significativamente o volume de tráfego na rede produzido entre o VXt em Delphi e os clientes em Java, já que não é necessário que sejam enviadas as informações de todos os campos existentes no sistema a cada ciclo de execução de uma instrução.

3.2 O modelo de contexto

Uma outra característica importante é que a API da biblioteca Softx86 utiliza estruturas de contexto para representar a CPU. Estas estruturas de contexto são declaradas na

aplicação hospedeira (VXt). Com isto é possível emular mais de uma CPU simultaneamente utilizando a mesma biblioteca.

Portanto, com a incorporação da biblioteca Softx86, tornou-se possível implementar um modelo de contexto. Um contexto representa uma instância de uma máquina completa que é executada pelo VXt. Uma instância de contexto compreende:

a) uma instância de *motherboard* a qual, por sua vez, possui uma instância de `TCpu`, `TPic`, `TSO`, `TMemória` e instancias de `TDevice`;
b) `TCpu` é uma classe que viabiliza que o VXt possua uma cópia fiel do conteúdo dos registradores atualizados pela biblioteca Softx86;
c) `TPic` implementa as funcionalidades do *Programmable Interrupt Controller* (PIC);
d) `TSO` implementa um módulo (inicialmente vazio) representando o suporte necessário para a instalação de um sistema operacional na máquina;
e) `TMemória` implementa a memória RAM da máquina;
f) `TDevice` descreve as características principais de um dispositivo de hardware a ser instalado na máquina.

Sendo uma ferramenta didática, é possível que, em determinados momentos, o professor necessite apresentar outros exemplos (além daquele que está sendo utilizado) para caracterizar uma determinada situação. Assim sendo, o VXt permite que várias instâncias de contexto sejam disparadas, com a restrição de que, em determinado momento somente uma estará sendo executada. O modelo de múltiplos contextos possibilita as seguintes variações (MATTOS, et al, 2004):

a) O professor conduz a aula e os alunos acompanham a execução;
b) O professor dispara um trabalho em grupo e pode conectar-se na estação do líder de uma equipe para acompanhar o andamento do trabalho;
c) Cada aluno individualmente pode executar sua própria cópia do servidor e da interface cliente em uma máquina local podendo executar exercícios individuais.

3.3 Execução de uma instrução

A execução de uma instrução envolve uma operação de busca de um *byte* na memória, sua decodificação e efetiva execução. A busca de uma instrução de máquina é apontada pelo par de registradores CS:IP (*code segment*: *instruction point*er). A função **`softx86_decompile_exec_cs_ip(wctx)`** é utilizada para obter uma representação textual da instrução a ser executada para fins de visualização. A função **`softx86_step(wCtx)`** é quem efetivamente executa uma instrução atualizando o contexto de execução da CPU.

Ao ser executada a operação **`softx86_step(wCtx)`**, a biblioteca softx86 lança mão das rotinas de *callback* implementadas em Delphi no VXt para (a) buscar um ou mais bytes na memória do VXt (dependendo do tamanho da instrução e do número de operandos necessários) e (b) para escrever bytes na memória (quando este for o caso). A cada instrução executada com sucesso, uma cópia do conteúdo dos registradores da CPU é obtida pelo VXt de forma a atualizar do formulário específico que apresenta o valor dos registradores ao aluno. Se, após a execução de uma instrução, constatar-se que houve uma interrupção (de software ou de hardware) e, o *flag* de interrupções indica que elas estão habilitadas, então ocorre o *handshake* entre a CPU e o PIC de tal forma a

alterar os valores atuais de CS:IP para o endereço do tratador de interrupções previamente inicializado na tabela de vetores de interrupção (como ocorre no hardware real).

3.4 O acesso à memória do VXt

Conforme citado anteriormente, o acesso à memória RAM do VXt é realizado através de funções *callback*. O quadro 2 apresenta o código do procedimento para acesso de leitura e de escrita à memória do VXt.

Um aspecto que merece destaque no código apresentado no quadro 2 é a necessidade explícita de verificação sobre a tentativa de acesso à memória de vídeo. Caso o acesso à memória seja destinado a faixa de endereços onde reside a memória de vídeo, são chamados os procedimentos **`leCaracterTelaCGA`** e **`escreveCaracterTelaCGA`**, respectivamente, para ler o conteúdo de uma coordenada de tela e escrever o conteúdo em uma coordenada de tela.

```
//Funcao: Callback que retorna um byte de memória ao softx86
//-------------------------------------------------------------------------------
procedure pOn_read_memory(_ctx : pointer; address : sx86_udword; var buf : sx86_ubyte; size : Integer); cdecl;
begin
  if address < ctx.GetContextoAtual.ram.InicioMemVideo then
    copymemory(@buf,@ctx.GetContextoAtual.ram._ram[address],size)
  else
    buf := ctx.getContextoAtual.cga.leCaracterTelaCGA(address);
end;

//Funcao: Callback que escreve um byte enviado pelo softx86 na memória do VXt
//-------------------------------------------------------------------------------
procedure pOn_Write_Memory(_ctx : pointer; address : sx86_udword; var buf : sx86_ubyte; size : Integer); cdecl;
begin
  if address < ctx.GetContextoAtual.ram.InicioMemVideo then
    copymemory(@ctx.GetContextoAtual.ram._ram[address],@buf,size)
  else
    ctx.getContextoAtual.cga.escreveCaracterTelaCGA(address,buf);
end;
```

Quadro 2 – Funções callback para acesso a memória RAM do VXt.

3.5 O módulo cliente Java

A interface do cliente Java é idêntica a do VXt em Delphi, porém a sua única ação é conectar-se ao servidor. O usuário, em sua interface, somente é capaz de acompanhar a execução do VXt que estará sendo rodado numa outra máquina junto com o servidor.

Toda vez que o VXt aciona a abertura ou fechamento de uma janela, a mesma ação ocorrerá com o cliente, que abrirá ou fechará a janela correspondente em sua interface. As informações que o cliente Java recebe do *middleware*, caso não sejam responsáveis por abrir ou fechar alguma janela, são primeiramente armazenadas em um *ArrayList*, a partir do qual são recuperadas as informações na ordem em que chegaram, atualizando os campos e as janelas da sua interface, sendo posteriormente removida para a utilização da próxima informação que se encontra na fila de dados do *array*. Esse procedimento é necessário para que não se perca nenhuma informação proveniente do *middleware*, numa situação em que o servidor transmita mais rapidamente do que a capacidade do cliente de processar as mensagens recebidas. A conexão entre o cliente Java e o *middleware* acontece através do uso de *sockets*.

3.6 Comunicação via chat

O VXt também possui um recurso de *chat* para viabilizar a comunicação entre os clientes (alunos) e o servidor (professor). Este recurso permite ao servidor enviar mensagens para um cliente em específico, ou para todos os clientes que estejam conectados de uma só vez. O cliente somente pode se comunicar com o servidor.

4 O template para geração de código de sistema operacional no VXt

Até a versão anterior do VXt, a geração de código para testes era realizada através da (a) injeção manual do código executável diretamente na estrutura de dados que representa a memória (na forma de códigos hexadecimais correspondentes) ou (b)através da construção de programas em *assembly*, sua montagem e respectiva geração de código no formato .COM.

Contudo, deve-se destacar que o VXt possui suporte para a carga de programas no formato .EXE. Entretanto, tanto os programas no formato .COM como programas no formato .EXE, em geral, utilizam funções do DOS para a inicialização do ambiente de execução (*run-time*). Este fator sempre limitava a utilização do VXt porque comprometia a execução dos programas exemplo visto que, apesar da biblioteca softx86 implementar o conjunto completo de instruções do processador, a máquina VXt não possuía suporte de sistema operacional – particularmente do BIOS e do DOS. Assim sendo, a partir da primeira chamada de sistema (*system call*) via INT 21H, o programa sendo executado no VXt deixava de executar corretamente.

Tendo em vista superar esta dificuldade, trabalhou-se na concepção de uma solução que permitisse a utilização de um compilador C como ferramenta de desenvolvimento de código para execução pelo VXt e que permitisse o desenvolvimento de aplicações que num ambiente real seriam caracterizadas como embarcado.

Conceitualmente, um sistema embarcado, geralmente, consiste de um microprocessador e um pequeno conjunto de periféricos que executam um programa dedicado. Este programa é armazenado em memória não volátil e geralmente possui limitados recursos de entrada e saída. Como geralmente os sistemas embarcados não possuem um sistema operacional, o programador é responsável por prover todas as funções de entrada e saída de baixo nível. Além disso, diferentemente de programas suportados por um sistema operacional, um sistema embarcado geralmente não é carregado a partir de um disco, mas sim a partir de memórias não voláteis. Conseqüentemente, desenvolver e depurar tais sistemas é uma atividade desafiadora.

Estas características tornam os programas embarcados especialmente interessantes para a aplicação em sala de aula e o VXt é um facilitador neste processo.

Em geral, a utilização de um compilador comercial para o desenvolvimento de aplicações embarcadas requer a alteração nas rotinas de inicialização do ambiente de *run-time* deste compilador de tal forma que todas as chamadas ao BIOS e ao DOS sejam removidas. A solução adotada envolveu a construção de um pequeno núcleo de rotinas, as quais fazem esta função. Deve-se ressaltar que, no caso do VXt não se pretendeu substituir as chamadas ao DOS e ao BIOS tendo em vista que na realidade não existe um DOS nem um BIOS implementados.

O quadro 3 apresenta a parte do *template* de código C utilizado para a geração de código para o VXt.

```
typedef unsigned char far *pointer;
void inicializaTabVetoresInterrupcoesinicializaPonteiros();
void tratadorInterrupcaoSoftware();
void tratadorInterrupcaoSoftware();
void pmonta(pointer *c, unsigned int seg,unsigned int off);
void pdesmonta(pointer c,unsigned int *seg,unsigned int *off);
void main(){
 inicializaTabVetoresInterrupcoes();
 while(1){   asm {nop};   }
}
void  tratadorInterrupcaoSoftware(){
 asm{ //código que salva o contexto em execução
      push  ax;       pushbx;      pushcx;      pushdx;
      push  es;       pushds;      pushsi;      pushdi;
      push  bp
   }
// O código do tratador da interrupção desenvolvido pelo usuário é inserido aqui
 asm{// código que restaura o contexto sendo executado antes do atendimento da interrupção
      pop bp;
      pop di;       popsi;      popds;      popes;
      pop dx;       popcx;      popbx;      popax;
      iret
   }
}
void  tratadorInterrupcaoHardware(){
 asm{
   push   ax;   pushbx;   pushcx;   pushdx;
   push   es;   pushds;   pushsi;   pushdi;
   push   bp
   }
// O código do tratador da interrupção desenvolvido pelo usuário é inserido aqui
 asm{// código que restaura o contexto sendo executado antes do atendimento da interrupção
   pop     bp;
   pop     di;   popsi;   popds;   popes;
   pop     dx;   popcx;   popbx;   popax;
   popf
   iret
   }
}
```

Quadro 3 – Trecho do *template* de código para o VXt.

O quadro 4 apresenta um conjunto de rotinas utilitárias que apresentam o código necessário para a inicialização da tabela de vetores de interrupção apontando para um tratador cujo código é escrito em C.

```
/*----------------------------rotinas utilitárias ----------------------------------------------------
//funcao que monta um pointer a partir de um valor de segmento e offset
void pmonta(pointer *c, unsigned int seg,unsigned int off){
  union{
            pointer addr; unsigned int partes[2];
       }lptr;
  lptr.partes[0] = off;       lptr.partes[1]= seg;      *c =lptr.addr;
}
```

```
//funcao desmonta um pointer em segmento e offset
void pdesmonta(pointer c,unsigned int *seg,unsigned int *off){
union{
          pointer addr;    unsigned int partes[2];
  }lptr;
  lptr.addr = c;              *off = lptr.partes[0];      *seg = lptr.partes[1];
}

/*----------------------------rotinas utilitárias ----------------------------------------------------
//funcao que inicializa os ponteiros da tabela de vetores de interrupção para um tratador exemplo
void inicializaTabVetoresInterrupcoes(){
unsigned int far *ptr;
int i;
unsigned int  seg,off;
  pdesmonta((pointer)&tratadorInterrupcaoHardware,&seg,&off);
  pmonta((pointer)&ptr,0,0x00);
  off += 4; //deslocamento necessário para pular o código gerado pelo turboc
  for (i=0;i<255;i++){
     *ptr=off;
     ptr++;
     *ptr=seg;
     ptr++;
  }
}
```

Quadro 4 – rotinas utilitárias

5 Resultados e discussões

O presente trabalho descreveu o estágio atual do projeto VXt – Virtual XT. O projeto, de caráter didático, consiste na implementação em software de uma máquina baseada no processador 8086. Foram apresentadas as principais características do sistema, sua arquitetura e descrito o *template* para a geração de código de sistema operacional.

As próximas etapas do projeto envolvem o desenvolvimento de dispositivos periféricos e a continuidade no desenvolvimento de código *romable* em linguagem de alto nível. Testes já realizados permitiram o desenvolvimento de programas utilizando o compilador Turbo-C (Borland) e a carga e execução dos mesmos no VXt. Com isto pretende-se viabilizar atividades de desenvolvimento de pequenos sistemas operacionais "embarcados" os quais podem ser analisados em profundidade e de acordo com as características de cada aluno em particular.

Conforme descrito, a propagação da interface para os alunos de uma sala de aula permite ampliar as formas de utilização da ferramenta como recurso de apoio didático. Estão sendo finalizados os preparativos para disponibilizar o software para download no link (www.inf.furb.br /~mattos/vxt). Cabe destacar, também, que este é um projeto de iniciação científica e conta com apoio financeiro do programa PIBIC/FURB.

Referências

CARISSIMI Alexandre S.; OLIVEIRA Rômulo S.; TOSCANI, Simão S. **Sistemas operacionais e programação concorrente**. São Paulo: Sagra Luzzatto, 2003.

CARVALHO, Osvaldo. **GT Middleware.** [S.l.], 2005**.** Disponível em: <http://www.rnp.br/pd/gts2004-2005/middleware.html>. Acesso em: 28 ago. 2006.

CASAS, Luis A. A. **Contribuições para a modelagem de um ambiente inteligente de educação baseado em realidade virtual**. 1999. Tese (Doutorado em Engenharia de Produção) - Departamento de Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis. Não paginado. Disponível em: <http://www.eps.ufsc.br/teses99/casas/index.html>. Acesso em: 24 maio 2006.

GOEDERT, Elton. **Propagação da interface do VXt usando o modelo cliente/servidor.** 2006. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HORSTMANN, Cay. **Big Java**. São Paulo: Bookman, 2004.

LINZMEIER, Marilene. **Tutorial da linguagem Assembly utilizando o VXt**. 1999. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MATTOS, Mauro M. et al. VXt**:** um ambiente didático para ensino de conceitos básicos de sistemas operacionais e arquitetura de computadores. In: WORKSHOP DE COMPUTAÇÃO DA REGIÃO SUL, 1., 2004, Florianópolis. **Anais…** Florianópolis: Unisul, 2004. Paginação irregular.

SILVEIRA, Janira. **Extensão da ferramenta Delphi2Java-II para suportar componentes de bancos de dados**. 2006. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOURCEFORGE. **Software x86 CPU emulator library**: help. Version 0.00.0033. [S.l.], 2003. Documento eletrônico disponibilizado com o Softx86. Disponível em: <http://sourceforge.net/search/?type_of_search=soft&words=softx86>. Acesso em: 02 nov. 2006.

VEIGA, Marise S. Computador e educacão? Uma ótima combinação. In: BELLO, José L. P. **Pedagogia em foco**. Petrópolis, 2001. Disponível em: <http://www.pedagogiaemfoco.pro.br/inedu01.htm>. Acesso em: 14 jun. 2006.

Experimentos preliminares sobre o uso da reputação na formação de parcerias entre agentes

Priscilla Barreira Avegliano[1], **Jaime Simão Sichman**[1]

[1]Laboratório de Técnicas Inteligentes
Escola Politécnica
Universidade de São Paulo – Brasil

priscilla.avegliano@poli.usp.br, jaime.sichman@poli.usp.br

Resumo. *Os Sistemas Multiagentes baseiam-se na cooperação mútua de seus integrantes. Os conceitos de reputação e confiança, tão difundidos em outras áreas de pesquisa, mostram-se bastante úteis para os agentes no processo de escolha do parceiro com o qual irão cooperar. Selecionar o agente com melhor reputação pode ocasionar, entretanto, custos superiores. Este trabalho tem por objetivo analisar, por meio de uma simulação, a relação custo-benefício do uso da reputação como parâmetro na escolha de um parceiro.*

1. Introdução

A mudança do paradigma da computação centralizada para a distribuída acarretou o surgimento de novas vertentes de pesquisa na Inteligência Artificial, como a Inteligência Artificial Distribuída (IAD) e, posteriormente, os Sistemas Multiagentes (SMA).

Diferentemente da IA Clássica, que possui como foco a inteligência individual, os SMA têm como cerne a inteligência emergente das interações entre os agentes, sendo o estudo da inteligência proveniente do **comportamento social**. Em um SMA, as entidades artificiais (agentes) apresentam cada qual sua autonomia e capacidade de socialização, havendo cooperação, negociação e competição entre as entidades, da mesma forma que na sociedade humana.

Por apresentar tal semelhança, os SMA beneficiam-se do fato de tomar emprestados certos conceitos estudados nas Ciências Sociais. É o caso do conceito de reputação, que permeia os trabalhos de áreas como a Psicologia, Sociologia, Filosofia e Economia.

Recentemente, diversos modelos para o cálculo de reputação em SMA foram propostos [Rubiera et al. 2001, Yu and Singh 2002, Mui et al. 2002, Sabater and Sierra 2001, Sabater et al. 2005] com o intuito de inferir qual o agente, possível parceiro no caso de cooperação, desempenha suas ações de forma mais satisfatória.

Entretanto, buscar o melhor parceiro, ou seja, aquele com melhor reputação, pode implicar custos superiores para o estabelecimento da parceria. Este trabalho tem por objetivo analisar a relação custo-benefício na utilização da reputação como parâmetro integrante no processo de seleção de um parceiro. Para tanto, foi desenvolvida uma ferramenta de simulação denominada *RePart: Reputation-based Partnership*.

