UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

DESENVOLVIMENTO DE UM COMPONENTE DE SOFTWARE PARA MANUTENÇAO DE TAXAS

Jonas Pereira da Silva Júnior
Orientador: Prof. Dr. Fábio Nogueira de Lucena

Goiânia (GO), 2003

JONAS PEREIRA DA SILVA JÚNIOR

DESENVOLVIMENTO DE UM COMPONENTE DE SOFTWARE PARA MANUTENÇÃO DE TAXAS

Projeto Final apresentado ao Instituto de Informática da Universidade Federal de Goiás, como requisito para a obtenção do certificado de Especialização em Análise e Projeto de Sistemas de Informação.
Área de concentração: Análise e Projeto Orientados a Objeto.
Orientador: Prof. Dr. Fábio Nogueira de Lucena

Goiânia (GO), 2003

JONAS PEREIRA DA SILVA JÚNIOR

DESENVOLVIMENTO DE UM COMPONENTE DE SOFTWARE PARA MANUTENÇÃO DE TAXAS

Projeto Final de Especialização apresentado e aprovado em _____ de _____ de 2003 pela Banca Examinadora constituída pelos professores.

Prof. Dr. Fábio Nogueira de Lucena

Prof. Dr. Dirson Santos de Campos

Prof. Dr. Carlos Alberto Moreno Barbosa

A meus pais, que durante todos os meus anos como estudante, me apoiaram e incentivaram na minha busca de ampliação do conhecimento.

AGRADECIMENTOS

Ao professor e orientador deste trabalho, Fábio Nogueira de Lucena, pela sua dedicação e seriedade tanto nas aulas como nas orientações.

A todos os professores do curso de Especialização em Análise e Projeto de Sistemas de Informação, pela qualidade das aulas e do material repassado para os alunos.

Aos amigos que me ajudaram nas minhas dúvidas durante todos os módulos do curso.

SUMÁRIO

LISTA DE FIGURAS

GLOSSÁRIO

SISPG

Sistema de Gerenciamento da Pesquisa e da Pós-Graduação da UFG.

XML

Linguagem estendida de marcação.

DOM

Modelo de objetos hierárquicos.

JUNIT

Ferramenta que suporta a criação e execução de testes de unidade.

WWW

World Wide Web

SGML

Linguagem utilizada para definir a estrutura de outras linguagens baseadas em “markup".

DTD

Definição do tipo do documento.

HTML

Linguagem universal destinada à elaboração de páginas com hiper-texto.

C++

Linguagem de programação orientada a objetos

XSL

Linguagem de especificação de estilo para documentos XML.

CSS

Linguagem de especificação de estilo para documentos XML.

UML

Linguagem de modelagem unificada

RESUMO

O SISPG é um sistema que está sendo desenvolvido para facilitar o gerenciamento das atividades da pós-graduação da Universidade Federal de Goiás (UFG). O presente projeto visa o desenvolvimento de um componente de software que realizará parte das atividades do SISPG, em particular, aquelas pertinentes às taxas de serviços prestados pela UFG. Os resultados obtidos incluem a eliciação de requisitos e um protótipo que os implementa.

ABSTRACT

SISPG is an information system under development to help the management of data from graduate courses of UFG. This work deals with the development of the tax component of SISPG. This component is responsible for keeping track of UFG fees. The obtained results include the rising of requirements and a prototype that it implements them.

1. INTRODUÇÃO

Embora seja uma universidade pública, vários serviços oferecidos pela UFG estão associados ao pagamento de taxas. A inexistência de um ambiente integrado onde taxas podem ser consultadas, oferecendo informações atualizadas, está entre os inconvenientes que a criação deste componente deverá eliminar. Esta dificuldade foi detectada através das necessidades do SISPG que não eram atendidas pela inexistência, por exemplo, de software cuja função fosse administrar estas taxas.

