Ferramenta de *Software* para o Auxílio ao Processo de Ensino-Aprendizagem de Métodos Estocásticos

Marlon Domenech[*]
Herculano De Biasi[**]

Resumo

Este artigo descreve a arquitetura geral de um sistema e as características principais de um protótipo de ferramenta de software destinada a auxiliar estudantes durante o processo de ensino-aprendizagem de determinados métodos estocásticos como "Cadeias de Markov" e "Teoria das Filas". A ferramenta aqui descrita utiliza, entre outras técnicas, algoritmos de grafos e álgebra linear para resolver problemas apresentados a ele. Estes são resolvidos automaticamente, passo-a-passo e cada passo é mostrado de forma gráfica ao usuário, em linguagem natural, o que permite aos estudantes compreender e acompanhar todo o processo de forma mais fácil, incentivando o aluno ao autodidatismo, e prática dos conteúdos propostos com exercícios e avaliações.
Palavras-chave: Processos Estocásticos. Engenharia de Software. Padrões de Projeto. Ensino. Desenvolvimento Dirigido por Testes.

1 **INTRODUÇÃO**

O presente artigo apresenta resultados de trabalhos finalizados em fevereiro de 2011, os quais tiveram como objetivo o desenvolvimento de uma ferramenta de *software* que auxiliasse no processo de ensino-aprendizagem de métodos estocásticos, mostrando e complementando, de forma visual, prática e passo-a-passo (estudo dirigido) os conceitos discutidos em sala de aula. Essa complementação é relevante, pois por muitas vezes os conceitos que não são satisfatoriamente entendidos, podem ser retomados ou reforçados em um laboratório com o auxílio de uma ferramenta especializada, que além de exemplificar conceitos da teoria, pode proporcionar ao aluno que ele pratique o que foi trabalhado em sala.

Nos últimos anos, a utilização de ferramentas computacionais tem-se mostrado um importante meio de auxílio ao processo de ensino-aprendizagem de diversas áreas do conhecimento. No exterior, Universidades das mais conceituadas, como MIT, Stanford, Berkeley (Attwood, 2009) oferecem vídeo-aulas para que o aluno possa revisar os conteúdos. Iniciativas como a da entidade sem fins lucrativos chamada Khan Academy, endossadas pela Google e por Bill Gates (Young, 2010) indicam um aumento de aproveitamento por parte do aluno ao utilizar ferramentas similares à proposta neste artigo. A disciplina de métodos estocásticos, também chamada de métodos probabilísticos, ou simplesmente probabilidade, trabalha com conteúdos normalmente classificados como pertencentes à área da pesquisa operacional, mas que possuem como característica fundamental

[*] Pesquisador; graduando em Ciência da Computação pela Unoesc *Campus* Videira; marloncdomenech@gmail.com
[**] Orientador; Mestre em Ciência da Computação pela Universidade Federal de Santa Catarina; herculano.debiasi@gmail.com

uma componente não-determinística, ou aleatória. Infelizmente é fato reconhecido e notório a dificuldade do ser humano em trabalhar com conceitos que envolvam probabilidade, pois esses são, frequentemente, contraintuitivos. Problemas simples de probabilidade, como o chamado "Problema de Monty Hall" (Mlodinov, 2010), ludibriaram, inclusive, matemáticos de renome mundial, como o húngaro Paul Erdős.

2 **METODOLOGIA**

Os trabalhos, cujos resultados estão apresentados neste artigo, iniciaram-se com a revisão bibliográfica referente aos processos estocásticos, a fim de identificar os aspectos (principalmente os que tangem à natureza operacional das técnicas abordadas e possíveis forma de exibição/tratamento/resolução de problemas) mais importantes a serem incluídos no *software*. Foi então realizada uma pesquisa de campo com professores da Unoesc de áreas relacionadas, a fim de validar os requisitos encontrados para o *software*. Essa etapa veio para verificar se os requisitos encontrados para o *software* eram válidos conforme experiência de profissionais que lecionam disciplinas relacionadas. Assim, validou-se que o que seria feito na etapa de projeto do *software* e codificação estaria de acordo com a necessidade dos usuários. O próximo passo foi a de análise de requisitos, o que incluiu a modelagem de cenários e de diagramas de *use-cases*. Em seguida, passou-se ao projeto do sistema, que indicou, entre outros, a melhor metodologia e arquitetura para o *software*, o banco de dados, a plataforma-alvo e a linguagem de programação.