Este trabalho divide-se da seguinte maneira. Uma breve introdução da utilização do conceito de reputação em SMA e suas motivações, bem como os trabalhos correlatos

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 100-111.

serão elencados na seção 2. A seguir, o modelo selecionado para integrar os agentes utilizados na simulação, sendo responsável pelo cálculo da reputação de parceiros, será descrito na seção 3. O simulador desenvolvido e empregado no processo de simulação, bem como o experimento realizado serão apresentados, respectivamente, nas seções 4 e 5. Os resultados do experimento serão analisados na seção 6. Finalmente, na seção 7, será apresentada a conclusão do trabalho, além dos trabalhos futuros a serem desenvolvidos.

2. Sistemas Multiagentes e Reputação

Esta seção faz uma breve introdução acerca dos conceitos teóricos que justificam o uso da reputação em SMA.

2.1. Parcerias entre agentes

Tipicamente, em um SMA, os agentes possuem, cada qual, objetivos que devem ser satisfeitos. Para tanto, um conjunto de ações deve ser executado, o que conceitualiza um plano para a obtenção de um objetivo. Nem sempre um agente tem a capacidade de atuar de modo a efetuar todas as ações necessárias, tornando-se **dependente** de outros agentes [Castelfranchi 1990]. Surge, então, o conceito de parcerias, na qual um agente executa uma ação a fim de que um outro agente atinja seu objetivo em troca de um benefício posterior.

Para o estabelecimento de uma parceria, um agente deve:

1. escolher o objetivo a ser perseguido;
2. selecionar o plano para atingir tal objetivo;
3. verificar se todas as ações que compõem o plano podem ser executadas por ele;
4. caso não seja possível, escolher um parceiro que esteja capacitado a desempenhá-las.

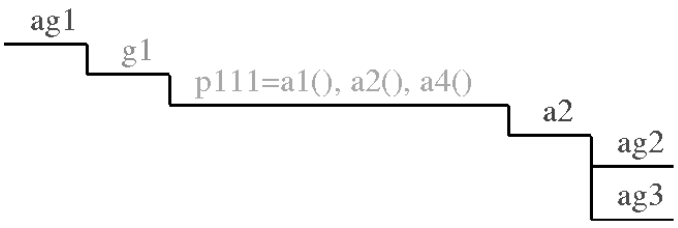


Figura 1. Grafo de Dependência [Sichman and Conte 2002] do agente ag1.

O processo de estabelecimento de parcerias pode ser melhor elucidado com o auxílio da Figura 1. Neste exemplo, o agente ag1 possui o objetivo g1. Para atingi-lo, deve executar o plano p111, que consiste das ações a1, a2 e a4. O agente ag1 não é capaz de executar a ação a2, tornando-se dependende de outros agentes para a execução de seu plano e, por conseguinte, para a obtenção de seu objetivo. Os agentes ag2 e ag3 são capazes de executar a ação a2. Se o agente ag1, em contrapartida, souber executar uma ação que beneficie qualquer um destes agentes, uma parceria entre eles pode ser firmada.

Em uma parceria, um dos agentes deve realizar uma ação que favoreça ao outro agente, sem ainda ter sido beneficiado. Se houver uma "quebra de contrato" por parte do outro agente, ou seja, se este não cumprir sua parte no acordo firmado, o primeiro agente será prejudicado. Portanto, a cooperação e o estabelecimento de parcerias representam

um risco [Barber et al. 2003]. Torna-se necessária, então, a adoção de parâmetros que exprimam o grau de comprometimento do agente em uma parceria. É o papel desempenhado pelo conceito de reputação, apresentado a seguir.

2.2. Reputação

Na sociedade humana, a reputação tem como finalidade desencorajar fraudes e trapaças, além de favorecer a cooperação entre os integrantes. As sociedades virtuais e os SMA passaram a utilizar a reputação de forma similar, ou seja, como uma forma de punir transgressores [Sabater et al. 2005].

Em SMA, os modelos para o cálculo da reputação apresentam duas abordagens [Sabater and Sierra 2005]: a cognitiva e a baseada na Teoria dos Jogos. A primeira reflete um estado mental do agente, de acordo com suas crenças internas e individuais. Nestes modelos, a reputação é inferida a partir das informações que o agente avaliador possui do alvo. Por outro lado, a reputação nos modelos baseados na Teoria dos Jogos resulta de manipulações estritamente numéricas. Estes modelos geralmente fazem uso de teorias tais como probabilidades bayesianas e Teoria de Dempster-Shafer.

Após a análise de diversos modelos computacionais de reputação, o modelo *Repage*, de caráter cognitivo, foi selecionado para integrar o simulador e será descrito na seção 3.

2.3. Trabalhos correlatos

Com a grande variedade de modelos propostos para o cálculo da reputação em SMA [Rubiera et al. 2001, Yu and Singh 2002, Mui et al. 2002, Sabater and Sierra 2001, Sabater et al. 2005], foram desenvolvidos alguns cenários e plataformas de teste para a análise de desempenho dos mesmos [Carbo et al. 2002, Fullam et al. 2005]. Estes trabalhos visam, em linhas gerais, realizar um comparativo entre os modelos propostos e determinar qual apresenta o desempenho mais satisfatório, ou seja, identifica mais rapida ou precisamente um parceiro fraudulento. Sendo assim, considera-se que os agentes escolham sempre o parceiro com a melhor reputação.

A ferramenta de simulação RePart, por outro lado, propõe uma análise do custo-benefício do uso da reputação. A reputação do parceiro em potencial é considerada, por parte dos agentes integrantes do simulador RePart, apenas como um dos parâmetros que influenciam esta tomada de decisão. O custo demandado pela parceria também é um fator relevante na escolha de um parceiro.

Analogamente ao trabalho apresentado em [Monteiro and Sichman 2005], que estuda a formação de parcerias entre agentes heterogêneos no que toca o custo, importância e utilidade de uma parceria, no simulador RePart os agentes apresentam personalidades distintas com relação ao uso da reputação no processo de seleção de um parceiro. Maiores detalhes são apresentados na seção 5.

3. Repage

O sistema *Repage: REPutation and ImAGE* [Sabater et al. 2005] é um módulo computacional para o cálculo e gerenciamento dos conceitos de **imagem** e **reputação**. Esta diferenciação é a grande inovação inerente ao modelo.

Segundo a visão defendida por Sabater *et al.* (2005), *imagem* é uma crença avaliadora sobre determinado alvo (agente), podendo ser proveniente de experiências diretas do próprio agente avaliador com o agente alvo, ou de informações sobre a imagem do agente alvo frente a agentes informantes, chamada de *imagem de terceiros*. Por outro lado, a *reputação* é uma "voz compartilhada" anonimamente entre os agentes, ou seja, é uma crença sobre o que os outros agentes dizem a respeito de determinado alvo. Conseqüentemente, *imagem* e *reputação* podem divergir (maiores detalhes podem ser encontrados em [Conte and Paolucci 2002]).

Neste modelo, os valores da imagem e da reputação são contextuais, ou seja, um agente pode possuir uma boa imagem como vendedor, mas uma imagem ruim como informante. Portanto, uma informação sobre a imagem ou reputação deve conter o alvo e o papel desempenhado por este.

Os valores das imagens e reputações são representados por tuplas de valores positivos, compostas por 5 números, que indicam o grau de aderência da avaliação a cada conjunto *fuzzy* e que somados resultam em 1, além de mais um número que indica o grau de crença na avaliação. Tais conjuntos significam *muito ruim*, *ruim*, *neutro*, *bom* e *muito bom*.

O cálculo da imagem ou reputação de um alvo é feito segundo um processo de inferência sofisticado, que se baseia na existência de predicados organizados conceitualmente em níveis interconectados. Desta forma, cada predicado possui antecessores e sucessores. Predicados de níveis mais baixos na hierarquia fundamentam os predicados do nível superior e assim sucessivamente. A arquitetura interna do modelo é apresentada na Figura 2.

No primeiro nível, os predicados ainda não foram analisados pelo agente avaliador. O predicado *Comunicação* reflete os dados provenientes de agentes informantes: a reputação do agente alvo; a imagem que o informante possui do alvo e a imagem que, segundo o informante, outros agentes devidamente identificados possuem do alvo (*imagem de terceiros*). Além disso, temos o predicado *Contrato*, que representa o acordo firmado entre os agentes, que tem seu resultado expresso no predicado *Cumprimento*. No nível subseqüente, o predicado *Resultado* é gerado a partir dos predicados *Contrato* e *Cumprimento* do nível inferior. Já o predicado *Comunicação Avaliada* é calculado balanceando-se as informações do predicado *Comunicação* com a imagem do informante.

No próximo nível, a *Comunicação Avaliada* dá subsídio a dois predicados: *Voz Compartilhada* e *Avaliação Compartilhada*. O primeiro predicado é o elemento central na determinação da reputação de um agente alvo e se origina da reputação do agente alvo, propagada. Já a *Avaliação Compartilhada* é formada a partir das imagens transmitidas entre agentes (*imagem de terceiros*). A construção da imagem de um alvo baseia-se na avaliação compartilhada do mesmo e dos resultados das parcerias efetuadas entre o agente avaliador e o alvo.

O quarto nível apresenta 5 tipos distintos de predicados: *Reputação*, *Candidato a Reputação*, *Confirmação*, *Candidato a Imagem* e *Imagem*. Como evidenciado pelos nomes, os predicados *Candidato a Imagem* e *Candidato a Reputação* não apresentam força suficiente para serem adotados como *Imagem* e *Reputação*, o que pode ser ocasionado por falta de informações ou inconsistências. Na medida em que o grau de certeza em um

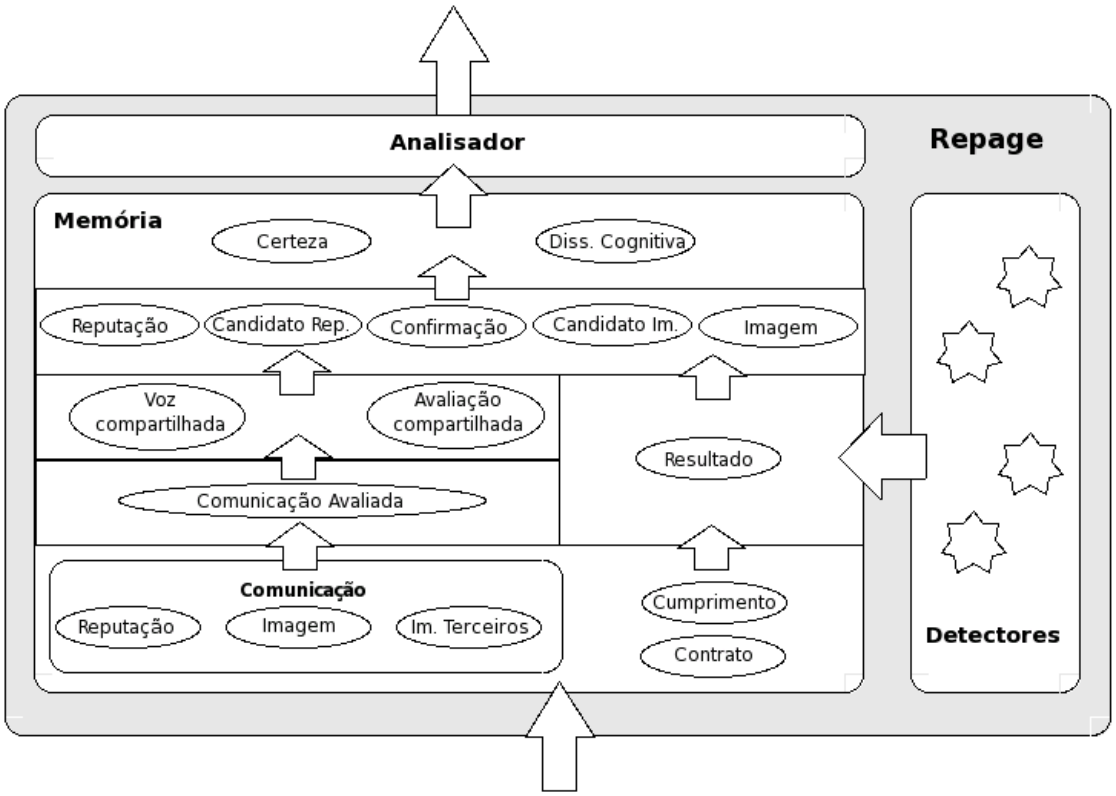


Figura 2. Arquitetura interna do modelo *Repage*. Adaptado de [Sabater et al. 2005].

candidato a imagem ou reputação aumenta e um certo nível é atingido, eles tornam-se *Imagem* ou *Reputação*. A *Reputação* (ou *Candidato a Reputação*) é obtida a partir do predicado *Voz Compartilhada*, do nível imediatamente anterior. Já a *Imagem* (ou *Candidato a Imagem*) é gerada a partir dos predicados *Avaliação Compartilhada* (nível 3) e *Resultado* (nível 2). Uma descrição mais detalhada deste modelo pode ser obtida em [Sabater et al. 2005].

O modelo *Repage* será responsável pelo cálculo de imagem e reputação por parte dos agentes que integrarão o sistema *RePart*, apresentado na próxima seção.

4. RePart

Esta seção tem por objetivo descrever a ferramenta de simulação *RePart*, desenvolvida a fim de possibilitar um estudo mais detalhado do uso de reputação como parâmetro integrante nos processos decisórios dos agentes na escolha de parceiros.

O sistema permite que parcerias entre agentes sejam estabelecidas, bem como a troca de informações entre os mesmos, tornando possível a propagação de imagens de terceiros e reputações. Por conseguinte, o simulador deve estabelecer a rede social de cada agente, determinando quais agentes poderão ser consultados.

A arquitetura do sistema *RePart* consiste em 3 componentes, conforme indica a Figura 3:

1. Agentes: são os protagonistas das parcerias, sendo cada qual autônomo para a

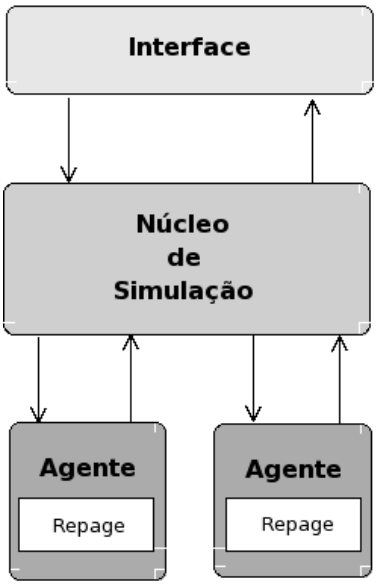


Figura 3. Arquitetura da ferramenta de simulação *RePart*.

escolha de seus parceiros. Apresentam personalidades distintas para a seleção de parceiros, definidas pelo usuário, e capacidade de armazenar imagens e reputações de agentes alvo, obtidos por meio do módulo *Repage*.

2. Núcleo de Simulação: inicia a simulação, criando os agentes de acordo com os parâmetros especificados pelo usuário, atribuindo-lhes também sua rede social. Ao longo do processo de simulação, permite a troca de mensagens entre os agentes vizinhos e é responsável pelo incremento nos passos discretos de tempo do sistema (ciclos).
3. Interface com o usuário: permite que o usuário entre com os parâmetros da simulação, tais como porcentagem de agentes com perfis distintos na sociedade, número total de agentes na simulação, número de ciclos a serem simulados etc. Também torna possível ao usuário a observação dos passos da simulação, exibindo dados sobre os agentes integrantes das parcerias, resultados das mesmas e ganhos individuais e coletivos no processo de simulação.

A ferramenta de simulação *RePart* foi utilizada para a execução de simulações do experimento descrito na seção a seguir, a fim de promover a análise da relação custo-benefício na utilização da reputação como parâmetro integrante no processo de seleção de um parceiro.

5. Descrição do Experimento

O experimento proposto baseia-se em uma economia composta por consumidores e empresas, que comercializam um bem material ou um serviço. Segundo sua estratégia de seleção de parceiro, ou personalidade, o consumidor deve contratar a empresa que mais lhe parecer conveniente. Cada empresa cobra uma certa quantia de dinheiro em troca da mercadoria. Este preço permanece inalterado ao longo dos ciclos e é definido no momento da criação da empresa por meio de uma função pseudo-aleatória, sendo um valor entre 0 e 1.

A empresa, por sua vez, entrega a mercadoria ao consumidor, apresentando, esta, uma qualidade diretamente proporcional à eficiência da empresa. Esta caracterização da empresa é atribuída pelo simulador no momento de sua criação e também é definida por meio de uma função pseudo-aleatória.

O consumidor, ao receber a mercadoria, formará uma imagem a respeito da empresa que será proporcional à qualidade da mercadoria entregue pela mesma.

De forma resumida, cada ciclo de simulação corresponde às seguintes ações, repetidas para cada consumidor:

1. Consultar agentes vizinhos sobre imagens de empresas;
2. Analisar as informações coletadas e selecionar a empresa mais atrativa;
3. Realizar o pagamento beneficiando a empresa;
4. Receber a mercadoria da empresa;
5. Avaliar a mercadoria e formar uma imagem acerca do desempenho da empresa;

5.1. Módulo de tomada de decisão

Os agentes que integram o sistema apresentam mecanismos internos para suas tomadas de decisão. No caso do experimento descrito, as empresas não apresentam nenhum tipo de decisão a ser tomada, visto que sua única função é fornecer as mercadorias aos consumidores.

Já os consumidores devem selecionar, dentre as empresas disponíveis no mercado, aquela que lhe parecer mais atrativa. A fim de responder à questão de quais são os custos associados à utilização da reputação na escolha de parceiros, foram estabelecidos os seguintes perfis para os agentes no sistema:

Conservadores que priorizam a confiança depositada no agente parceiro em potencial, em detrimento do custo que tal parceria poderia acarretar;

Ousados que ponderam o custo da parceria e a confiança depositada no agente parceiro em potencial na sua tomada de decisão;

Avaros que buscam parcerias com o menor custo, independente da confiança depositada no candidato a parceiro.

A confiança depositada em um agente será calculada segundo o procedimento de raciocínio explicitado no pseudo-algoritmo indicado na Figura 4. Note que a imagem do agente alvo sempre possui prioridade frente a reputação do mesmo, visto que uma imagem é uma avaliação proveniente de experiências diretas ou de terceiros, enquanto que a reputação é apenas uma informação sobre o que se diz na sociedade sobre determinado alvo [Pinyol et al. 2007]. Os valores da imagem e reputação dos possíveis parceiros serão calculados pelo módulo *Repage*, sendo este um módulo individual de cada agente.

Baseando-se neste cálculo, os agentes devem estabelecer qual o risco que cada parceria pode implicar, dependendo de sua personalidade.