Ainda convém ressaltar a volatilidade dos valores cobrados por cada serviço. De um período para outro, serviços podem ter suas taxas eliminadas, da mesma forma que outras podem ser criadas e estar associadas a valores a serem pagos para execução deles. O componente proposto neste trabalho contemplará este cenário e aquele motivado pela existência de sistemas de informação, que deverão efetuar consultas ao componente proposto tanto com o propósito de obter valores atualizados quanto alterá-los.

1.1. Objetivo do Trabalho

O objetivo principal deste trabalho é facilitar a manutenção e consulta às taxas cobradas pela UFG através do desenvolvimento de um componente de software que auxilie a realização destas atividades.

1.2. Justificativa

O desenvolvimento deste projeto é justificado pela facilidade com que as taxas poderão ser geridas e particularmente, consultadas.

Atualmente as taxas encontram-se espalhadas por várias páginas estáticas HTML. Quando ocorre uma alteração de valor, surge nítida a fraqueza desta abordagem. Noutros, onde se esperava o acesso ao valor de taxas, estas não estão presentes pelas dificuldades inerentes ao gerenciamento “manual”. O componente cujo desenvolvimento é o alvo deste projeto deverá eliminar tais dificuldades.

A PROAD é o órgão da UFG responsável pela manutenção destas taxas, o que é realizado através do emprego de um editor de texto, ferramenta inadequada pelo tamanho e dificuldades comentadas acima. Noutras palavras, o componente alvo deste sistema estará não apenas contribuindo com as atividades da PROAD, mas com toda a comunidade interessada nas taxas praticadas pela UFG.

1.3. Metodologia adotada

Este projeto é parte de outro maior cujo processo de software é descrito no documento Development Case [3]. De forma geral, o processo é iterativo, incremental e seus artefatos são registrados através do emprego da UML. Em [7] uma visão geral das tecnologias adotadas, no desenvolvimento deste componente, pode ser obtida.

1.4. Organização do Relatório

O capítulo 2 contém uma breve explicação da utilização da tecnologia XML no ambiente de desenvolvimento JAVA (as tecnologias empregadas no desenvolvimento do código). Detalhes estão além do escopo deste trabalho e podem ser obtidos em [5]. O capítulo 3 lista as taxas atualmente utilizadas pela UFG, tece alguns comentários sobre elas e o capítulo 4 apresenta artefatos produzidos ao longo do desenvolvimento deste componente. O capítulo 5 fornece os artefatos de projeto e, por último, o capítulo 6 fornece a conclusão.

2. JAVA E XML

A linguagem de programação Java foi desenvolvida pela Sun Microsystems no início da década de 90. Java é uma extensão da linguagem C++; dessa forma, os programadores familiarizados com a linguagem C++ podem compreender Java mais facilmente.

Java é considerada útil para a construção de aplicações distribuídas, permitindo que vários computadores sejam acessados através de uma rede e por diversos usuários localizados remotamente.

Uma característica importante é que a linguagem Java é multiplataforma, ou seja, um programa escrito em Java pode ser executado em qualquer plataforma (sistema operacional combinado com hardware) sem necessidade de alterações no código-fonte. Esta funcionalidade é possível se existir, na plataforma alvo, uma máquina capaz de executar bytecodes.

XML é uma metalinguagem definida como um subconjunto de SGML, cujo objetivo é fornecer os benefícios não existentes em HTML e ser mais fácil de utilizar do que SGML.

Em XML, projetistas podem criar seus próprios elementos de acordo com a aplicação que está sendo modelada, dando importância ao conteúdo e à estrutura da informação, sem se preocupar com a apresentação. Para que um parser XML verifique se um documento está correto ou não (parser de validação), ele processa inicialmente seu DTD correspondente, para verificar a estrutura do documento. Vários parsers, disponíveis gratuitamente no WWW, verificam (análise léxica) e validam (análise sintática) documentos XML de acordo com seu DTD associado. Além desses, há também parsers que não exigem a presença de um DTD e que, portanto, somente validam seus elementos.