A documentação foi produzida em forma de diagramas UML, principalmente os de classes e sequencia, sendo também utilizado o diagrama de E-R, normalmente usado para descrever o modelo de dados de um sistema em um nível abstrato mais alto. Os componentes principais do sistema também foram diagramados e roteiros explicando o funcionamento completo do *software* foram gerados para facilitar o entendimento de seu funcionamento. O *software* foi implementado de forma iterativa com o projeto e análise. Optou-se também pela metodologia de desenvolvimento dirigido por testes, a fim de garantir principalmente a qualidade das operações matemáticas inclusas no *software*, assim como demais funcionalidades. Foram aplicados diversos padrões de projeto como MVC, DAO (nas classes de comunicação com o Banco de Dados), Singleton (na criação de objetos que podem ser únicos, como a conexão com o Banco de Dados), entre outros, os quais têm por objetivo proporcionar maior flexibilidade no momento de estender a plataforma para *web*, assim como proporcionar algumas facilidades no desenvolvimento.

Observou-se também que, para o desenvolvimento da ferramenta, diversas técnicas de várias áreas do conhecimento foram aplicadas, entre elas álgebra, probabilidade, estatística, processos estocásticos, pesquisa operacional, engenharia de *software*, teoria dos grafos, abrindo um leque para a realização de diversos estudos nas áreas afins por parte dos pesquisadores e enfatizando o caráter multidisciplinar do projeto de pesquisa. Finalmente, vídeo-aulas foram desenvolvidas para complementar a funcionalidade do *software*. O estágio atual do *software* já permite aplicação prática em sala de aula e um estudo de validação já está programado para o segundo semestre de 2011.

2.1 CENÁRIOS DE UTILIZAÇÃO E CASOS DE USO

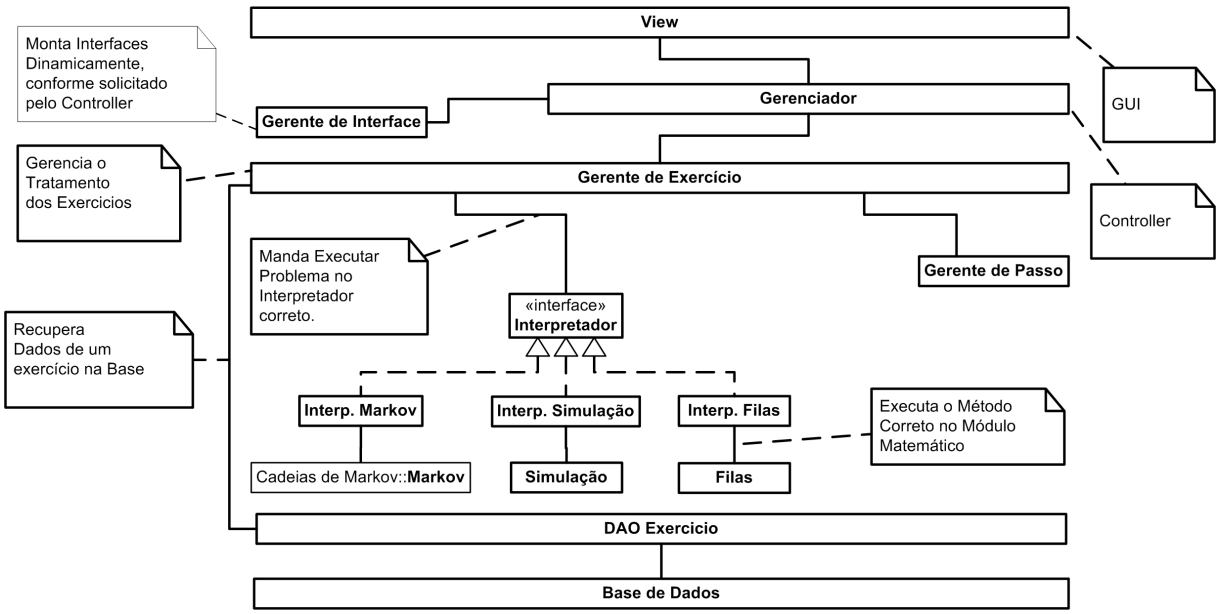
Quando o sistema foi inicialmente idealizado, pensou-se em dois cenários principais, chamados de "estático" e "dinâmico". Atualmente só o primeiro está implementado.