Agentes **conservadores** calculam o risco de uma parceria segundo a equação:

$$\text{risco} = \frac{1}{\text{confiança}} \tag{1}$$

```
ImgX := imagem do agente X
Se ImgX não é vazia
Então
   Confiança := ImgX
Senão
   Confiança := RepX
Fim
```

Figura 4. Processo de determinação da confiança em um agente alvo.

Já os agentes ousados também apresentam uma preocupação com o custo demandado pela parceria. Serão mais atrativas aquelas que lhes implicarem menores custos, mantendo ainda certa relação com a confiança depositada no agente parceiro. É possível dizer que uma ação que demande um custo maior represente uma perda maior caso o "contrato" não seja cumprido. Conseqüentemente, o risco é diretamente proporcional ao custo da parceria. A partir destas bases, o cálculo do risco é feito, por parte dos agentes **ousados**, segundo a equação:

$$\text{risco} = \frac{\text{custo}}{\text{confiança}} \tag{2}$$

Os agentes avaros, em contrapartida, apenas analisam o custo que a parceria implica, em detrimento da imagem ou reputação do candidato a parceiro. Estes agentes basearão sua escolha de parceiros única e exclusivamente nos preços cobrados pelos mesmos. O risco da parceria é calculado pelos agentes **avaros** segundo a equação:

$$\text{risco} = \text{custo} \tag{3}$$

Os agentes também apresentam uma probabilidade de exploração de novos parceiros, que apresenta um decaimento exponencial. Esta probabilidade α é norteada segundo a equação:

$$\alpha = \frac{1}{numAgConhecidos} \tag{4}$$

na qual $numAgConhecidos$ indica o número de agentes com os quais o agente avaliador já efetuou parcerias.

Com probabilidade α o agente seleciona um parceiro aleatoriamente. Caso contrário, ou seja, com probabilidade $1 - \alpha$, o agente parceiro selecionado será aquele que apresentar o menor risco, calculado de acordo com a personalidade do agente avaliador.

6. Resultados obtidos

As simulações realizadas apresentavam um cenário composto por compradores e empresas divididos igualmente em número (100 agentes cada) e 300 ciclos. Foram analisadas a qualidade da mercadoria recebida pelos agentes e a quantidade de dinheiro gasta pelos mesmos a cada rodada. Os valores expressos nos gráficos representam a média destes valores por perfis, distribuídos uniformemente na população de consumidores (33% conservadores, 33% ousados e 33% de avaros).

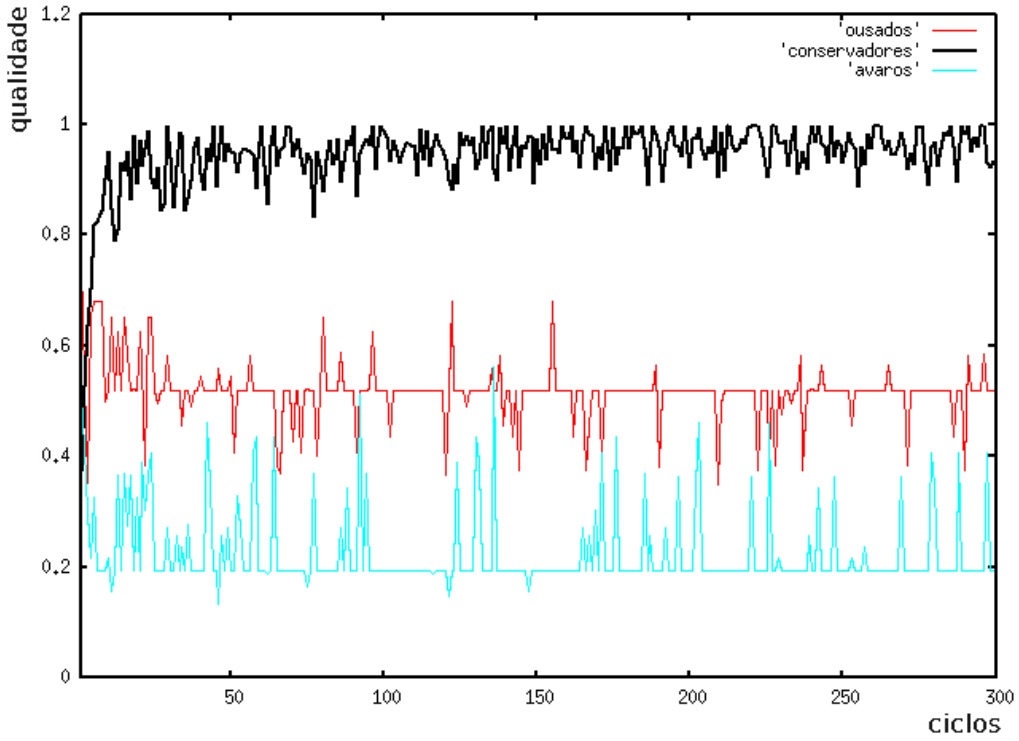


Figura 5. Qualidade de produtos recebidos pelos agentes, por perfil.

Em todas as simulações realizadas, temos que os agentes conservadores atingem sempre um patamar de qualidade de mercadoria bastante elevado (variando entre 73% e 97%). Este resultado condiz com o esperado, já que agentes com este perfil buscam exclusivamente maximizar a qualidade das mercadorias recebidas, em detrimento do custo que tal parceria poderia acarretar. A qualidade das mercadorias recebidas apresenta também uma menor variação quando comparada aos valores dos demais perfis.

Na maioria dos casos, o desempenho dos agentes ousados com relação à qualidade de mercadorias recebidas é mediano, confrome denota a Figura 5. Em alguns episódios esporádicos, dependendo da distribuição de preços e capacidades de empresas, nota-se que o desempenho dos ousados torna-se muito semelhante ao dos avaros ou dos conservadores, sem nunca superar, entretanto, o desempenho destes. O comportamento de agentes ousados é bastante variável.

Os agentes avaros, que não levam em consideração a confiança depositada no agente parceiro, apresentam sempre o pior desempenho com relação à qualidade da mercadoria recebida.

Os custos médios associados às parcerias realizadas pelos agentes a cada ciclo são apresentados na Figura 6.

É possível observar que o bom desempenho dos agentes conservadores com relação à qualidade das mercadorias recebidas eleva os custos associados às parcerias. Nos experimentos realizados, os valores da qualidade das mercadorias, para agentes conservadores, variam entre 0,73 e 0,97 (após estabilizarem), enquanto que os custos das mesmas ficam em torno de 0,08 a 0,36. Os valores pagos pelos agentes conservadores, se

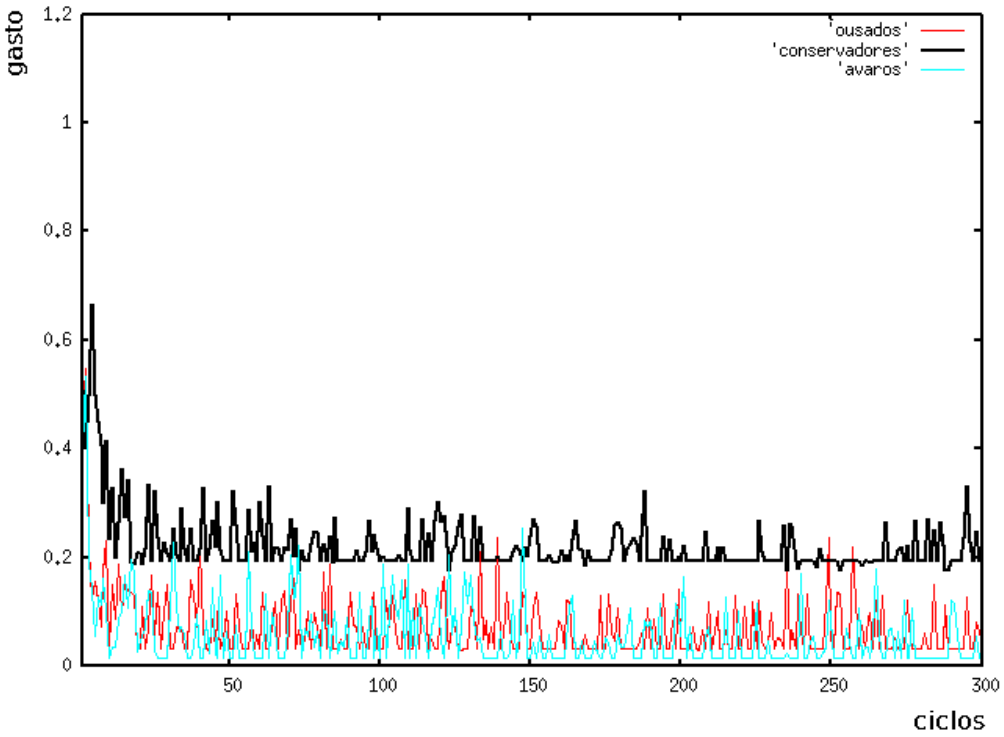


Figura 6. Custos médios por parceria.

comparados à qualidade das mercadorias obtidas, mostram-se baixos.

O uso da reputação (ou, mais especificamente, no caso do modelo *Repage*, imagem e reputação) mostrou-se eficaz na seleção de agentes parceiros, evidenciado pela qualidade superior atingida com as parcerias, sem acarretar, entretanto, em custos substancialmente elevados.

Este experimento pode ser útil em uma situação na qual um agente ag1 possui dois planos para a obtenção de um objetivo: o plano p1, que depende da ação a2 do agente ag2, que apresenta boa reputação frente ao agente avaliador (ag1), e o plano p2, que depende da ação a3 do agente ag3. Para o estabelecimento de uma parceria com os possíveis parceiros, o agente ag1 deve desempenhar ações que os beneficiem. A ação demandada pelo agente ag2 é mais custosa que a demandada pelo agente ag3. Qual é a decisão mais vantasoja para o agente ag1?

Os resultados dos experimentos indicam que, caso uma boa qualidade seja almejada e não existam grandes restrições de custo, é melhor optar pelo agente com a melhor reputação, ou seja, o agente ag2. Agindo desta forma, o agente ag1 garante um patamar de satisfação bastante elevado com a parceria, apresentando custos, que se comparados à satisfação, são relativamente baixos.

Caso o agente ag1 apresente uma restrição quanto aos custos associados à parceira, a escolha do agente ag3 pode satisfazê-la, embora a qualidade do benefício recebido pelo agente ag1 seja, muito provavelmente, baixa.

7. Conclusão e trabalhos futuros

À luz dos trabalhos desenvolvidos na área de SMA, foi proposto um experimento que visa analisar o impacto do uso da reputação, sob o ponto de vista de custos despendidos. Os resultados indicam que o uso de imagem e reputação otimizam as parcerias realizadas, sem representar custos extremamente elevados para os agentes que as utilizam, contribuindo, desta maneira, para um melhor desempenho do sistema como um todo. Além disso, os experimentos realizados também validaram a ferramenta de simulação *RePart*.

Esta ferramenta será empregada nos trabalhos futuros, que terão como foco o impacto de manipulações na informação acerca da reputação de agentes alvos. O experimento ocorrerá da mesma forma que o descrito na seção 5, com o acréscimo da chamada "publicidade" por parte das empresas. Estas se beneficiarão do fato da reputação ser uma voz anônima na sociedade e infiltrarão informações falsas sobre seus desempenhos. Com isso, espera-se analisar a dinâmica da influência de manipulações de informação.

Com o experimento apresentado neste trabalho e os futuros trabalhos espera-se analisar, respectivamente, pontos importantes em SMA, que englobam questões como qual a melhor estratégia na seleção de um parceiro e quais os efeitos gerados pela manipulação de informações em uma sociedade cujos integrantes baseiam suas escolhas, em parte, em informações anônimas.

Agradecimentos

Este trabalho foi financiado pela CAPES, por meio de bolsa de Demanda Social destinada a Priscilla Barreira Avegliano. Jaime Simão Sichman é parcialmente financiado pelo CNPq, processos 304605/2004-2, 482019/200-2 e 506881/2004-0.

Referências

Barber, K., Fullam, K., and Kim, J. (2003). Challenges for Trust, Fraud, and Deception Research in Multi-agent Systems. *Trust, Reputation, and Security: Theories and Practice*, 2631:8–14.

Carbo, J., Molina, J., and Davila, J. (2002). Comparing predictions of sporas vs. a fuzzy reputation agent system. *3rd International Conference on Fuzzy Sets and Fuzzy Systems, Interlaken*, pages 147–153.

Castelfranchi, C. (1990). Social power: a point missed in multi-agent DAI and HCI. *Decentralized AI*, pages 49–62.

Conte, R. and Paolucci, M. (2002). *Reputation in Artificial Societies: Social Beliefs for Social Order*. Springer, New York, USA.

Fullam, K. K., Klos, T. B., Muller, G., Sabater, J., Schlosser, A., Topol, Z., Barber, K. S., Rosenschein, J. S., Vercouter, L., and Voss, M. (2005). A specification of the Agent Reputation and Trust (ART) testbed: experimentation and competition for trust in agent societies. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 512–518, New York, NY, USA. ACM Press.

Monteiro, J. and Sichman, J. S. (2005). A simulator for multi-agent partnership formation based on dependence graphs. In *AAMAS '05: Proceedings of the fourth international*

joint conference on Autonomous agents and multiagent systems, pages 1223–1224, New York, NY, USA. ACM Press.

Mui, L., Mohtashemi, M., and Halberstadt, A. (2002). A computational model of trust and reputation. In *HICSS'02, Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 2431–2439, Hawaii, USA.

Pinyol, I., Paolucci, M., Sabater, J., and Conte, R. (2007). Beyond Accuracy. Reputation for Partner selection with Lies and Retaliation. In *AAMAS '07: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 134–146, Hawaii, USA. ACM Press.

Rubiera, J. C., Lopez, J. M. M., and Muro, J. D. (2001). A fuzzy model of reputation in multi-agent systems. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 25–26, Montreal, Canada. ACM Press.

Sabater, J., Paolucci, M., and Conte, R. (2005). Repage: REPutation and ImAGE Among Limited Autonomous Partners. *Journal of Artificial Societies and Social Simulation*, 9(2).

Sabater, J. and Sierra, C. (2001). REGRET: reputation in gregarious societies. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 194–195, Montreal, Canada. ACM Press.

Sabater, J. and Sierra, C. (2005). Review on Computational Trust and Reputation Models. *Artificial Intelligence Review*, 24(1):33–60.

Sichman, J. and Conte, R. (2002). Multi-agent dependence by dependence graphs. In *AAMAS 2002*, pages 483–490, Bologna, Italy.

Yu, B. and Singh, M. P. (2002). An evidential model of distributed reputation management. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 294–301, Bologna, Italy. ACM Press.

Strategy representation complexity in an evolutionary n-Players Prisoner's Dilemma model

Inácio Guerberoff Lanari Bó[1], **Jaime Simão Sichman**[1]

[1]Laboratório de Técnicas Inteligentes – Escola Politécnica
Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto, trav.3, n.158 – Cidade Universitária
05508-900 – São Paulo – SP – Brazil

{inacio.bo,jaime.sichman}@poli.usp.br

Abstract. *This work presents the results of experiments made with a spatial evolutionary model of agents playing the n-Players Prisoner's Dilemma, using two different ways to represent the agent's strategies: finite automata and adaptive automata. Since adaptive automata can represent complex strategies that cannot be represented by finite automata, comparative analysis of the co-evolution of strategies using both representations may lead to a better understanding of the role of complex strategies in evolutionary games. Here are presented the differences observed on the total utility obtained by the agents, the speed in which they converge to a nearly-stationary state, and the characteristics of the prevailing strategies.*

1. Introduction

The Prisoner's Dilemma[Poundstone 1993] is widely used as a paradigm for the study of the evolution of cooperation in a society of agents. It's adequacy for this task comes from the fact that it represents in a very simple way the dilemma faced by agents who may choose or not to cooperate with each other in a non-zero-sum game[1]. It can be described as follows:

> Two suspects of a crime are arrested by the police and interrogated simultaneously and at different rooms. Each one has two choices: remain silent or to accuse the other suspect. If one suspect accuses the other suspect and he remains silent, the one who accuses goes free, while the other receives a 10-year sentence. If both remain in silence, both spend six months in jail, and finally if one accuses the other, both receive a 5-year sentence.

It is clear that the best choice for both agents is to remain in silence, assuring that each one will spend only six months in jail. However, if we look at the decision of each agent, we see that these choices will probably not happen. From the perspective of a single agent in the game, it has only two choices: accuse (which we'll call "defect") and remain in silence (which we'll call "cooperate"). If he defects, the possible outcomes for it are to spend 5 years in jail (if the other player defects

[1]A game where the utility obtained by an agent is not exactly balanced by the utility lost by the other agent. This means that the sum of the utilities earned depends on the choices made by them.

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 112-124.

too) or to go free (if the other agent cooperates). On the other hand, if the agent cooperates, he has two other possible outcomes: spend six months in jail (if the other agent also cooperates), or 10 years (if the other agent defects). Since the agent doesn't have how to know what the other will do, the safest choice for him, no matter what the other player does, is to defect.

If we assume that both agents are rational and have the same description of the rules, the result of the game is that both defect, so they both spend five years in jail. This happens even though they could achieve a much better result if both cooperated. No cooperation is achieved because the lack of knowledge on what the other agent will do makes the expected outcome from defecting greater than cooperating. If both could make some kind of agreement, the results would be different.

In essence, the dilemma is that cooperating is good only if the other also cooperates. This situation is seen in many social and biological situations: traffic behavior, cartels in economic markets, arms race between countries, etc.

We have seen that if both agents are rational and have the same description of the rules, no cooperation is achieved in one round of the game. However, if both players play for more than one round, the situation changes. This is called the *Iterated Prisoner's Dilemma*[Axelrod 1997b]. Formally, in this version, with two participants, each one opts simultaneously each round to one between two options: to cooperate ("C") or to defect ("D"). The players get a utility R in case both play "C", and P in case both plays "D". On the other hand, in case a player chooses "C" and the other "D", the player who cooperated receives S and the one who defected receives T, where $T > R > P > S$ and $R > 1/2(T + S)$. This means that an agent has a great incentive to defect (with the perspective of earning T). However, the global result is bigger when both cooperate (because $2R > T + S$), and in fact the only Nash equilibrium[2] for the game is the one where both play "D". If the game will be repeated indefinitely, however, there is the possibility for the agents to establish mechanisms that support a mutual cooperation between them, because the lack of knowledge on the number of interactions that will occur between them makes advantageous the mutual cooperation. It is fundamental, however, that the strategies of the players have a way to deal with free-riders, through some mechanism of punishment.