Outros benefícios disponíveis no WWW são os editores de documentos XML. Eles permitem a criação simplificada de documentos XML (com ou sem o DTD), possibilitando inserir e eliminar elementos, propriedades e informações associadas, mudar parte da estrutura para outro local e validar o documento. Exemplos de editores para documentos XML são: Microsoft XML Notepad, XML Spy, XED etc.

Pode-se notar que o documento XML não trata a apresentação das informações, mas somente o conteúdo a ser apresentado. Sendo assim, existe a

necessidade da utilização de outro recurso que é responsável pelos atributos de apresentação. Esta tarefa pode ser realizada através da utilização de parsers específicos ou linguagens apropriadas para associar estilos ao conteúdo de um documento XML.

No contexto da utilização de XML, Java entra como um fator de essencial importância, principalmente no que diz respeito à conversão de um documento XML em um documento HTML. Através da implementação de um parser nesta linguagem, é possível recuperar os elementos dos arquivos DTD e XML, aplicar as estruturas e estilos presentes no documento XSL e CSS, gerando finalmente um documento HTML que contém o resultado que deve ser apresentado no browser. Segundo, Java fornece um formato portável de dados que complementa adequadamente o código portável de Java.

DOM, a API utilizada neste projeto, significa Document Object Model. DOM trata a informação armazenada em nosso documento XML como um modelo de objetos hierárquicos. Cria uma árvore de nós (baseado na estrutura e na informação do documento XML); o acesso à informação do documento pode ser através de interações com essa árvore. DOM preserva a seqüência dos elementos lidos a partir dos documentos XML, porque ele trata tudo como se fosse um documento. Isto justifica seu nome: “modelo de objeto do documento”. Se DOM for utilizado como o modelo de objetos em Java para a informação armazenada no documento XML, então não é necessário se preocupar com a outra API. Contudo, se considerar que DOM não é um bom modelo de objetos para tratar a informação armazenada no documento XML, então talvez compense estudar a outra API. Ela é útil quando temos que criar nossos próprios modelos personalizados de objetos. Mais detalhes poderão ser obtidos em [6].

3. TAXAS UTILIZADAS PELA UFG

Seguem abaixo as taxas utilizadas pela UFG descritas no Ofício Circular/PROAD nº 017/2002 do dia 28 de julho de 2002, contendo os valores atualizados no momento em que este texto foi escrito (24/03/2003).

Descrição	Valor
2º grau do Colégio de aplicação	25,00
Pré-Graduação da Escola de Música	24,00
Cursos de Atualização ou Extensão a partir de	16,00
Cursos de Especialização ou Aperf. A partir de	41,00
Cursos de Mestrado(por semestre/trimestre a partir de)	42,00
Cursos de Doutorado(por semestre/trimestre a partir de)	42,00
Matrícula de Aluno Especial	62,00
Trancamento da matrícula	1,0
Processo de Matrícula Extemporânea	1,5
Concurso Magistério-Professor Auxiliar e Substituto	75,00
Concurso Magistério-Professor assistente	84,00
Concurso Magistério- Professor Adjunto	105,00
Concurso Magistério-Professor Titular	125,00
Concurso Magistério- Professor Visitante	125,00
Concurso de Livre Docência	163,00
Concurso Público Administrativo (***)	0,00
Curso Espec. e/ou Aperf. Atuali.Exten. a partir de	9,00
Curso de Mestrado a partir de	12,00
Curso de Doutorado a partir de	21,00
Proc. De Atestados de Vagas Complemento Habilitação	77,00
Declarações que Demandam Pesquisa de Arquivo	14,00
Declar. Simples Exp. P/ s,c. Unid. Coord. Cursos, etc.	6,00
Proc. De Prorrogação Prazo p/ Integral Curricular	22,00
Currículo do Curso (Portaria Ministerial 515/79)	14,00
De curso Intens. Prepar. Prof. De Ensino Médio (CADES)	19,00