No cenário estático, o professor é o único que pode inserir o exercício ou problema na base de dados do sistema. O aluno pode então acessá-lo e visualizar, passo-a-passo, sua resolução. No cenário "dinâmico", o aluno será capaz de alterar os parâmetros do exercício, a fim de realizar testes e verificar como o sistema se comporta frente às alterações, o que será fundamental para estudar sistemas de filas e simulações. Além disso, no cenário "dinâmico", o aluno poderá resolver o problema proposto diretamente no *software*, o qual avaliará a qualidade da resolução do aluno, indicando erros e acertos, no que se refere à resultados finais e parciais de resolução. Também poderá ser feita uma comparação entre a resolução feita pelo aluno e a resolução feita pelo *software*, indicando em que parte do processo o aluno cometeu algum erro.

2.2 ARQUITETURA GERAL

O Esquema 1 a seguir mostra a arquitetura geral do sistema. Nela é possível ver implementado o MVC, onde a *View* está explícita no módulo mais acima, o *Controller* é representado pelo Gerenciador, enquanto que os demais componentes são parte do *Model*. Na base do diagrama é possível ver o Banco de Dados, e a ele ligado o componente DAO. Abaixo da *View*, ao lado do Gerenciador, é possível ver o componente Gerente de Interface, que é responsável por montar a *View* dinamicamente (tanto *web* como *Desktop*), conforme os passos de exercícios são executados. Abaixo do Gerente de Interface está o Gerente de Exercício, que é quem controla o fluxo de informações pelos demais componentes. Ligado a ele, mais à direita, está o Gerente de Passo, que faz a divisão dos passos de execução. Abaixo está a Interface Interpretador, a qual define por qual componente matemático e por qual algoritmo um problema deverá ser resolvido, além de definir a lista de componentes da *View* para cada passo de execução, por meio de linguagem específica para tal propósito. Por fim, abaixo dos interpretadores estão os módulos matemáticos, que fazem a resolução dos problemas e constroem o esquema de passos.

Esquema 1– Arquitetura geral do sistema



Fonte: os autores.

Uma característica do *software* é a utilização de vários módulos chamados “interpretadores”. É neles em que estão implementadas as bibliotecas matemáticas para a resolução dos problemas. Os interpretadores foram desenvolvidos segundo o padrão de projeto *Strategy* (Gamma, 1995).

O sistema foi assim planejado para permitir uma fácil expansão futura, visto que módulos adicionais, como de programação linear, já estão previstos. Esse padrão também foi pensado para que o sistema possa, mais adiante, escolher algoritmos diferentes para executar a mesma função, de acordo com a tarefa apresentada. Isso possibilita que o sistema possa levar em consideração vários *trade-offs* na hora da escolha do algoritmo, como velocidade do algoritmo, consumo de memória, e, o que é mais importante, escolher entre um algoritmo mais simples ou mais avançado para ser seguido pelo aluno.

Por exemplo, ao resolver sistemas de equações lineares ele poderia escolher entre resolução por determinantes, por Gauss, Gauss-Jordan, fatorização LU e assim por diante. Todos eles possuem vantagens e desvantagens dos pontos de vista de velocidade, complexidade, ou simplesmente por fatores didáticos em relação ao estágio atual de aprendizagem do aluno.