Since the experiment of competition of strategies for the Iterated Prisoner's Dilemma made by Axelrod [Axelrod 1997b], many works have been made starting from the same principle - agents with heterogeneous strategies playing between them in an evolutionary environment - to analyze different configurations of games, types of strategies and other variables.

[Delahaye and Mathieu 1994] had carried through an analogous competition to Axelrod's in a modified version of the Prisoner's Dilemma, where the agents could oppose to play with the opponent, and verified that strategies with high degree of complexity presented good results in the long-term. [Ifti et al. 2004] used a con-

[2]The Nash equilibrium is combination of choices in a game where no agent can improve his outcome by changing his choice

tinuous version of the Dilemma where agents distributed in a space had only local interactions, and verified that this restriction can lead to the formation of "cooperative clusters", increasing the ratio of cooperative plays.

Other works don't start with a pre-defined set of strategies, but through genetic and evolutionary mechanisms study the development and evolution of them. [Lindgren and Nordahl 1994] use genetic algorithms and strategies with changeable memory size in a space model where the agents play the Prisoner's Dilemma. [Eriksson and Lindgren 2005] uses finite automata to represent the strategy of agents where the payoff matrices can be randomly modified.

1.1. n-Player Prisoner's Dilemma

The generalization of the Prisoner's Dilemma for more than two players (n-Player Prisoner's Dilemma, or NPPD) presents more complex situations and new challenges for the participant agents. Formally, in the NPPD the utility an agent gets depends on its own play ("C" or "D") and on the amount of other agents who cooperated in the same round. Calling $V(C|i)$ the utility earned by a player who played "C" where i other agents cooperated, and $V(D|i)$ the utility earned by the player who played "D" in the same situation, the following conditions defines the NPPD:

$$V(D|i) > V(D|i-1) \tag{1}$$

$$V(C|i) > V(C|i-1) \tag{2}$$

$$V(D|i) > V(C|i) \tag{3}$$

$$(i+1)V(C|i+1) + (n-i-1)V(D|i+1) > iV(C|i) + (n-i)V(D|i) \tag{4}$$

The inequality 1 and 2 reflect the fact that, independently of the play made by the agent, its utility will be bigger the more agents cooperate in the same round. The inequality 3 says that, for the same number of players cooperating, to play *defect* individually produces a better result than *cooperate*. Finally, inequality 4 demands that, in a group with n participants, in case a player pass from **D** to **C** (therefore, starts to cooperate), the total utility of the group increases.

In [Lindgren and Johansson 2001] are presented the following equations for V, which obeys the restrictions of the definition of the NPPD:

$$V(C|n_C) = \frac{n_C}{n-1} \tag{5}$$

$$V(D|n_C) = \frac{T.n_C}{n-1} + \frac{P(n-n_C-1)}{n-1} \tag{6}$$

The constants T and P reflects, respectively, the advantage of defecting (*temptation score*) and the punishment for the mutual defecting. Is assumed, therefore, $1 < T < 2$ e $0 < P < 1$. Finally, n_C is the total number of cooperating agents.

[Glance and Huberman 1994] shows that NPPD games have some characteristics that turns strategies based on reciprocity - as in [Axelrod 1997b] *tit-for-tat* - inefficient. This occurs because there is no possibility of, through the game, penalizing an agent without affecting the other participants. Even though, the authors show that there are two sufficiently stable equilibria for populations playing the NPPD: one with few agents cooperating and another one with many. The study of the stability of these equilibria shows, also, that when occurs transitions between these equilibria, the change will be very fast.

The role of increasing the number of players and the size of the memory used in the strategy of the agents (that is, how many results of previous plays are considered to decide the next play to be made) is analyzed by [Hauert and Schuster 1998], through many simulations using 2, 3 and 4 participants in each game and different sizes of memory. They observe that the growth in the number of participants makes it difficult to establish cooperation, however regarding the size of the memory, the relation is inverse: a minimum size is necessary so that the cooperation can be established continued.

Finally, [Lindgren and Johansson 2001] develops a model where agents distributed in space participate of NPPDs with five participants. The strategies of the agents are represented by finite automata, and new strategies appear through mutations which occurs during the inheritance phase of the evolutionary model.

This work presents a contribution to the study of the evolution of the cooperation, starting from the model proposed in [Lindgren and Johansson 2001], and introducing the use of adaptive automata for the representation of the agents's strategies. Due to its capacity to represent more complex strategies, the comparative analysis of the evolution of both models can help better understanding the role of the complexity of the strategies in these situations.

2. The Model

2.1. Evolutionary environment

The most important characteristic of the environment created for the simulations here presented is that it is evolutionary. This means that the components of the simulation (the agents) are born, reproduce and die throughout time. If we add to this the fact that the probability of reproduction is a result of some desired characteristic (as performance in some task, for example) and the possibility of mutation in the characteristics of the agent in its reproduction, we have as result a system that reproduces, in essence, the mechanism of natural selection.

Also, the environment used in this work has geographic distribution, limiting the interactions between the agents to its neighbors. When the simulation environment does not uses this restriction, generally an agent can interact with all

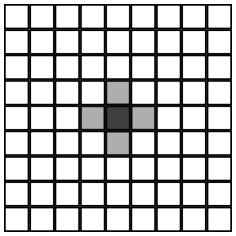


Figure 1. Grid

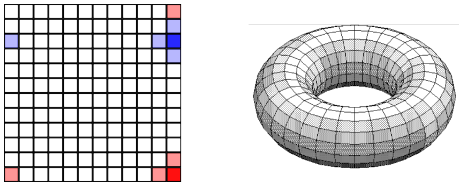


Figure 2. Neighborhood on the edge and torus

the available agents, which can modify the dynamics of the system, and also in its equilibrium results[Lindgren and Nordahl 1994].

The geographic space of this environment is represented by a grid of arbitrary size, whose bidimensional projection can be seen in figure 1. In it, each square is occupied by an agent (if the bidimensional projection has 50 cells in each column and 50 cells in each line, for example, it will have 2500 agents). Each agent has neighbors, following a neighborhood criterion. Here we will use the neighborhood of Von Neumann of degree 1, that considers neighbors the four cells immediately connected to its four sides (as shown in figure 1). The cells in the edges of the grid have neighbors on the opposite sides of the grid. Figure 2 shows two examples of this case. Considering this neighborhood in the edges, the simulation space can be better understood as a discrete *torus* (figure 2).

We have, therefore, that each agent interacts in each round with four neighbors, configuring NPPDs of 5 participants. If we define the grid as being of size mxn, for example, there will be $m.n$ games with 5 participants in each generation. As each game will be repeated, between the same participants, n_{rep} times, there will be $m.n.n_{rep}$ rounds in each generation. Since the strategies are not modified within a generation, the order of choice in which the agents are chosen to play is irrelevant. In other words, the choice of the next agent to interact with its neighbors during the simulation can be sequential or random.

After all the agents have played with their neighbors in a generation, each one will have participated in $5n_{rep}$ games, because he participates in games where it is in the center of the neighborhood as well as in the games where its four neighbors are in the center of the neighborhood. This means that, even though it has only four neighbors, in each generation an agent participates in games with 12 other agents, in groups of 5 (figure 3 shows the player in question in dark ash, the four neighbors with which participate in all $5n_{rep}$ games with horizontal crosshatches, and the eight players with which he participates indirectly through the games of its neighbors with vertical crosshatches).

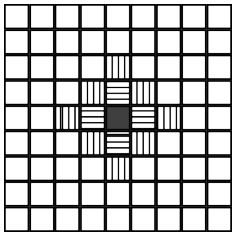


Figure 3. Extended neighborhood

At the end of each generation (when all the agents had completed its series of games), each agent accumulated a utility, resultant of the sum of all the individual results in its $5n_{rep}$ games. This utility reflects the performance that the agent got playing through its strategy. In other words, it indicates the adequacy of the strategy in providing good results for him, and will be criterion used to decide the reproduction of strategies. This way, in the end of a generation each agent compares its total utility with the one of its four neighbors. In case its utility has been bigger or equal to all the utilities earned by its neighbors, it remains with the same strategy. In the other case, he will copy (inherit) the strategy of the agent, in the same neighborhood, that earned the biggest utility. In case more than one has same punctuation, the choice will be random between them. This operation must be made in parallel for all the agents in the end of each generation (so an occurred inheritance is not propagated in the same generation).

During the inheritance a mutation can occur with a probability P_m [3]. For example, if the inheritance occurred without mutation, we would have that an agent **A**, who has the strategy E_a would inherit the strategy E_b from an agent **B**. At the end of this process, **A** would have the strategy E_b and **B** would still have the strategy E_b. However, in case a mutation occurs in this process , **B** will still be with the strategy E_b, but **A** will receive a strategy $E'_b \neq E_b$. The mutations are small changes in the structure that represents the strategy, as the addition or removal of transition or state.

2.2. Strategy representation

2.2.1. Finite Automata

The strategy of an agent must say which is the play that it must do in each round. They can be extremely simple strategies (as "always play C") up to most complex ones (as strategies based of the plays history). In this model the strategies of the agents are represented by finite automata. Automata are structures basically composed by three components:

- A set of states
- A transitions function
- A initial state

Figure 4 shows a strategy for the NPPD with 5 participants. It has three states: **D1**, **C1** and **D2**. The arcs represent the transitions, and the numbers

[3]Where P_m is the same for all agents

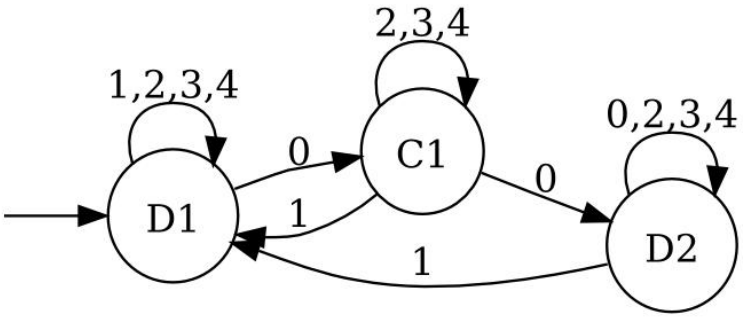


Figure 4. Strategy by finite automaton

associated with them represent the input that triggers the transition. The input is the number of other players who cooperated. The initial state is the one with an empty arrow in its direction (in this in case, it is **D1**). At every moment, the current state defines which will the play of the agent: "C" for cooperate and "D" for defect. The numbers associated to the letters of the states are only used to differentiate them.

The following sequence of plays shows how this strategy may be used:

1. At the first round the agent plays "D" (since his initial state is **D1**)
2. In this round, none of the other four players played cooperate
3. The transition "0" changes the current state from **D1** to **C1**
4. The agent plays "C" (since his current state is **C1**)
5. In this round, two other players cooperate, and the other two plays "D"
6. The transition "2" takes from **C1** to **C1** itself, so the current state is still **C1**
7. In the next round, the agent will play "C"

Each agent has its own strategy, and plays in accordance with it. Strategies represented by finite automata make possible a vast gamma of varieties, however they have limitations. Finite automata are capable to recognize only regular languages in the Chomsky hierarchy[Lewis and Papadimitriou 2004]. This means that they have the guarantee to arrive at a defined state only for a determined class of sequences. In terms of strategies for the NPPD, behaviors as learning and recognition of patterns cannot be represented through them.

2.2.2. Adaptive Automata

Adaptive automata [Neto 2001] are a class of automata that have the ability to modify its own structure, in accordance with the input they receive. These changes are executed through adaptive functions, associated to some transitions of the automaton. These functions can add or remove states and transitions to the automaton while it is used. It can be proven [da Rocha and Neto 2001] that adaptive automata have computational power equivalent to a Turing Machine, which means that, if used for representation of strategies, can present more complex behaviors than the finite automata. As an example of complex behavior, learning mechanisms (that

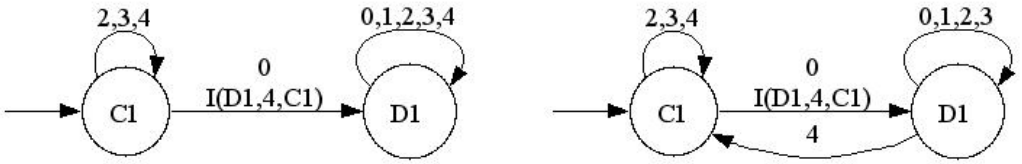


Figure 5. Adaptive automaton before and after the execution of the adaptive function

adapts the behavior of the automaton according to some input pattern) can be built with this structure[Menezes and Neto 2002].

Also, the class of adaptive automata includes the one of finite automata, because when removing adaptive functions of them we get finite automata. This is an important characteristic, that makes possible to use them naturally where the finite automata are used.

In this work, the adaptive functions are always composed by up to three actions of insertion or removal. Figure 5 shows an example. On the left side an adaptive automaton is presented, which has on the transition "0", that goes from **C1** to **D1**, an adaptive function composed by a single action of insertion. Actions of insertion are represented by letter **I** and of removal for letter **R**. The parameters are the origin state, the label of the transition and the destination state). In case the current state is **C1** and the automaton receives input "1", this action is executed, and inserts a transition from **D1** to **C1** for the event "4". Therefore, even though the initial automaton does not have a way to come back to the **C1** state being in **D1**, after the transition from **C1** to **D1** is triggered, this possibility is created.

Beyond referencing existing states, the adaptive actions can have as parameter new states. The action **I(D1,3,C_ref)**, for example, inserts a transition with label "3" that goes from the state **D1** to a new state of type "C". Thus, beyond creating new transitions between existing states, adaptive functions can create new states. It can be proved that, with this specification, it is possible to build strategies that cannot be represented by finite automata.

2.2.3. Mutations

As described in the section 2.1, new strategies appear in the population through mutations, which happens during the inheritance, at the end of a generation. The mutations can be of the following types:

1. Change the initial state of the automaton
2. Change the target of an existing transition
3. Change the type of a state (Ex: from "C" to "D")
4. Add a new state, linking it to an existing one
5. Associate an adaptive function to an existing transition

Mutations 1,2,3 and 4 are present in [Lindgren and Johansson 2001], and are enough to make possible the development of any finite automaton. Mutation 5,

on the other hand, is the responsible for more complex strategies. The adaptive functions used in this mutation are generated of randomly, having 1, 2 or 3 actions of insertion or removal.

3. Description of the experiments

To analyze the impact of the use of this new technique on the representation of strategies in the evolution of the cooperation, two experiments had been made: one where the mutation that associates adaptive functions to transitions is allowed and another one where it is not. Consequently, in the last case the strategies are equivalents to those used in [Lindgren and Johansson 2001] and in the other, strategies based on adaptive automata.

The size of the grid used was of 50x50, totalizing 2500 agents, who starts the simulation with the same strategy, composed of only one state of type "D" (that is, they never cooperate). Each simulation was composed by 2700 generations, and, to explore the characteristics of long-term of the strategies, each game in a generation is composed by 150 rounds ($n_{rep} = 150$). In the end of each generation, the automaton is returned to its initial situation (equivalent, in a finite automaton, to return the current state to its initial state and, in the adaptive automaton, to the previous structure before the changes caused by adaptive functions).

In order to represent communication failures and agent's mistakes, in 1% of the rounds the play made by the agents will be the opposite of that defined by its strategy. Also, the utility earned by the agent is deducted by a "complexity cost", proportional to the number of states that composes the strategy. With this, strategies with bigger number of states with equivalent results to those with little states will have lesser probability to be reproduced.

Since the adaptive functions are generated randomly, throughout time its action can become incorrect (for example, indicating the removal of a transition that does not exist anymore). In this case, the function is ignored during the execution.

Finally, the values of T and P of the equation 6 were defined as 1.5 e 0.25, respectively, and the mutation probability (P_m) is defined as 2.5%.

4. Results

4.1. Aggregate data

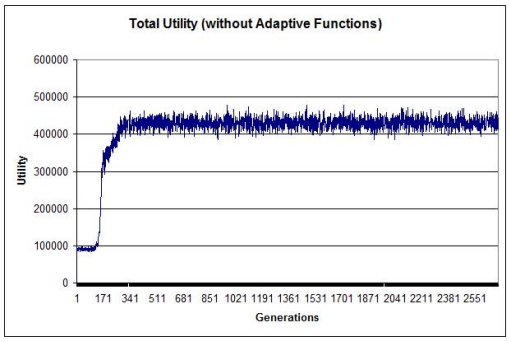
The table 1 shows the aggregate results gotten by the two simulations. It can be observed that, in both cases, there was a fast convergence to a situation of broad cooperation between the agents, even with the incentive to an individualistic and defecting behavior in the definition of the game.

Figures 6(a), 6(b), 6(c) and 6(c) show the total utility earned by the agents throughout time and the total number of plays "C" and "D" in each generation. In them we can see that, after the convergence to a situation of prevalence of cooperation, the system presents a relative stability until the end of the simulation. This means that the strategies of the agents had managed, in both cases, to keep a situation of cooperation with considerable stability.

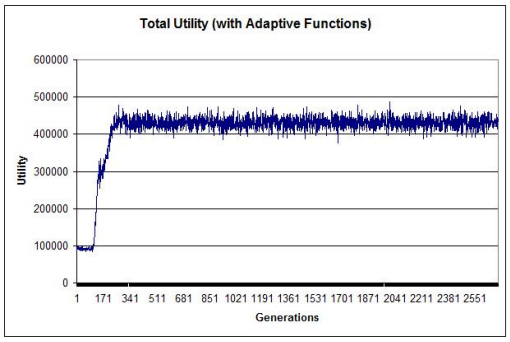
	With adaptive functions	Without adaptive functions
Average utility by generation	413313	409464
Average C plays by generation	1612881 (86%)	1597642 (85%)
Standard deviation (σ)	339877	374207
Average D plays by generation	262051 (14%)	277357 (15%)
Standard deviation (σ)	339911	374207
Number of generations until C plays prevail	128	157

Table 1. Simulation results

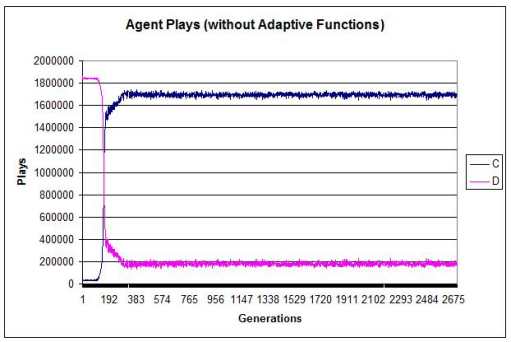
The persistent presence of about 15% of defective plays, however, indicates an equilibrium situation where those plays have an important role in the maintenance of the cooperation [4]. An analysis of the strategies of the agents can help understanding this better.