2.3 LEVANTAMENTO DE REQUISITOS

Foi elaborado um questionário para fins de pesquisa de campo, ou seja, levantamento de dados iniciais com futuros usuários da plataforma, a fim de elucidar as reais necessidades destes e suas opiniões sobre os conteúdos relevantes. O questionário foi enviado para 66 profissionais de ensino de todos os Campi da Unoesc, por meio de dados fornecidos pela Unoesc Videira com deferimento da vice-reitoria acadêmica. Destes, apenas 5 se propuseram a responder e o resultado detalhado desta pesquisa de campo, assim como o questionário utilizado, encontram-se no endereço <https://docs.google.com/viewer?a=v&pid=explorer&chrome=true&srcid=0B3PFWyOE12UaM2IxZTU2NzYtNTc3MS00YTY4LTgxMWYtYWNlMGY5ZDU3ZGEx&hl=en_US>. Essa pesquisa de campo mostrou que a proposta da plataforma

é realmente válida e vai de encontro com os anseios dos futuros usuários. As perguntas do questionário de pesquisa e a tabulação dos resultados podem ser visualizados no endereço . Foi possível, mediante análise dos dados, verificar quais são as características que os usuários acreditam ser importantes na plataforma e que, consequentemente, foram adotadas como casos de uso.

2.4 UTILIZAÇÃO DE GRAFOS

Uma das características desse projeto é a utilização de técnicas de várias disciplinas. Algoritmos relacionados à Teoria dos Grafos foram utilizados para resolver os seguintes problemas:

- Para descobrir se a cadeia é ergódica, ao invés de multiplicar a matriz de transição por ela mesma diversas vezes e correr o risco de entrar em um laço infinito (como é o caso de cadeias em que mesmo após diversas transições não há a probabilidade de uma transição de um estado para outro em um único passo), um algoritmo testa se há um caminho de um estado para todos os outros estados). Isto é feito com todos os estados. Se o teste obtiver sucesso então é ergódica.
- Para saber se a cadeia é periódica, primeiro é verificado se não há nenhum estado absorvente (senão não é uma cadeia periódica) e depois testa-se se há um ciclo no grafo, haja visto que cadeias periódicas denotam um ciclo entre estados.
- Para saber se a cadeia é absorvente, primeiro é verificado se existe algum estado absorvente. Existindo, o algoritmo testa por meio de grafos se há um caminho no grafo de algum estado para o estado absorvente. Se existir é uma cadeia absorvente.

2.5 DESENVOLVIMENTO BASEADO EM TESTES

Como o *software* necessita resolver equações matemáticas dinamicamente para resolver os exercícios propostos, foi dada uma ênfase muito grande para garantir que o módulo matemático estivesse livre de *bugs*. Para isso optou-se pelo desenvolvimento baseado em testes. Foi utilizada a biblioteca do Java JUnit 4.5. e escritos 83 métodos de teste, que testaram principalmente os métodos matemáticos e de grafos. Os métodos de teste incluem várias das chamadas “condições de borda ou limite”, como por exemplo, casos em que o sistema de equações resultante de uma cadeia de Markov acaba tendo valor indeterminado para suas variáveis. Nesses casos, é verificado se o algoritmo toma o caminho certo e resolve o problema corretamente.

3 **REPRESENTAÇÃO DOS DADOS**

Atualmente, os dados necessários estão sendo armazenados em um banco de dados relacional, mas já existe um estudo para a mudança para um banco de dados objeto-relacional como o PostgreSQL. Hoje a descrição dos problemas e dos parâmetros dos exercícios é armazenado em uma longa *string*, fazendo-se necessário realizar um *parsing* nela para recuperar as informações re-

levantes. A gravação e recuperação dos objetos diretamente na base simplificaria a programação do sistema, não havendo mais a necessidade do *parser*.

As informações necessárias para representar os problemas e seus parâmetros, assim como as respostas, são codificadas em uma linguagem própria. Abaixo podem ser vistos como alguns dados são codificados nela:

P1 = passo 1
NI = número de iterações
MT = matriz de transição
MEA = matriz de estados atual
E = se é ergódica

Cada módulo interpretador gera suas saídas utilizando essa mesma forma de representação. O módulo responsável pela geração da interface gráfica (Gerente de Interface), realiza um *parsing* dessas informações para construir a representação visual correspondente.