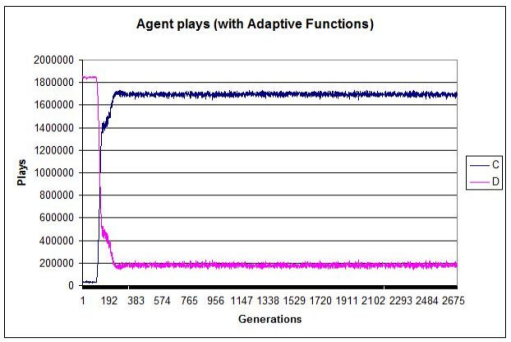
(a) Total Utility without adaptive functions



(b) Total Utility with adaptive functions



(c) Total agent plays without adaptive functions



(d) Total agent plays with adaptive functions

Figure 6. Simulation results

4.2. Analysis of the strategies

Throughout the 2700 generations, 35475 different strategies where developed on the simulation with adaptive functions and 13100 for the simulation without adaptive

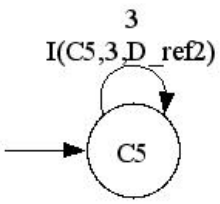
[4]It's important to stand out that random plays caused by agent's mistakes represent only 1% of the plays, not being able, therefore, to explain the observed result

functions. Even though, in the last generation the strategies with bigger population had basically the same principle. Figure 7 shows these strategies, for both cases.

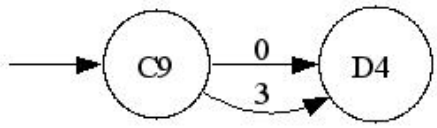
The strategy that prevails at the end of the simulation with adaptive functions (figure 7(a)) has only one state and a transition with label "3". This transition, however, has an adaptive function that inserts a new state of type "D" on this same transition. As result, its behavior can be described by two rules:

- Start playing "C", and keep this way until the result of the last round wasn't 3 (only one agent defecting)
- In case exactly one other agent doesn't cooperates, starts defecting until the end of the generation

The strategy that prevails in the simulation without adaptive functions (figure 7(b)) has two states, presenting the same behavior that in the case without them, with a difference: beyond starting defecting when one another agent defects, also does it when no other agent cooperates.



(a) Prevailing strategy for simulation with adaptive functions



(b) Prevailing strategy for simulation without adaptive functions

Figure 7. Prevailing strategies after 2700 generations

Considering also the ratio of cooperative plays observed throughout the simulation (table 1 and figures 6(c) and 6(d)), it can be inferred that the stability should be due to the "revengeful" characteristic of the observed strategies. They are based on not forgiving an agent who defects, even though this have a cost for all the other participants.

A situation where the majority of the agents uses strategies with this characteristic isn't favorable, however, to the development of other more complex ones, due to the fact that they, from the moment there was a defecting play, defects indefinitely, creating a "trap", where the success of different strategies becomes extremely difficult. The fact that this occurred in both the simulations, however, indicates that, given the initial conditions used and the characteristics of the model, the maintenance of the cooperation is only accomplished through the generations, and not during them.

We see, therefore, that even with the introduction of adaptive functions, the co-evolution of strategies did not present a different result from that where simpler strategies had been used, represented by finite automata.

5. Conclusion and future works

This work presented a model for the study of the co-evolution of strategies for the NPPD with five participants, where the strategies used by the agents are developed

and selected through a mechanism similar to the natural selection. Two forms of representation of strategies had been tested: finite automata and adaptive automata, being the last one capable of representing more complex strategies.

The comparison of the results obtained in both simulations showed that there was a fast convergence for a situation of broad cooperation between the agents, obtained in both cases through strategies of "revengeful" behavior: the cooperation is kept while all the participants cooperate. From the moment where this does not occur, it will not cooperate until the end of the generation.

It hasn't been observed, however, significant differences in the aggregate data or in the analysis of the dominant strategies on both simulations. A possible explanation for this is the fact that strategies with this "revengeful" characteristic appear quickly through few mutations, and creates a "trap" that hinders the development of different ones.

Future works include improvements on the algorithm of generation of adaptive functions, that today consists of random choices, resulting in a great number of useless functions, thus diminishing the space of possible functions (today about 32000). Simulations with a bigger number of generations will also be made, testing the stability with better precision. Tests with different configurations of initial strategies will also be made, in order to observe the impact of the initial conditions in the development of the system.

6. Acknowledgment

Inácio Guerberoff Lanari Bó is supported by CNPq, Brazil, grant number 310704/2005-7. Jaime Simão Sichman is partially financed by CNPq Brazil, grant numbers 482019/2004-2 e 304605/2004-2.

References

[Axelrod 1997a] Axelrod, R. (1997a). Advancing the art of simulation in the social sciences. Working Papers 97-05-048, Santa Fe Institute. available at http://ideas.repec.org/p/wop/safiwp/97-05-048.html.

[Axelrod 1997b] Axelrod, R. (1997b). *The evolution of cooperation*. HarperCollins.

[da Rocha and Neto 2001] da Rocha, R. and Neto, J. (2001). Autômato adaptativo, limites e complexidade em comparação com a Máquinha de Turing. *Proceedings of the second Congress of Logic Applied to Technology - LAPTEC'2000*, pages 33–48.

[Delahaye and Mathieu 1994] Delahaye, J. and Mathieu, P. (1994). Complex strategies in the iterated prisonerís dilemma. *Chaos & Society*, 94.

[Eriksson and Lindgren 2005] Eriksson, A. and Lindgren, K. (2005). Cooperation driven by mutations in multi-person Prisoner's Dilemma. *Journal of theoretical biology*, 232(3):399–409.

[Glance and Huberman 1994] Glance, N. and Huberman, B. (1994). The Dynamics of Social Dilemmas. *Scientific American*, 270(3):76–81.

[Hauert and Schuster 1998] Hauert, C. and Schuster, H. (1998). Extending the Iterated Prisoner's Dilemma without Synchrony. *Journal of Theoretical Biology*, 192(2):155–166.

[Ifti et al. 2004] Ifti, M., Killingback, T., and Doebeli, M. (2004). Effects of neighbourhood size and connectivity on spatial Continuous Prisoner's Dilemma. *Arxiv preprint q-bio.PE/0405018*.

[Lewis and Papadimitriou 2004] Lewis, H. and Papadimitriou, C. (2004). *Elementos de teoria da computação*. Bookman.

[Lindgren and Johansson 2001] Lindgren, K. and Johansson, J. (2001). Coevolution of strategies in n-person prisonerís dilemma. *Evolutionary Dynamics-Exploring the Interplay of Selection, Neutrality, Accident, and Function. Reading, MA: Addison-Wesley*.

[Lindgren and Nordahl 1994] Lindgren, K. and Nordahl, M. (1994). Evolutionary dynamics of spatial games. *Proceedings of the Oji international seminar on Complex systems: from complex dynamical systems to sciences of artificial reality: from complex dynamical systems to sciences of artificial reality table of contents*, pages 292–309.

[Menezes and Neto 2002] Menezes, C. and Neto, J. (2002). Um método híbrido para a construção de etiquetadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos. *Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, pages 19–21.

[Miller 1989] Miller, J. (1989). The coevolution of automata in the repeated iterated prisonerís dilemma. *Sante Fe, NM: Santa Fe Institute working paper*, pages 89–003.

[Neto 2001] Neto, J. (2001). Adaptative rule-driven devices-general formulation anda case study. pages 234–250.

[Poundstone 1993] Poundstone, W. (1993). *Prisoner's Dilemma*. Oxford University Press.

[Weibull 1995] Weibull, J. (1995). *Evolutionary Game Theory*. MIT Press.

Aplicação de p-Medianas ao Problema do Corte Guilhotinado Bi-Dimensional para Peças Regulares

Gilberto Irajá Müller[1], **Arthur Tórgo Gómez**[1]

[1]Universidade do Vale do Rio dos Sinos – UNISINOS
PIPCA - Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Av. Unisinos, 950 – Bairro Cristo Rei – CEP 93022-000 – São Leopoldo – RS

irajamuller@terra.com.br, breno@unisinos.br

Abstract. *This paper presents a model applied to Guillotineable Bin-packing Problem without constraints about cutting number stages. The proposed model uses p-Median for arrangement of the parts. Experiments were realized with the aim of validating the proposed model presenting interesting results and good quality in the response time.*

Resumo. *Este artigo apresenta uma modelo aplicado ao Problema de Corte Guilhotinado Bi-dimensional sem restrições quanto ao número de estágios de corte. O modelo proposto utiliza p-Medianas para o arranjo das peças. Foram realizados experimentos com o objetivo de validar o modelo proposto apresentando resultados interessantes e de boa qualidade no tempo de resposta.*

1. Introdução

O Problema de Corte e Empacotamento tem sido estudado nas últimas décadas objetivando auxiliar diversas indústrias a minimizar a perda de matéria-prima, tais como: vidro, metal, couro, entre outros [Fritsch and Vornberger 1995]. Essas indústrias, normalmente têm seus processos de embalagem e cortes automatizados e necessitam de um Plano de Corte referente ao material utilizado de forma a minimizar o desperdício de material. O Problema de Corte e Empacotamento consiste em descobrir o melhor arranjo de um conjunto de peças de dimensões diversas em um objeto com dimensões maiores. Esse é um problema de otimização combinatória sendo classificado como NP-Hard [Garey et al. 1979].

No Brasil, uma iniciativa para associar o trabalho científico à prática industrial foi proposta por um consórcio constituído pelo Instituto Nacional de Pesquisas Espaciais (INPE), Instituto Tecnológico de Aeronáutica (ITA), Universidade de São Paulo (USP) e a Universidade Federal de São Carlos (UFSCar), que iniciou um projeto para Problemas de Cortes e Empacotamento [CEAC 1999] e vem sendo amplamente pesquisados. Esse projeto tem como objetivos: desenvolver programas que resolvessem problemas industriais de natureza combinatória, achar aplicabilidade dos resultados obtidos das pesquisas e integrar pesquisadores de todo o Brasil que já desenvolviam trabalhos na área de Corte e Empacotamento.

Para o problema de corte guilhotinado, de acordo com Christofides e Hadjiconstantinou [Christofides and Hadjiconstantinou 1995], é um problema mais específico do Problema de Corte e Empacotamento e consiste em um conjunto de peças retangulares pequenas, com tamanhos diversificados e um objeto retangular principal, sendo que o objetivo é maximizar o valor de peças cortadas através do arranjo destas peças no objeto

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 125-133.

principal, sujeito à restrições quanto ao número de peças e número de estágios.

De forma a tratar o problema supracitado, é desenvolvido um modelo baseado no problema clássico de p-Medianas. Dessa forma, este trabalho possui a seguinte organização: na Seção 2 é apresentado o problema do corte guilhotinado; na Seção 3 é descrito o modelo baseado nas p-Medianas; na Seção 4, apresenta-se os experimentos e, por fim, na Seção 5, é apresentado a conclusão deste trabalho.

2. O problema do corte guilhotinado

O tipo de corte abordado neste trabalho é do tipo guilhotina e, tanto as peças como os objetos são do tipo retangular, o que se pode afirmar que o corte será do tipo ortogonal, ou seja, paralelo um dos lados do objeto principal [Christofides and Hadjiconstantinou 1995]. A Figura 1 ilustra o corte guilhotinado e o não-guilhotinado.

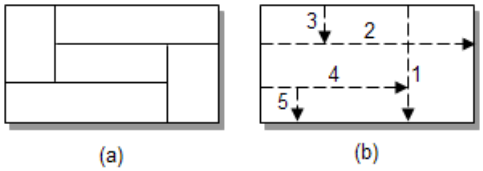


Figura 1. (a) Corte não-guilhotinado (b) Corte guilhotinado com os estágios de corte.

Neste trabalho busca-se a minimização da área de desperdício interno entre as peças: caso duas peças sejam colocadas lado a lado, ou uma em cima da outra e estas apresentarem lados adjacentes compartilhados, não haverá espaço desperdiçado entre elas e o espaço será computado como "0". Existem algumas hipóteses em relação ao problema, sendo elas:

- A quantidade de itens de uma determinada peça terá um limite para cada plano de corte projetado e é definido como corte restrito [Daza et al. 1995];
- Os objetos deverão estar em estoque, o que significa dizer que serão suficientes para a demanda de peças a serem cortadas;
- As máquinas que irão fazer o corte deverão ser do tipo guilhotina, em que a lâmina quando faz o corte, percorre o objeto de uma extremidade a outra dividindo o objeto em duas partes;
- A medida que corresponde a espessura da lâmina no objeto não será observada neste trabalho;
- Neste modelo propõe-se o corte não estagiado, pois não há restrições quanto ao número de estágios de corte;
- Tanto as peças quanto os objetos, deverão ser representados por duas dimensões altura e largura e sua área o produto das duas dimensões;
- Assume-se para este trabalho que os valores correspondentes às dimensões terão números inteiros;
- Durante a projeção das peças ocorrerá o corte normalizado, ou seja, deverão tocar-se de forma parcial ou total [Christofides and Hadjiconstantinou 1995];
- Cada peça possui um identificador único;
- As peças podem ter dimensões homogêneas, ou seja, a mesma altura e a mesma largura;

- As peças podem girar num ângulo de 90º [Morabito and Arenales 1995];
- As dimensões do objeto são observadas como o limite;
- As peças não podem ser sobrepostas;
- A primeira peça é colocada no canto esquerdo inferior do objeto.

3. Problema das p-Medianas

P-Medianas é um problema clássico de otimização combinatória que tem como objetivo construir um conjunto de elementos em que a associação de um novo elemento ao conjunto (medianas), é feita através dos pesos entre o último elemento associado e os elementos que ainda não foram associados ao conjunto, de forma que o escolhido proporcione o melhor benefício, segundo um critério pré-determinado [Lorena and Senne 2003].

Pode-se representar o problema das p-Medianas através de um grafo em que as instalações possíveis candidatas à demanda são os vértices, a ligação entre as instalações são as arestas e o peso é um benefício entre elas. Considere $G = (V, A)$ um grafo onde V são os vértices e A as arestas: determine um conjunto de medianas Vp com cardinalidade p, de forma que o somatório dos pesos de cada vértice das demandas $V - Vp$ até seu vértice mais próximo em Vp seja minimizada [Corrêa 2000]. O conjunto de p instalações que forma uma solução são chamados de p-Medianas [Corrêa 2000].

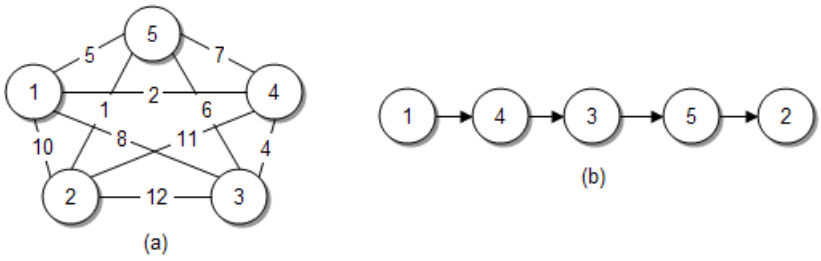


Figura 2. (a) Grafo p-Mediana (b) p-Mediana gerada após análise dos pesos.

A Figura 2(a) apresenta um grafo com um conjunto de peças onde os vértices do grafo são os identificadores das peças e os números internos nas arestas mostram o desperdício interno entre as peças. Na Figura 2(b) é apresentada a seqüência de peças após a avaliação dos pesos.

A seguir é apresentado a formulação clássica para o problema de p-Medianas, onde: n é o número total de vértices do grafo; a_i é a demanda do vértice j; d_{ij} é a distância do vértice i ao vértice j; p é o número de instalações utilizadas como medianas; $x_{ij} = 1$ se o vértice i for designado ao vértice j, $x_{ij} = 0$, caso contrário; $y_{ij} = 1$ se o vértice j for uma instalação utilizada como mediana e $y_{ij} = 0$, caso contrário.

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} a_i \, d_{ij} \, x_{ij} \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad para \; i = 1, 2, \ldots, n \tag{2}$$

$$x_{ij} \leq y_j \quad para\ i, j = 1, 2, \ldots, n \tag{3}$$

$$\sum_{j=1}^{n} y_j = p \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\} \quad para\ i, j = 1, 2, \ldots, n \tag{5}$$

A função objetivo (1) minimiza a soma dos pesos dos vértices das demandas das instalações; a restrição (2) assegura que todos os vértices farão parte da mesma mediana. A restrição (3) proíbe que um vértice de demanda seja designado a uma instalação que não seja mediana e, na restrição (4), é definido o total de vértices de instalações selecionadas. Por fim, a restrição (5) garante a não-negatividade.

3.1. Modelo utilizando p-Medianas

Para o modelo com p-Medianas foi desenvolvida uma formulação para tratar o problema considerando as hipóteses supracitadas, onde: CD é a soma da área total das peças projetadas com a área total de desperdício interno; N = número total de peças; W = altura do objeto principal; L = largura do objeto principal; i = índice para a peça; l_i = altura da peça i; w_i = largura da peça i;

$$min\ Z(x) = CD - \sum_{i=1}^{N} l_i w_i; \quad l_i, w_i \in I\!N^* \tag{6}$$

$$0 \leq CD \leq WL; \quad W, L \in I\!N^* \tag{7}$$

A função objetivo (6) busca a minimização e representa a área total de desperdício interno do objeto principal. A restrição (7) assegura que a área total das peças projetadas não ultrapasse a área do objeto principal.

3.1.1. Arquitetura

A arquitetura do modelo está estruturada em três módulos que descrevem as etapas de um processo de Corte Guilhotinado Bi-Dimensional para peças Regulares. A Figura 3 ilustra a arquitetura.

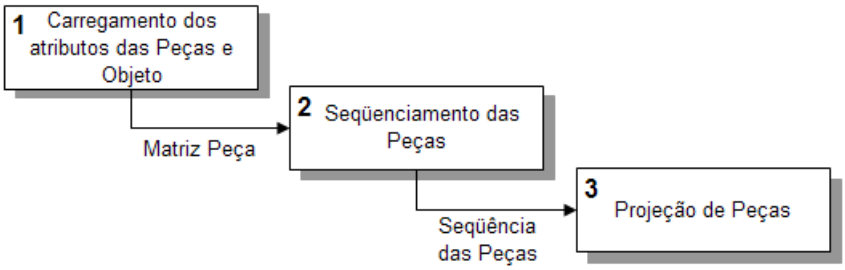


Figura 3. Arquitetura do Modelo.