Exemplo: P1|NI{5}|MT{{1;2}{3;4}}|MEA{{1;2}{3;4}}|E{1}

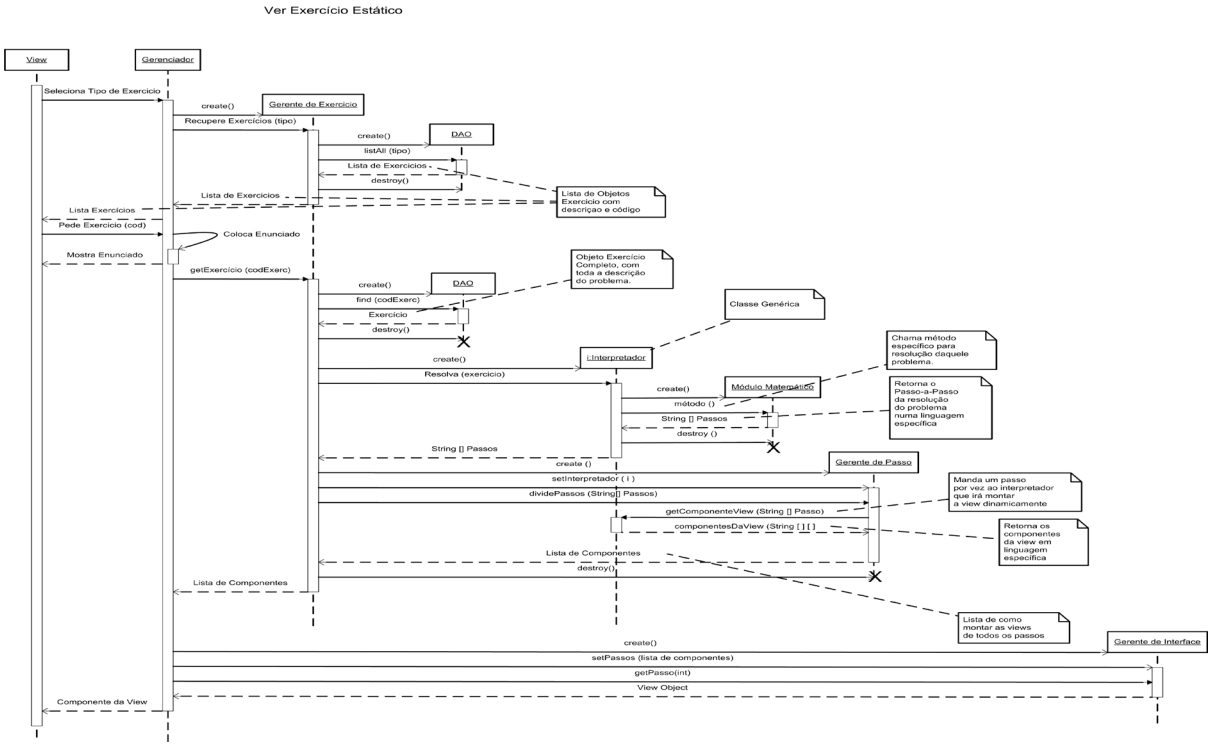
Atualmente o sistema conta com 35 tabelas. Tanto a descrição formal dos parâmetros fornecidos pelo problema (campo *desc_formal_inf*) quanto a descrição formal do que deve ser resolvido (campo *desc_formal_quest*) pelo *software*, estão presentes na tabela 'Questão'. Ambas utilizam a mesma linguagem, descrita anteriormente, para a codificação das informações.

4 **EXEMPLO DE EXECUÇÃO DE EXERCÍCIO ESTÁTICO**

O usuário seleciona na *view* o tipo de exercício que deseja acessar (Cadeias de Markov, Teoria das Filas ou Simulação). O *controller* trata esse evento, fazendo uma requisição ao gerente de exercício. O gerente de exercício pede para uma DAO recuperar todos os exercícios daquele tipo da base de dados. A DAO retorna para o gerente de exercício, este para o *controller*, que altera a *view*, postando ao usuário a lista de exercícios. O usuário pede um exercício específico por meio de um evento na *view*. O *controller* trata esse evento, fazendo uma requisição ao gerente de exercício. O gerente de exercício pede para uma DAO recuperar um exercício da base de dados. A DAO recupera e retorna o exercício para o gerente de exercícios. O gerente de exercício pega o exercício e passa ao interpretador correto, do módulo de exercício escolhido. Esse interpretador chama o método apropriado no módulo matemático, e como resposta obtém um *String*[] com os passos para a resolução.

O interpretador retorna esses passos para o gerente de exercícios que os manda para o gerente de passos. O gerente de passos instancia uma lista para controle dos passos exibidos ao usuário. O gerente de passos irá então enviar cada passo, um por vez, para o interpretador, o qual irá retornar, descrito na linguagem já descrita, um conjunto de objetos que devem ser montados (dinamicamente) na *view*, quando o usuário solicitar aquele passo da resolução do exercício. O gerente de passo retorna ao gerente de exercício a lista dos componentes necessários para a montagem da *view*, de forma dinâmica.

Esquema 2 – Parte do diagrama de seqüência para o exercício estático



Fonte: os autores.

Este último passa a lista ao *controller*. O *controller* manda a lista ao gerente de interface, o qual interpreta a lista de objetos a serem montados dinamicamente na *view* e, assim que solicitado pelo *controller*, envia um objeto da *view* (uma pagina JSP ou um JPanel, por exemplo) ao *controller*, o qual atualiza a *view* e mostra o passo da resolução ao usuário.

5 **GERAÇÃO DOS RESULTADOS**

A seguir é mostrado um exemplo descrição da resolução passo-a-passo de um exercício típico de Cadeias de Markov, realizada de forma automática pelo *software*. Essa descrição é interna ao *software*, sendo que ao usuário é mostrada a interface gráfica presente na seção 6. Os dados de entrada são a matriz de transição e o vetor de estado inicial. A matriz de transição $\begin{bmatrix} 0,4 & 0,6 & 0,0 \\ 0,2 & 0,8 & 0,0 \\ 0,0 & 0,0 & 1,0 \end{bmatrix}$ é representada da forma: {{0.4;0.6;0.0}{0.2;0.8;0.0}{0.0;0.0;1.0}}.

O vetor de estado inicial $\begin{bmatrix} 0,2 & 0,3 & 0,5 \end{bmatrix}$ é representado como: {{0.2;0.3;0.5}}

Queremos saber qual será o resultado do sistema após três iterações. Resolução pelo *software*:

passo 1:
Número de Iterações: 3
vetor estado inicial
{{0.2;0.3;0.5}}
matriz de transição
{{0.4;0.6;0.0}{0.2;0.8;0.0}{0.0;0.0;1.0}}
descrição da solução
Descrição = Realizar 3 iterações na cadeia de markov, tendo como estado inicial do sistema o vetor dado.