No primeiro módulo é feita a leitura dos dados das dimensões das peças e dos parâmetros de entrada como quantidade de peças. No segundo modulo é feita a construção da seqüência em que as peças serão projetadas e, no terceiro módulo, é projetado as peças bem como gerado o plano de corte.

Neste modelo assume-se que as peças somente serão projetadas em um sentido: (i) ou no sentido horizontal, sendo uma peça ao lado da outra, (ii) ou no sentido vertical, uma peça em cima da outra; no módulo de seqüenciamento de peças é gerado a ordem em que as peças serão projetadas. O módulo é baseado no algoritmo de p-Mediana ilustrado no Algoritmo 1, onde as peças são projetadas no objeto segundo a avaliação de um peso que representa o melhor encaixe entre a última peça projetada e àquelas que ainda não foram projetadas.

No Módulo Projeção de peças são armazenadas as coordenadas das peças com base na seqüência e nas dimensões das peças projetadas e é gerado o plano de corte.

$P = Conjunto\ de$ ***peças*** $projetadas$;
$Carregar\ matriz\ de$ ***Peças*** A;
$i = Seleciona\_$***Peça**$\_Maior\_Area(A)$;
while $(A \neq \emptyset)$ **do**
  $Atualizar\ P$;
  $menorpeso = MAXINT$;
  $j = 1$;
  $projetar = \emptyset$;
  while $(j \leq$ ***Peças*** $não\ Projetadas)$ **do**
    if $menorpeso > Calcula\_Peso\_$***Peças***$(A, i, j)$ **then**
      $projetar \leftarrow j$;
      $menorpeso \leftarrow Calcula\_Peso\_$***Peças***$(A, i, j)$;
    endif
    $j \leftarrow j + 1$;
  endw
  $Projeta\_$***Peça**$\_no\_Objeto(A, i, projetar)$;
  $i \leftarrow projetar$;
endw
$Plano\_Corte(P)$;

Algoritmo 1: Algoritmo p-Mediana.

O Algoritmo 1, após carregar as informações das peças e do Objeto (Módulo 1), executa o procedimento *Seleciona_peça_Maior_Area*, onde é escolhida entre todas as peças àquela que tiver a área maior calculada pelo produto da largura w_i e altura l_i.

O algoritmo objetiva a próxima peça entre as outras que não fazem parte do conjunto p de medianas e, para isso, analisa qual peça possui o melhor encaixe com a última peça colocada, ou seja, escolhe das restantes a que possuir o menor desperdício interno em relação a peça i. O procedimento (Módulo 2) percorre toda a matriz de peças a serem projetadas até que a última peça faça parte do conjunto p de medianas.

No procedimento *Calcula_Peso_peças* é determinado o peso que representa o desperdício interno entre as peças, analisando as dimensões das duas peças (i,j) e qual o melhor arranjo entre elas.

O procedimento *Projeta_peça_no_Objeto*, em que após terem sido avaliados os pesos e escolhida a seqüência das peças anteriormente (Módulo 2), é feita a projeção das peças no objeto observando os limites das dimensões deste. A primeira peça é colocada no canto esquerdo inferior, a segunda a sua direita e assim sucessivamente, formando "tiras" de peças até o limite lateral. Caso a peça não caiba nessa tira, passa a ser a primeira peça da tira superior seguinte, até que todas as peças sejam projetas ou o limite superior do objeto seja alcançado.

À medida em que as peças são encaixadas no objeto, suas coordenadas são armazenadas. A altura de uma tira de peças corresponde a maior altura das peças projetadas em uma tira de peças e a largura da tira corresponde a soma das larguras das peças já projetadas nela. Neste procedimento as coordenadas são armazenadas tendo como referência a base da tira, a altura e largura da peça. O primeiro estágio do corte é paralelo à largura do objeto e corta primeiro uma tira, o outro estágio paralelo à altura do objeto e mais um corte caso exista diferença nas dimensões das peças. Por fim, o procedimento *Plano_Corte* visualiza o plano de corte gerado através do módulo 3.

4. Experimentos

De modo a validar o seqüenciamento e plano de corte, utilizamos o cenário proposto por Oliveira e Ferreira [Oliveira and Ferreira 1990] ilustrado na Tabela 1.

Tabela 1. Cenário de Validação.

Peça	Altura	Largura
1	18	22
2	29	8
3	19	19
4	13	16
5	29	8
6	16	4
7	13	16
8	18	22
9	19	19
10	29	8

A primeira peça a fazer parte do conjunto das medianas(peças projetadas), será a que tiver a maior área. As próximas peças serão determinadas pela menor área de desperdício (peso). A cada peça colocada no objeto principal são analisadas as peças restantes buscando-se a peça com o menor desperdício interno. As peças são colocadas uma ao lado da outra representando tiras. Quando o limite lateral não permitir que uma nova peça seja colocada, é iniciada outra tira de peças. Após formar a seqüência de peças, são armazenadas as coordenadas de cada corte que pertence ao Plano de Corte.

Na Figura 4 é apresentada a matriz de pesos após a construção da p-Mediana com a seqüência de projeção das peças representada na linha da matriz bem como o seu par representado pela flecha. A função objetivo foi de 312 com um tempo de execução de 0,01 segundos.

Ordem p-Mediana \ Peças	1	2	3	4	5	6	7	8	9	10
1	X	242	22	26	242	8	26	→0	22	242
8	X	242	22	26	242	→8	26	X	22	242
6	X	52	12	→0	52	X	0	X	12	52
4	X	169	39	X	169	X	→0	X	39	169
7	X	169	→39	X	169	X	X	X	39	169
3	X	190	X	X	190	X	X	X	→0	190
9	X	→190	X	X	190	X	X	X	X	190
2	X	X	X	X	→0	X	X	X	X	0
5	X	X	X	X	X	X	X	X	X	→0

Figura 4. Matriz de pesos após aplicação do Modelo p-Medianas.

Na Figura 5 é ilustrado o plano de corte após a execução do modelo com p-Medianas em que o primeiro corte separa o objeto em duas tiras de peças, e os próximos cortes separam as demais peças. Pode-se observar que a seqüência que representa o conjunto das peças projetadas é P={1,8,6,4,7,3,9,2,5,10}.

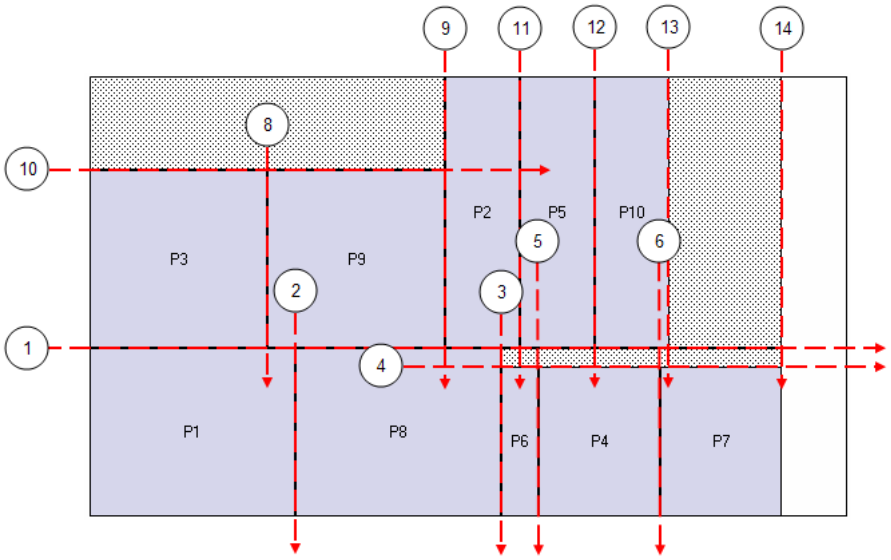


Figura 5. Plano de corte.

Na Tabela 2 é apresentado um comparativo entre os resultados obtidos por Daza [Daza et al. 1995] referente a oito problemas.

Tabela 2. Abordagem proposta por Daza et al. x p-Mediana

	Cenário(N)	Desperdício (Daza et al.)	Desperdício p-Mediana	%	tempo (seg)
p1	16	0	7	100,00	0,01
p2	41	59	108	83,05	0,06
p3	23	43	70	62,79	0,01
p4	24	31	50	61,29	0,01
p5	9	0	4	100,00	0,01
p6	27	0	15	100,00	0,01
p7	24	8	20	150,00	0,01
p8	25	34	72	111,76	0,01

O modelo p-Mediana apresentou boas respostas em virtude da avaliação com as peças restantes a cada iteração do algoritmo, ou seja, pela simplicidade dessa heurística e por não possuir uma geração de vizinhança complexa, é extremamente rápido e possui boa qualidade no tempo de resposta. Porém, os resultados dos experimentos não atingiram o desperdício interno obtido pelos cenários escolhidos na literatura conforme ilustrado na Tabela 2 em que sofreram variações de 61,59% a 150,00%. Um dos motivos que pode se observado é que o conceito clássico das p-Medianas permite que a próxima peça a ser projetada seja sempre avaliada com a última peça projetada e não pelo conjunto já projetado.

5. Conclusões

Este trabalho apresentou a aplicação de p-Medianas ao problema do corte guilhotinado. Foi apresentado um modelo que permite gerar eficientemente uma proposta de plano de corte onde é aplicado o conceito de p-Medianas. Comparando-se o modelo p-Mediana em que são implementados o mesmo conjunto de problemas, o modelo gerou resultados com áreas de desperdícios maiores do que o abordado por [Daza et al. 1995], pois o algoritmo de p-Mediana avalia a última peça colocada no conjunto com a próxima a ser considerada, enquanto o algoritmo proposto por [Daza et al. 1995] avalia o conjunto das peças já projetadas com a próxima da seqüência através de um mecanismo de geração de vizinhança que permite diversificar e intensificar a solução.

A restrição das p-Medianas faz com que as próximas peças sejam colocadas uma ao lado da outra, caso contrário, haveria muita fragmentação de desperdício interno. Como extensão futura deste trabalho, sugere-se o aprimoramento dos métodos de p-Mediana de forma a considerar todo o conjunto de peças projetadas através de um mecanismo mais eficiente de geração de vizinhança.

Referências

CEAC (1999). Cortes e empacotamento assistidos por computador. `http://www.lac.inpe.br/po/projects/ceac/ceac.html`. Acesso em 12 dez 2006.

Christofides, N. and Hadjiconstantinou, E. (1995). An Exact Algorithm for Orthogonal 2D Cutting Problems using Guillotine Cuts. *European Journal of Operational Research*, 83:21–38.

Corrêa, E. S. (2000). Algoritmos Genéticos e Busca Tabu Aplicados ao Problema das p-Medianas. *Universidade Federal do Paraná, Dissertação de Mestrado*.

Daza, V. P., Alvarenga, A. G., and Diego, J. (1995). Exact Solutions for constrained two-dimensional cutting problem. *European Journal of Operational Research*, 84:633–644.

Fritsch, A. and Vornberger, O. (1995). *Cutting Stock by Iterated Matching*. Universität Osnabrück.

Garey, M. R., Johnson, D. S., and Sethi, R. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. New York; Oxford: Freeman.

Lorena, L. A. N. and Senne, E. L. F. (2003). Abordagens Complementares para Problemas de p-Medianas. *Revista Produção*, 13(3):78–87.

Morabito, R. and Arenales, M. (1995). An and/Or-Graph approach to the solution of two dimension non-guillotine cutting problems. *European Journal of Operational Research*, 84:599–617.

Oliveira, J. and Ferreira, J. (1990). An improved version of Wang´s Algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, 44:256–266.

FERRAMENTA DE CONSTRUÇÃO DE DATA WAREHOUSE

Maurício Capobianco Lopes[1], Percio Alexandre de Oliveira[2]

[1]Departamento de Sistemas e Computação - Universidade Regional de Blumenau (FURB)
Rua Braz Wanka, 238 – 89.035-260 – Blumenau, SC – Brasil
mclopes@furb.br, percio@inf.furb.br

RESUMO: *O Data Warehouse é uma solução que procura de maneira flexível e eficiente tratar grandes volumes de dados e obter informações que auxiliem no processo para tomada de decisão. Assim, este artigo apresenta uma ferramenta com foco a usuários e projetistas de data warehouse, visando reduzir custos no processo de construção deste ambiente. A ferramenta foi desenvolvida em Java garantindo a portabilidade de seu sistema que implementa as principais fases de um projeto de data warehouse: extração, transformação e carga dos dados; visualização, análise e tratamento das informações.*

Introdução

A crescente competição em mercados cada vez mais dinâmicos está levando as empresas a tomarem decisões mais rapidamente. Sendo assim a informação tornou-se o bem mais valioso dentro das instituições. Os administradores tomam suas decisões com base na análise de dados objetivos, sintetizados e confiáveis acima de tudo, sempre com o intuito maior de melhorar e aperfeiçoar processos internos. É dentro deste cenário que hoje se torna imprescindível a utilização de recursos computacionais para levantar e analisar as informações necessárias. Uma das principais ferramentas que constitui a nova geração de Sistemas de Apoio a Decisão (SAD) é o Data Warehouse (DW), um banco de dados específico para propósitos gerenciais e estratégicos (DW BRASIL, 2005).

Para Kimball (1998 apud COME, 2001, p. 2), DW é o lugar onde as pessoas podem acessar seus dados. A abordagem de Ralph Kimball veio com um estilo mais simples e incremental, baseado na metodologia estrela que aponta para *Data Marts* (DM) separados, que deverão ser integrados na medida da sua evolução (BARBIERI, 2001). Já Wang (1998 apud COME, 2001, p. 2) tem uma definição um pouco mais elaborada quando diz que DW é o processo pelo qual os dados relacionados de vários sistemas operacionais são fundidos para proporcionar uma única e integrada visão de informação de negócios que abrange todas as divisões da empresa.

Assim, este trabalho apresenta uma ferramenta de DW para auxiliar seus usuários no processo de transformação de dados operacionais em informações gerenciais, viabilizando consultas em diversos níveis de detalhe. A ferramenta é totalmente executável em ambiente web, sendo, desta forma, acessível através dos principais navegadores hoje disponíveis no mercado. Trata-se, portanto, de uma ferramenta genérica para a construção e implantação de um DW, sem perder de vista sua

Grahl, E. A; Reis, D. S. (Eds.). Anais do XVI Seminário de Computação, Blumenau, 3-5 de outubro, 2007. p 134-146.

usabilidade, dando suporte tanto ao projetista do DW quanto ao seu usuário. Todos os detalhes de especificação e desenvolvimento desta ferramenta estão disponíveis no trabalho de Oliveira (2007).

DATA WAREHOUSE

O termo Data Warehouse significa armazém de dados. É definido como um ambiente que provê informações de suporte à decisão que, no ambiente operacional, se tornariam difíceis de serem obtidas. Em outras palavras, pode ser comparado como um banco de dados especial, estruturado de forma a facilitar o processamento para análise dos dados.

O conceito de DW surgiu da necessidade de integrar dados corporativos espalhados em diferentes máquinas e sistemas operacionais, para torná-los acessíveis a todos os usuários dos níveis decisórios (NAVARRO, 1996). Entretanto, essa integração deve ser feita com uma seleção cuidadosa e otimizada dos dados já que a prioridade na utilização do ambiente do DW é o processamento de consultas e não o processamento de transações. A Figura 1 ilustra toda a estrutura interna que o ambiente de DW representa.

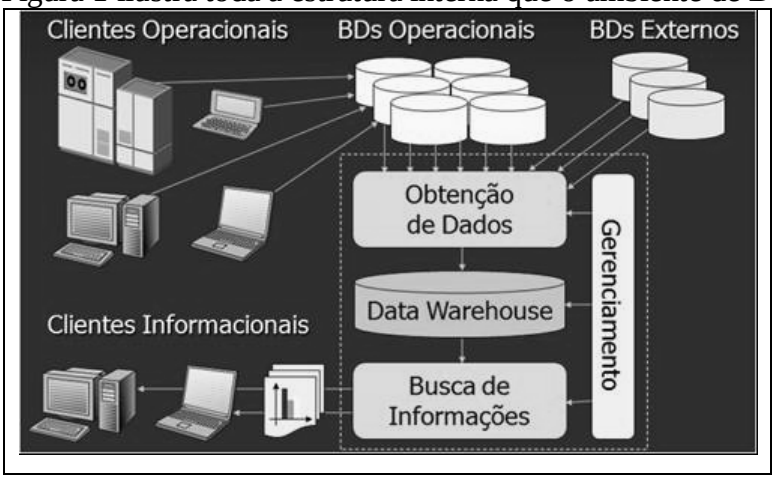


Figura 1 – Estrutura interna do DW

Um DW exige a criação de metadados que define as principais informações de um projeto de DW, além de sua documentação (VIEIRA, 2000). De acordo com Vieira (2000) algumas informações que o metadados deve conter são: a estrutura dos dados segundo a visão do programador e dos analistas de SAD; a origem das fontes de dados que alimentam o DW; a transformação dos dados ocorrida no processo de migração para o DW; o modelo de dados e seu relacionamento com o DW; o histórico das extrações de dados; as informações sobre as consultas e relatórios; acesso e segurança e os indicadores de qualidade de dados.

Uma das técnicas utilizadas para a criação do projeto lógico de um DW é a da modelagem dimensional. Esta técnica é caracterizada pela criação do esquema estrela a partir do esquema conceitual criado na fase de análise do DW. Para Kimball (1997 apud COME, 2001, p. 51) modelagem dimensional é uma técnica utilizada para a definição do projeto lógico de um DW. Três conceitos básicos são importantes nesta modelagem: tabelas fatos ou cubos de decisão que representam as transações de negócios, as dimensões que são os diferentes tipos de visões que os usuários irão utilizar para analisar as métricas e os indicadores ou métricas que podem ser definidos como os

atributos numéricos de um fato representando o comportamento de um negócio para as dimensões.

Outro conceito importante em um projeto de DW é o processo de extração, transformação e carga (ETC) que é o mais trabalhoso na construção de um DW. Durante essa etapa é importante ter uma eficiente integração de dados já que os mesmos podem vir de múltiplas fontes. Sua transformação deve ser feita de forma a gerar informações consistentes e de qualidade. Essa etapa é caracterizada por ser uma das mais críticas já que uma informação carregada erroneamente trará conseqüências imprevisíveis nas fases posteriores (SILVA, 2005, p. 19).

A FERRAMENTA

A ferramenta para a construção de um DW conta com dois atores: o usuário de consultas e o administrador projetista. Neste trabalho, a ênfase principal é com as funções disponibilizadas ao administrador, uma vez que ao usuário caberá apenas a tarefa de efetuar as consultas.