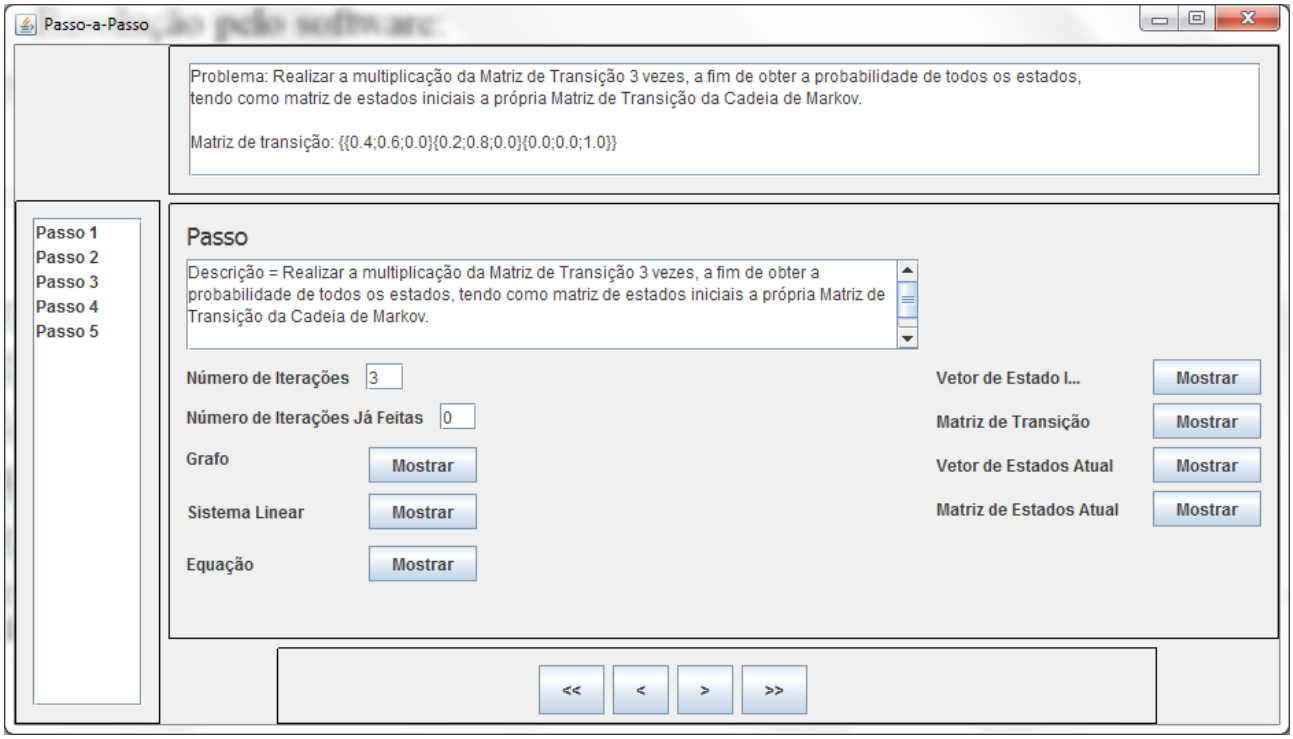
passo 2:
descrição da solução
Multiplicação do vetor de estado inicial pela coluna 1. Multiplica-se o valor 0.2 do vetor de estado inicial pelo valor 0.4 da Matriz de Transição, obtendo 0.08.

passo 3:
descrição da solução
Multiplicação do vetor de estado inicial pela coluna 2. Multiplica-se o valor 0.2 do vetor de estado inicial pelo valor 0.6 da Matriz de Transição, obtendo 0.12.

6 **EXIBIÇÃO DOS DADOS**

Os dados obtidos no processo descrito anteriormente são exibidos ao usuário. A interface gráfica atual é somente um primeiro protótipo, já que a maior parte do esforço de desenvolvimento até agora foi nos algoritmos numéricos. As matrizes e equações produzidas pelo *software* ainda não são renderizadas em forma gráfica e os dados gerados ainda estão sendo exibidos em forma de texto simples.

Esquema 3 – Interface gráfica do *software*



Fonte: os autores.

Um dos próximos incrementos no *software* será a apresentação de forma gráfico, para isso está sendo estudada a adoção da linguagem de descrição de fórmulas utilizado pela Wikipédia, chamada de AMS-LaTeX (superconjunto da linguagem LaTeX). Como na Wikipedia, as fórmulas serão descritas por essa linguagem e então interpretadas por um componente chamado renderizador, o que irá gerar dinamicamente um arquivo gráfico (png ou jpg) correspondente. Alguns renderizadores LaTeX *open source* para Java, como JMathTeX e JEuclid estão sendo investigados.

7 **TRABALHOS FUTUROS**

Para os próximos seis meses está prevista a implementação do módulo para suporte às técnicas relacionadas à "Teoria das Filas", incluindo a simulação gráfica de filas das mais diversas configurações. O usuário poderá alterar os parâmetros do sistema de filas, como número de atendentes, número de canais e tamanho da população e observar qual será o impacto no sistema como um todo. Também neste prazo será finalizada a migração do sistema para a plataforma *web*. A versão *web* do sistema está atualmente sendo projetada e desenvolvida com o *framework* Seam. A *interface* gráfica está sendo modelada em JSF (*Java ServerFaces*), a camada de persistência por meio de *beans* de entidade juntamente com anotações Java e a biblioteca Hibernate, que também torna fácil e transparente o mapeamento objeto-relacional. Como citado anteriormente a representação visual das equações e matrizes também será melhorada por intermédio do uso de um renderizador LaTeX. Com esses recursos implementados já será possível um primeiro estudo de validação da ferramenta, a ser feito durante a disciplina de Métodos Estocásticos da Unoesc, *Campus* de Videira, SC, no segundo semestre de 2011. Até o final do ano está também prevista a ampliação do módulo de Markov para suportar os chamados "Processos de Decisão de Markov", juntamente com ferramentas básicas de análise estatística. Também será implementado o módulo de Simulação Estocástica, suportando primeiramente algumas técnicas conhecidas como "Métodos Monte Carlo".

8 **CONCLUSÃO**

A partir dos resultados alcançados neste trabalho, é possível afirmar que, tratando-se de um projeto que envolve a concepção da arquitetura de um *software* que venha a ser robusto o suficiente para atuar em ambiente *web*, realizar a resolução de exercícios passo-a-passo dinamicamente dando respostas em linguagem natural, suportar módulos de Teoria das Filas, Cadeias de Markov e Métodos Monte Carlo, além da inclusão de vídeo-aulas, os objetivos foram alcançados.

Ainda, a partir desses resultados é possível vislumbrar diversos trabalhos futuros, como citado na sessão sete, os quais são perfeitamente suportados pela arquitetura desenvolvida até agora. Portanto, é possível concluir a partir dos resultados obtidos que é possível chegar à um *software* completo para o auxílio ao processo de ensino-aprendizagem de métodos estocásticos, a partir da arquitetura proposta e do que foi desenvolvido até agora.

Software Tool to Aid the Teaching-Learning Process of Stochastic Methods

Abstract

This article describes the general architecture of a system and the main features of a software tool prototype intended to help students during the teaching-learning process of some stochastic methods such as Markov Chains and Queue Theory. The tool described here uses, among other techniques, graphs and linear algebra algorithms to solve problems presented to it. These problems are solved automatically, step-by-step, and each step is graphically shown to the user, in natural language. This

allow students to understand and follow all the processes is an easier way, encouraging the student to autodidact, and the opportunity to practice the proposed content, by exercises and assessments.
Keywords: Stochastic Process. Software Engineering. Design Patterns. Teaching. Test Driven Development.

REFERÊNCIAS

ALENCAR, M. **Probabilidade e Processos Estocásticos**. São Paulo: Érica, 2009.

ATTWOOD, R. **Get it out in the open. Times Higher Education (London)**. Disponível em: <http://www.timeshighereducation.co.uk/story.asp?storycode=408300>. Acesso em: 18 dez. 2010.

FREITAS, P. **Introdução à Modelagem e Simulação de Sistemas**. 2. ed. Florianópolis: Visual Books, 2008.

GAMMA, E. et al. **Design Patterns**: Elements of Reusable Object-Oriented Software. 1. ed. Estados Unidos da América: Addison-Wesley, 1995.

HILLIER,F; LIEBERMAN,G. **Pesquisa Operacional**. 8. ed.São Paulo: McGrawHill, 2006.

KOLMAN, B. **Introdução à Álgebra Linear com Aplicações**. 6. ed. Rio de Janeiro: Prentice-Hall do Brasil, 1996.

LACHTERMACHER, G. **Pesquisa Operacional na Tomada de Decisões**. 3. ed. Rio de Janeiro: Prentice Hall do Brasil, 2009.

MLODINOW, Leonard. **O Andar do Bêbado**. São Paulo: Editora Zahar, 2009.

SHAMBLIN, James; STEVENS JR. **Pesquisa Operacional**: Uma abordagem básica. 3. ed. Atlas, São Paulo, 1989.

PRADO, Darci. **Teoria das Filas e da Simulação**. 4 ed. Editora INDG, 2009.

YOUNG, Jeffrey R. College 2.0: A Self-Appointed Teacher Runs a One-Man 'Academy' on YouTube. **The Chronicle of Higher Education**. Disponível em: <http://chronicle.com/article/A-Self-Appointed--Teacher-Runs/65793/>. Acesso em: 5 jan. 2011.