As operações realizadas pelo administrador podem ser divididas nos seguintes processos: montagem do projeto de DW, ETC, consultas, metadados e recursos adicionais da ferramenta.

Para as operações de montagem de projeto destacam-se os seguintes casos de uso:

a) **cadastrar Data Warehouse:** cria um novo projeto de DW baseado no modelo dimensional estrela;
b) **cadastrar dimensão:** grava as definições referentes a uma dimensão bem como seus atributos e chave primária;
c) **cadastrar cubo:** grava as definições referentes a um cubo de decisão bem como seus indicadores e dimensões relacionadas.

Para o processo de ETC destacam-se:

a) **cadastrar conexão:** cria uma nova conexão com um banco de dados que será disponibilizado para extração de dados para as dimensões e cubos do DW;
b) **cadastrar fonte de dados:** cria uma ou múltiplas fonte de dados através das quais irá se fazer a extração, transformação e carga dos dados para as dimensões ou cubos de um DW.

Para as operações de consultas destaca-se:

a) **cadastrar consulta:** define consultas gerenciais baseadas na modelagem dimensional do cubo de decisão;
b) **visualizar e configurar consultas**: acesso e configuração sobre as consultas cadastradas.

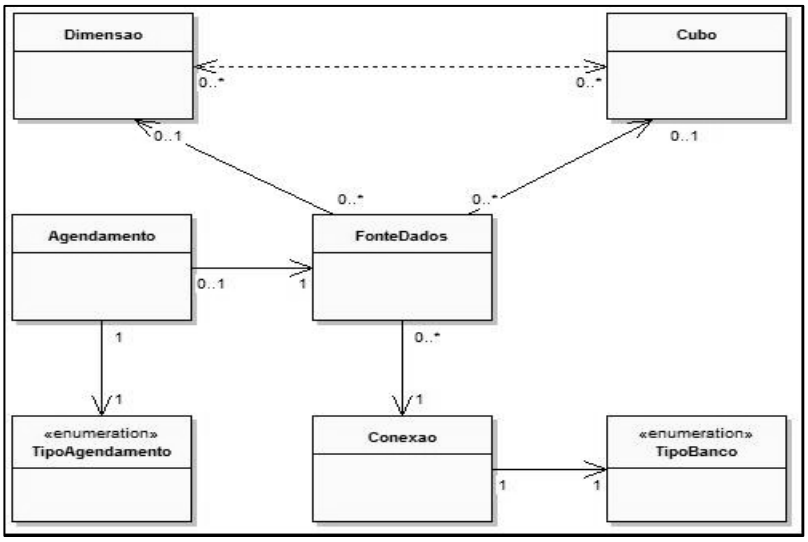
Outros recursos adicionais da ferramenta são:

a) **exportar metadados: e**xporta todas as definições referentes a um projeto de DW em padrão XML,
b) **importar metadados**: importa para o sistema um novo projeto de DW gerado em XML;
c) **visualizar agendamento:** apresenta ao administrador todos os agendamentos de fontes do dia corrente que ainda estão em aberto para processamento;
d) **visualizar log de mensagens:** mostra as principais ocorrências dentro do sistema como informações de importação, erro e tratamento de exceções;

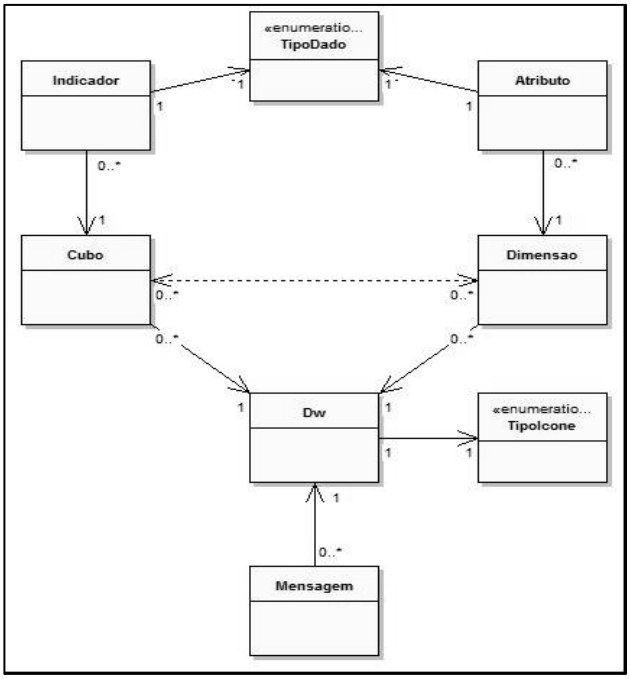
e) **limpar Data Warehouse:** processa limpeza de dados e do conteúdo dos projetos de DW do sistema;
f) **cadastrar usuário:** cria novos usuários para acesso ao sistema.

O diagrama de classes da ferramenta apresentado na Figura 2, foi dividido em três partes: (a) ETC que apresenta o modelo necessário para o processo de extração, transformação e carga dos dados dentro do DW; (b) PROJETO que apresenta o modelo necessário para o processo de construção de um projeto de DW; (c) CONSULTA que apresenta o modelo necessário para a visualização e configuração das consultas do DW. Em cada processo existem classes comuns que são utilizadas e que possuem funcionalidades específicas.

ETC



CONSULTA



PROJETO

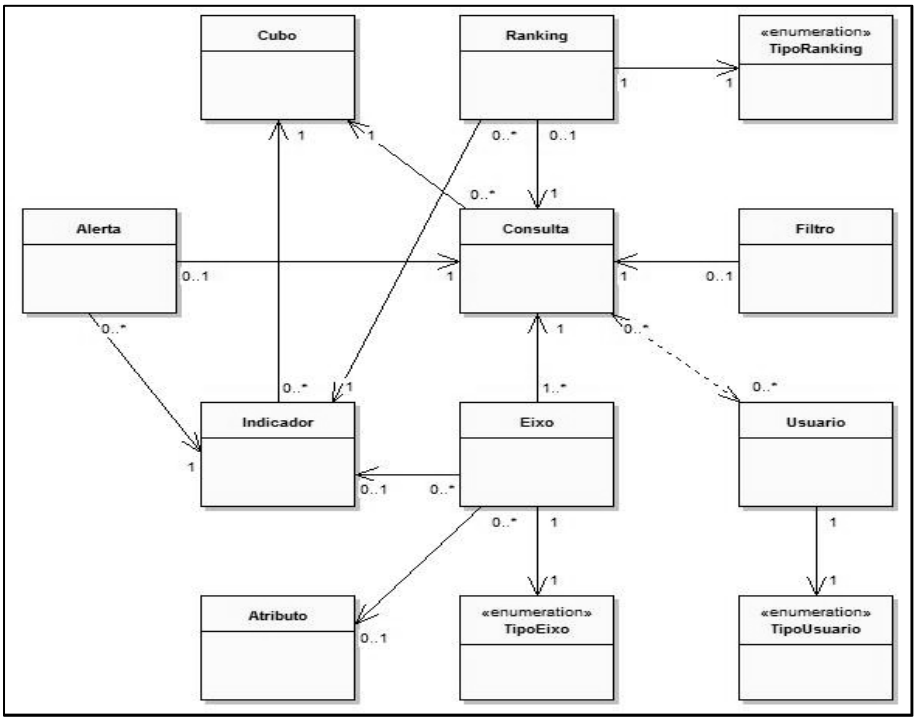


Figura 2 - Diagrama de classes dos pacotes principais do sistema

O sistema ainda apresenta outros pacotes menores que apresentam diagramas mais simples e que possuem finalidades específicas (OLIVEIRA, 2007).

A ferramenta foi desenvolvida na plataforma Java. O sistema foi compilado utilizando o J2SE 1.5 e roda em um servidor que implementa a especificação J2EE 1.4 ou superior. Para o desenvolvimento de aplicações web foi utilizado o *Integrated Development Environment* (IDE) Eclipse 3.2.1 acrescido do *plugin* MyEclipse 5.1.1 que utiliza *servlets* com interfaces JSP. O servidor de aplicações utilizado foi o Apache Tomcat 5.5.23. Para a implantação do AJAX foram utilizadas implementações javascripts com *grids*, para os quais utilizou-se a biblioteca de *scripts* da Zapatec, que possui diversas modelagens para tabela de dados. O banco de dados utilizado foi o MySQL 5.0 com interfaces de conexão JDBC. As tabelas do modelo objeto relacional estão descrita no trabalho de Oliveira (2007).

A seguir será apresentada a operacionalidade do sistema assim como suas principais interfaces e operações. Para ilustração do funcionamento de todo o processo de construção de um DW tomou-se como estudo de caso o faturamento de uma empresa fictícia com as seguintes definições de projeto:

a) manter o histórico de vendas da empresa;
b) consultar a soma das vendas em valor total da nota fiscal por data (ano e mês) e clientes (estado);
c) consultar a média de vendas em valor total da nota fiscal por representante (nome) e clientes (nome), alertando o usuário onde a média das vendas foram abaixo de duzentos reais na cor vermelha e acima na cor verde;
d) consultar os vinte produtos mais vendidos em quantidade no ano de 2007.

Para apresentar este estudo de caso foi criado dentro da ferramenta um projeto de DW utilizando o usuário padrão DWADMIN. A Figura 3 ilustra a tela de login de usuário e a tela onde o administrador pode cadastrar um novo projeto de DW ou já selecionar projetos existentes.

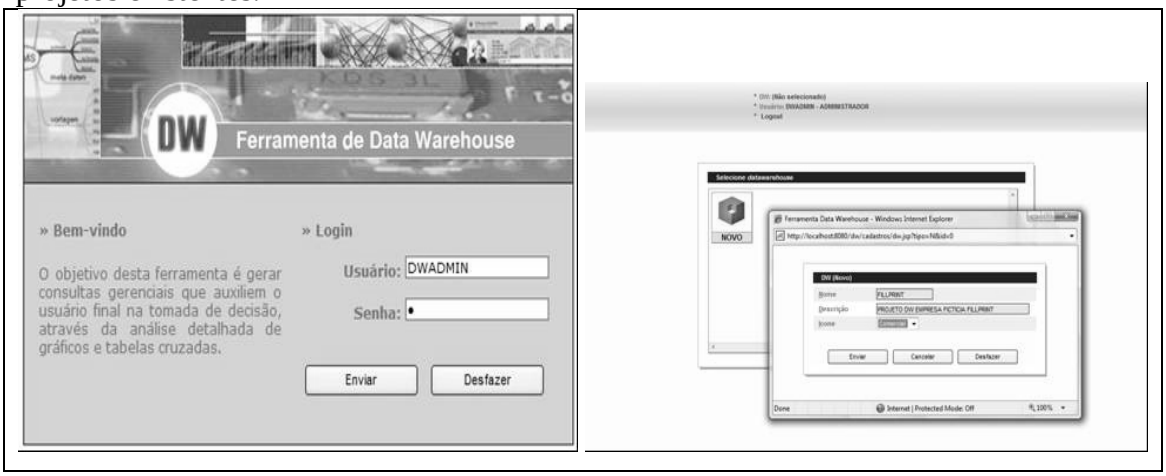


Figura 3 – Tela de login

Com o projeto criado, o administrador pode dar seqüência à montagem do DW. Assim, inicialmente é necessário cadastrar uma conexão com o banco de origem dos dados de vendas da empresa. Tendo uma conexão estabelecida com uma base de dados, o projeto de DW pode começar a ser definido através do cadastro das dimensões e atributos, cubos e indicadores e suas fontes de dados respectivamente. Para o estudo de caso foram feitas as seguintes definições: (a) dimensão cliente: atributos nome e estado; (b) dimensão data: atributos ano e mês; (c) dimensão representante: atributo nome; (d) dimensão produto: atributos nome e tipo; (e) cubo venda: indicadores valor total e quantidade.

A Figura 4 ilustra o cadastro de uma dimensão e de um atributo. Após todos os atributos estarem definidos para a dimensão, é realizada a definição da chave primária da dimensão ilustrada pela Figura 5.

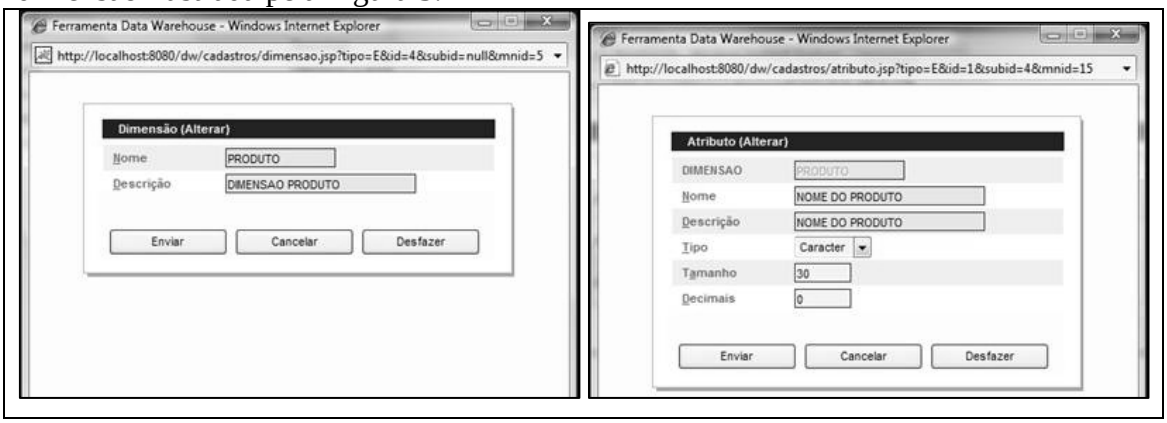


Figura 4 – Tela de cadastro de dimensão

Figura 5 – Tela de definição de chave primária da dimensão

Após o administrador cadastrar todas as dimensões a serem utilizadas no cubo de decisão, foi feito o cadastro do mesmo e definidos seus indicadores. Para o cubo também é necessário a definição das dimensões utilizadas.

O cadastro do cubo de decisão determina que o modelo projetado para o DW esteja pronto para ser carregado com os dados. Desta forma, o processo de ETC é realizado através das fontes de dados que cada dimensão e cubo possuem, pelo roteiro desta fonte de dados que realiza o mapeamento da origem do dado com o projeto definido no DW e, por último, pela importação dos dados que podem ainda ser agendados e processados periodicamente. A Figura 6 ilustra o cadastro de uma fonte de dados, neste caso para o cubo de decisão VENDA e o roteiro desta mesma fonte de dados. Neste caso, por ser um cubo, é necessário mapear tanto os indicadores como as chaves primárias de cada dimensão relacionada. Para as dimensões apenas os atributos necessitam ser roteirizados.

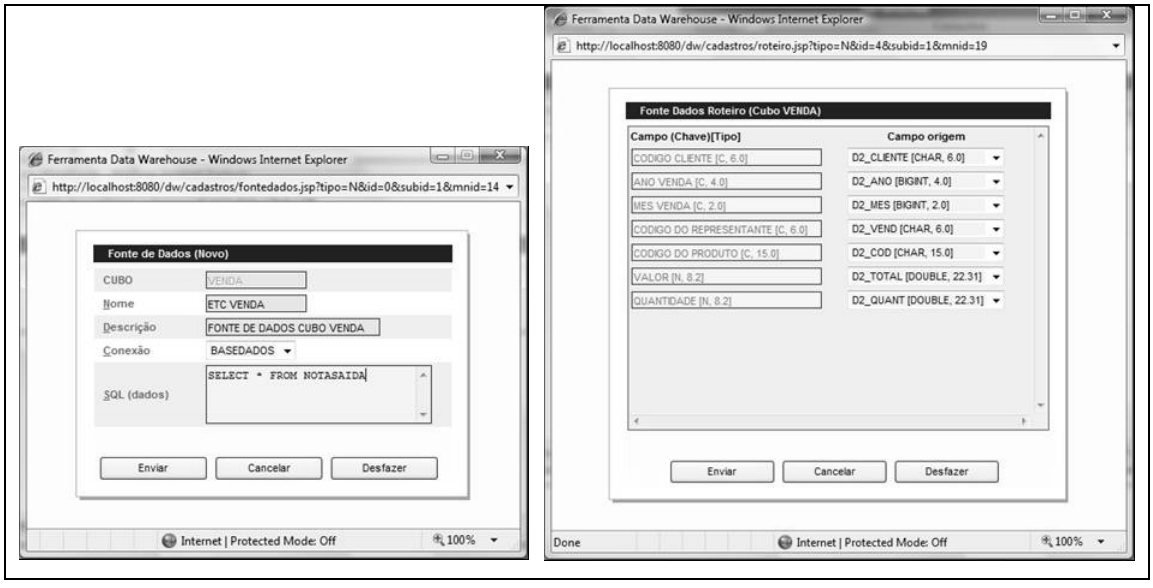


Figura 6 – Tela de cadastro de fonte de dados e tela de roteiro de uma fonte de dados

O próximo passo é a montagem das consultas. O administrador deve cadastrar a consulta e definir os seus eixos. A Figura 7 ilustra o cadastro de uma consulta e a

definição de um novo eixo para esta consulta. O eixo de indicadores possui a definição das funções de agregação representada na Figura 8, que também apresenta a consulta de vendas por estado, onde o administrador utiliza o recurso de drill-down.

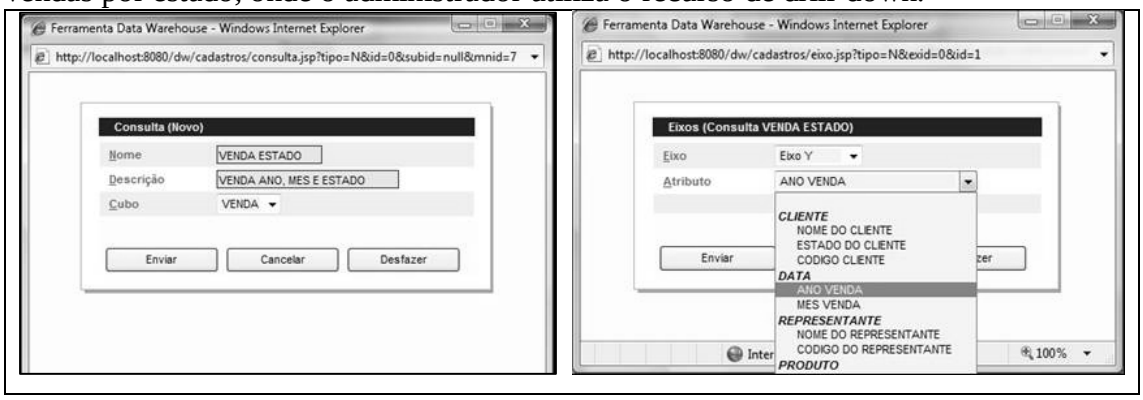


Figura 7 – Tela de cadastro de consultas e Tela de cadastro de eixos da consulta

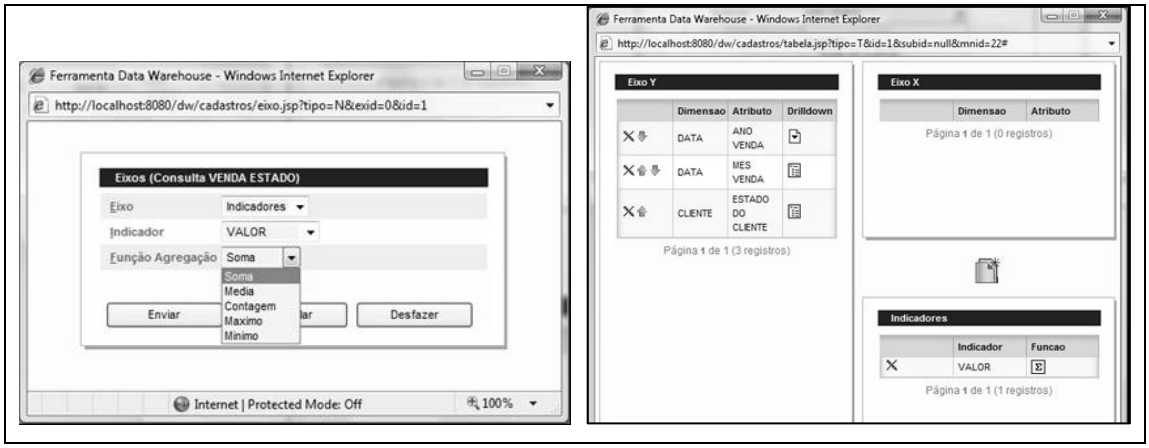


Figura 8 – Tela de cadastro de eixos de indicadores da consulta e Tela de definição da consulta com drill-down

Para visualizar o resultado das consultas é utilizado um perfil de usuário onde apenas a visualização das consultas, a edição de filtro, alerta e ranking estão disponíveis.

As Figura 9, Figura 10 e Figura 11 mostram um exemplo do resultado da consulta de vendas por ano, mês e estado do cliente.

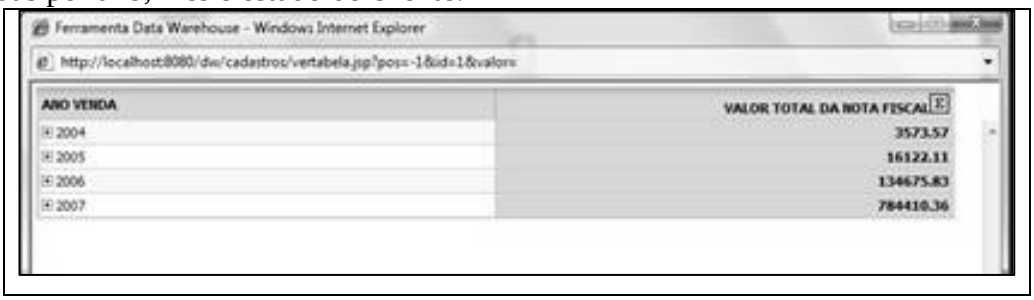


Figura 9 – Tela de consulta de vendas por ano com drill-down

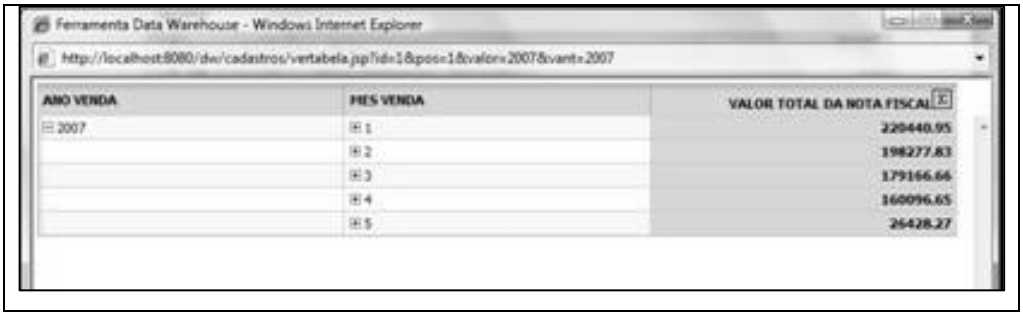


Figura 10 – Tela de consulta de vendas por ano e mês com drill-down

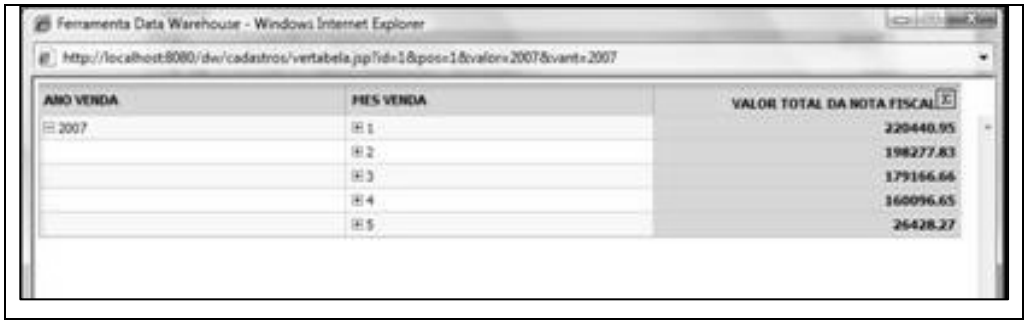


Figura 11 – Tela de consulta de vendas por ano, mês e estado com drill-down

Para montar a consulta de vendas por representantes e clientes é utilizado o recurso de alerta para destacar os resultados obtidos.

A Figura 12 mostra exemplos do resultado da consulta de vendas por representante e por clientes.

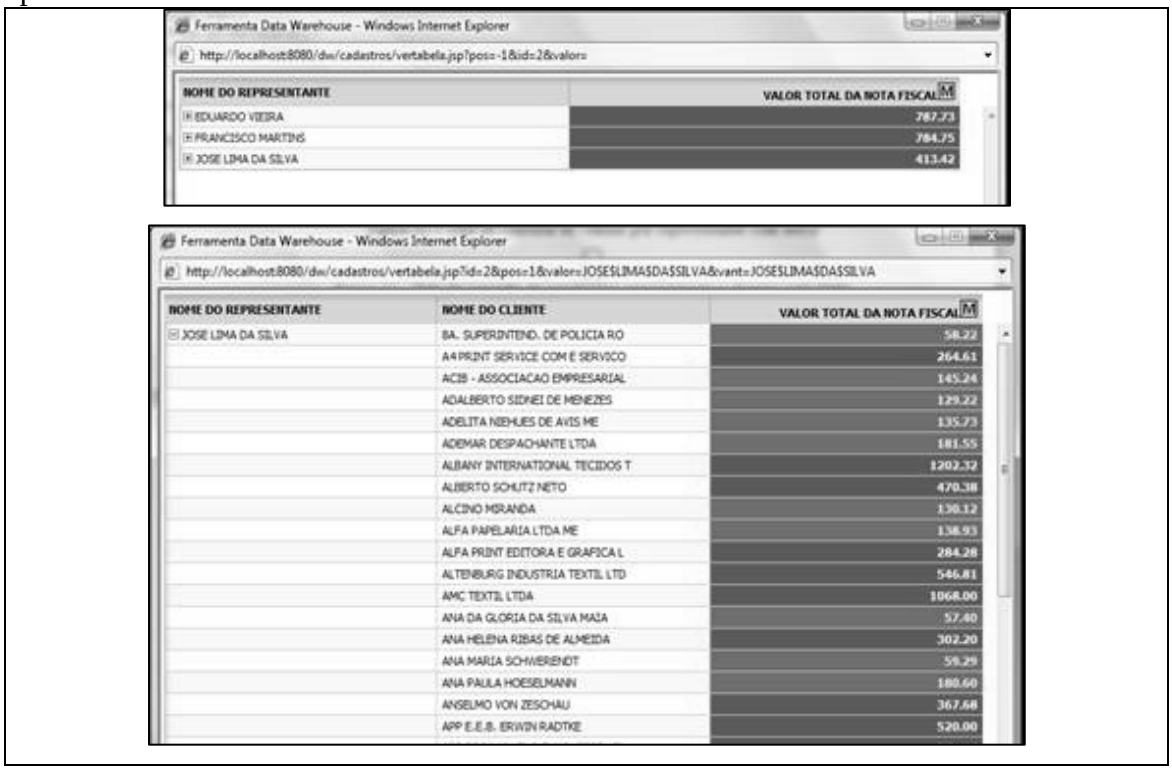
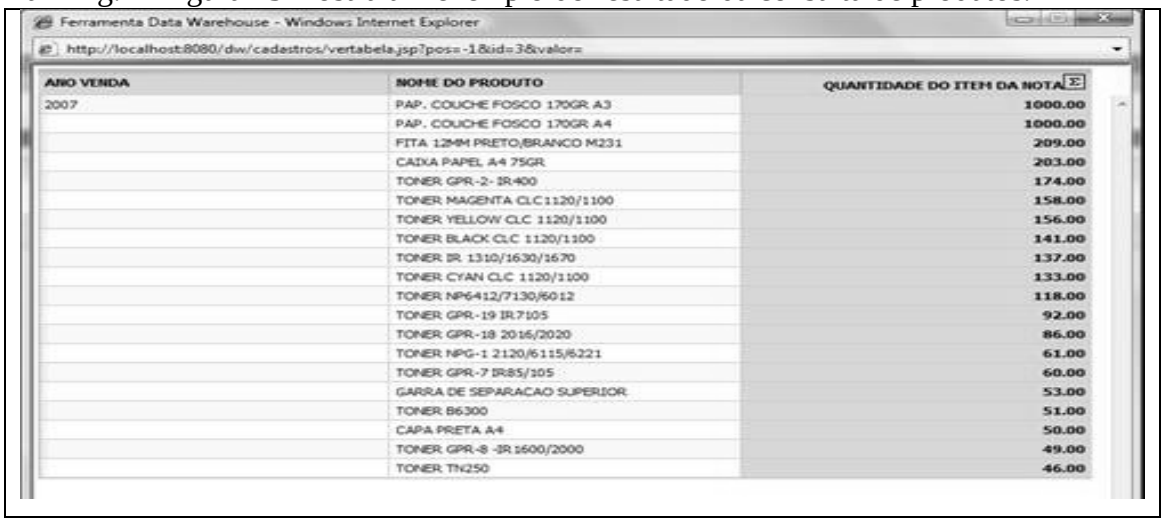


Figura 12 – Tela de consulta de vendas por representante com alerta e Tela de consulta de vendas por representante e clientes com alerta

Para montar a consulta de vendas por produto são utilizados os recursos de filtro e ranking. A Figura 13 mostra um exemplo do resultado da consulta de produtos.



ANO VENDA	NOME DO PRODUTO	QUANTIDADE DO ITEM DA NOTA
2007	PAP. COUCHE FOSCO 170GR A3	1000.00
	PAP. COUCHE FOSCO 170GR A4	1000.00
	FITA 12MM PRETO/BRANCO M231	209.00
	CAIXA PAPEL A4 75GR	203.00
	TONER GPR-2- IR400	174.00
	TONER MAGENTA CLC1120/1100	158.00
	TONER YELLOW CLC 1120/1100	156.00
	TONER BLACK CLC 1120/1100	141.00
	TONER IR 1310/1630/1670	137.00
	TONER CYAN CLC 1120/1100	133.00
	TONER NP6412/7130/6012	118.00
	TONER GPR-19 IR7105	92.00
	TONER GPR-18 2016/2020	86.00
	TONER NPG-1 2120/6115/6221	61.00
	TONER GPR-7 IR85/105	60.00
	GARRA DE SEPARACAO SUPERIOR	53.00
	TONER B6300	51.00
	CAPA PRETA A4	50.00
	TONER GPR-8 -IR1600/2000	49.00
	TONER TN250	46.00

Figura 13 – Tela de consulta de vendas por produto com filtro e ranking

RESULTADOS E DISCUSSÃO

A ferramenta de DW apresentada neste artigo foi desenvolvida com o propósito final de garantir desempenho e usabilidade funcional nas consultas gerenciais que se utilizam de grande volume de dados gravados em bases históricas e banco de dados transacionais.
Para avaliar o resultado obtido foi utilizado o estudo de caso citado neste trabalho sendo feitas comparações entre as consultas processadas diretamente no banco transacional e um banco de dados gerado pela ferramenta de DW. Os pontos avaliados são: tempo de processamento de um consulta, total de tabelas consultadas e quantidade de registros processados. A Tabela 1 mostra os resultados obtidos para a consulta de vendas por ano, mês e estado do cliente em cada nível de detalhamento dos dados. A Tabela 2 mostra os resultados obtidos para a consulta de vendas por representante e clientes e a

Tabela 3 mostra os resultados obtidos para a consulta de venda por ano e produto.

Tabela 1 – Resultados da consulta venda por data e estado

Venda ano, mês e estado						
Nível	Tempo processamento (s)		Total de tabelas		Total de registros	
	BD	DW	BD	DW	BD	DW
1	0,0185	0,0064	1	2	3518	2323 x 1
2	0,0192	0,0066	1	2	3518	2323 x 1
3	1,2249	0,0164	2	3	982 x 3518	27 x 145 x 1

Tabela 2 – Resultados da consulta venda por representante e cliente

Venda representante e cliente						
Nível	Tempo processamento (s)		Total de tabelas		Total de registros	
	BD	DW	BD	DW	BD	DW
1	0,0414	0,0119	2	2	3 x 3518	4 x 232
2	4,3536	0,0145	3	3	3 x 3518 x 982	4 x 232 x 1

Tabela 3 – Resultados da consulta venda por produto

Venda produto						
Nível	Tempo processamento (s)		Total de tabelas		Total de registros	
	BD	DW	BD	DW	BD	DW
1	18,1847	0,0195	2	3	3518 x 15969	27 x 145 x 1

O processo de ETC das principais fontes de dados envolvidas neste projeto é o responsável por este desempenho favorável ao DW, tornando as consultas mais eficientes e rápidas.

A Tabela 4 mostra o tempo exigido pelo processo de ETC. A otimização deste processo através de consultas SQL, bancos de dados indexados e relacionamentos corretos é que garantem um processo de ETC mais eficiente.

Tabela 4 – Resultados do processo de ETC

	Tempo processamento (min.)	Total de registros
Dimensão Cliente	1m 51s	970
Dimensão Produto	26min 12s	11594
Dimensão Data	5min 23s	3496
Dimensão Representante	< 1s	3
Cubo Venda	6min 45s	4359

CONSIDERAÇÕES FINAIS

Com o propósito principal de obter informações gerenciais detalhadas e resumidas provenientes de banco de dados históricos e transacionais, a ferramenta apresentada atinge os objetivos a que se propõe demonstrando ser bastante eficiente, uma vez que aplica os conceitos e técnicas de um sistema de apoio à decisão, neste caso o DW, auxiliando no processo de extração de dados, transformando-os em informações e apresentando-os de forma a obter indicações da evolução e histórico dos dados.

Uma das principais vantagens de migrar os dados transacionais para um banco de dados DW é a organização dos dados garantindo a integridade e qualidade com que os dados são gravados. É no processo de ETC que as informações passam a ser distribuídas e modeladas seguindo as definições do projeto criado na ferramenta. Para obter qualidade dos dados é implementado o conceito de chave primária dentro das dimensões e chave estrangeira dos cubos em relação às dimensões. As chaves primárias definidas garantem a unicidade dos registros, não permitindo ocorrências duplicadas, servindo ainda como referência na montagem das chaves primárias dos cubos de decisão. A ferramenta utiliza-se destas referências para executar a limpeza dos dados garantindo que registros que não possuem integridade referencial válidas sejam descartados.

A ferramenta apresenta técnicas de modelagem de dados que mostram o quanto é importante organizar, referenciar e garantir a integridade dos dados, transformando-os em informações de grande valor para as organizações. Através dos cubos de decisão a ferramenta orienta as informações por assunto, permitindo montar consultas para cada característica em comum que os dados possam apresentar. Ainda, de forma integrada, preocupa-se em trazer dados que possuam informações idênticas, porém de diferentes fontes unificando o estado do dado.

Desenvolvida em ambiente web a ferramenta é acessível através de qualquer navegador de internet, tendo assim um grande diferencial de usabilidade.

Ainda por estar em sua primeira versão, a ferramenta pode evoluir em diversas extensões, tais como, novas funções de agregação e aplicação de algoritmos de mineração de dados, entre outros, permitindo ganhos de qualidade e aproveitamento no gerenciamento estratégico, tático e operacional de uma organização.

REFERÊNCIAS BIBLIOGRÁFICAS

BARBIERI, Carlos. **BI – Business Intelligence – Modelagem & Tecnologia.** Rio de Janeiro: Editora Axel Books, 2001.

COME, Gilberto. **Contribuição ao estudo da implementação de data warehousing**: um caso no setor de telecomunicações. 2001. 132 f. Dissertação (Mestrado em Administração) – Curso de Pós-graduação em Administração, Universidade de São Paulo, São Paulo.

DW BRASIL. **Características de um data warehouse**. Brasília, 2005. Disponível em: <http://www.dwbrasil.com.br/html/artdw_carac.html>. Acesso em: 11 set. 2006.

INMON, William H. **Como construir o data warehouse**. Tradução Ana Maria Netto Guz. Rio de Janeiro: Campus, 1997.

NAVARRO, Maria C. **O que é Data Warehouse?** Brasília, 1996. Disponível em: <http://www.serpro.gov.br/publicacao/tematec/1996/ttec27>. Acesso em: 13 mai. 2007.

OLIVEIRA, Pércio A. Ferramenta de construção de Data Warehouse**.** 2007. 89 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

SILVA, Diogo. **SITC:** uma ferramenta de transformação e carga para um data warehouse**.** 2005. 31 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto de Matemática, Universidade da Bahia, Salvador.

VIEIRA, Fernando. **Alguns conceitos sobre DW**. São Paulo, 2000. Disponível em: <http://www.datawarehouse.inf.br/>. Acesso em: 19 set. 2006.

Índice de Autores