De Curso de especialização	25,00
De aperfeiçoamento e atualização	17,00
De extensão universitária	17,00
De curso de 2º grau	12,00
Certidão de conclusão de curso	14,00
2ª via de Qualquer Certificado Acima	77,00
Processo de Expedição e Registro de Diploma de Graduação	42,00
Processo de Reavaliação (Avaliação) de Diplomas	559,00
Diploma de Curso de 2º grau	12,00
Processo de Apostilamento de diploma	14,00
Processo de esp. de 2ª via de diploma de Graduação	201,00
Processo de registro de Diploma de outras IES	42,00
Guia de transferência	74,00
Histórico escolar	14,00
Processo de Mudança de curso / habilitação	24,00
Processo de Reingresso	42,00
Processo de Cancelamento de Cadastro	14,00
Cadastramento	43,00
Proc. de Aproveitamento de Disciplina	42,00
(Atest. Vaga , Mudanc. Curso,Reingres.,Aprov. Disciplina	75,00
Requerimento Colação de Grau Época Especial	62,00
Programa de Disciplina (Por Disciplina)	1,00
Fornecimento de Etiquetas Gomadas Cadastrais cada 1000 Um.	62,00
Cópias de Editais de Licitação	6,00

Figura 3.1. Tabela com as taxas da UFG

Conforme a tabela acima, uma taxa possui uma descrição e o valor associado. Atualmente o conjunto de taxas compreende cerca de 48 taxas. Embora pequeno, este número pode se expandir e, mesmo com a dimensão atual, localizar uma taxa pode exigir que o interessado conheça toda a descrição. Para evitar esta dependência, a cada taxa foi associado um código. Dessa forma, conforme a tabela, "Processo de mudança de curso" não será referenciada por esta seqüência de caracteres, mas pela sigla PMC, por exemplo, caso seja amplamente utilizada.

4. REQUISITOS

4.1. Descrição do componente "@ -TAXAS"

O componente deverá incluir, alterar e excluir taxas, além de manter informações de todo o histórico das taxas (período inicial de vigência e seu respectivo valor). Além disso, a criação de uma interface de software fará com que, tanto aplicativos que fazem parte do SISPG, quanto outros aplicativos utilizem essas informações, facilitando e ampliando o acesso às taxas.

4.2. Requisitos Funcionais

Os requisitos funcionais são formados pelos casos de uso do diagrama abaixo:

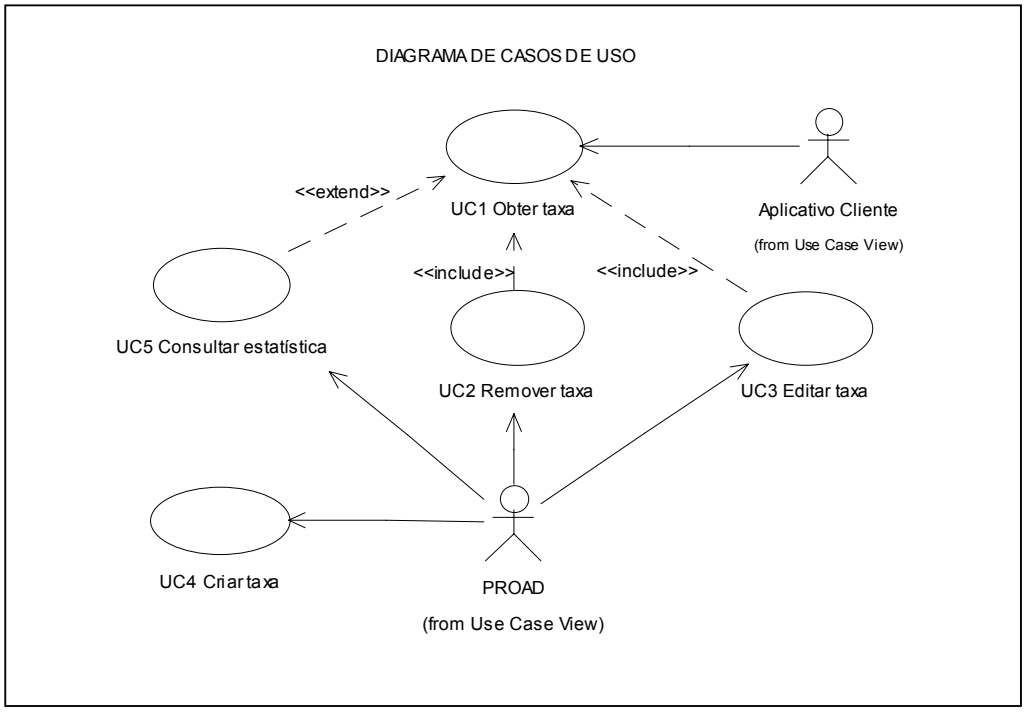

Figura 4.1 – Diagrama de Casos de Uso

4.3. Descrição dos Casos de Uso

UC1 Obter Taxa

Todas as taxas poderão ser consultadas, tanto pelo aplicativo que faz a manutenção, quanto por outros softwares através da interface do componente. Além de permitir que o usuário possa ter a localização de uma taxa facilitada por mecanismo de busca a ser oferecido por este caso de uso, o usuário também poderá consultar estatísticas.

- Aplicativos clientes buscam informações sobre taxas;
- PROAD obtêm informações sobre taxas;

Fluxo Normal:

1. Aplicativos clientes ou PROAD solicitação informações sobre uma taxa;
2. Informações da taxa são disponibilizadas;
3. Caso de uso é finalizado.

UC2 Remover Taxa

A remoção de qualquer taxa cadastrada deverá ser aceita pelo software, contudo, a cada pedido de exclusão deverá ser aberta uma janela de confirmação.

- Taxa é removida somente pelo PROAD;

Pré-condição:

A taxa a ser removida tem que estar selecionada.

Fluxo Normal:

1. PROAD seleciona taxa;
2. PROAD solicita remoção da taxa;
3. É mostrada uma confirmação de solicitação de remoção da taxa;
4. PROAD confirma solicitação;
5. Taxa é removida;
6. Caso de uso é finalizado.

Fluxo Alternativo:

1. PROAD seleciona taxa;
2. PROAD solicita remoção da taxa;
3. É mostrada uma confirmação de solicitação de remoção da taxa;
4. PROAD cancela solicitação;

5. Remoção da taxa é cancelada;
6. Caso de uso é finalizado.

UC3 Editar Taxa

As taxas poderão ser alteradas, porém com restrições. A alteração será permitida somente ao período corrente, valor corrente e à descrição da taxa.

- PROAD é o único responsável pela alteração de uma taxa;

Pré-condição:

A taxa a ser alterada tem que estar selecionada.

Fluxo Normal:

1. PROAD seleciona taxa;
2. PROAD solicita alteração da taxa;
3. PROAD efetua as alterações;
4. PROAD solicita gravação das alterações;
5. Taxa é alterada;
6. Caso de uso é finalizado.

Fluxo Alternativo:

1. PROAD seleciona taxa;
2. PROAD solicita alteração da taxa;
3. PROAD cancela alteração;
4. Alteração da taxa é cancelada;
5. Caso de uso é finalizado.

UC4 Criar Taxa

O software deverá permitir a inclusão de taxas, sendo que esta inclusão, somente será permitida se os campos de sigla, descrição e pelo menos um período da taxa estiver preenchido.

- PROAD cadastra novas taxas;

Pré-condição:

Os atributos sigla, descrição e pelo menos um período da taxa tem que estar preenchido.

Fluxo Normal:

1. PROAD solicita inclusão de taxa;
2. PROAD preenche as informações necessárias para a inclusão de uma taxa;

3. PROAD solicitação gravação da taxa;
4. Taxa é incluída;
5. Caso de uso é finalizado.

Fluxo Alternativo:

1. PROAD solicita inclusão de taxa;
2. PROAD cancela inclusão;
3. Inclusão da taxa é cancelada;
4. Caso de uso é finalizado.

UC5 Consultar Estatísticas

Deverá existir no software uma consulta com as informações estatísticas referente às taxas, tais como: maior taxa e seu respectivo valor, menor taxa e seu respectivo valor, valor médio das taxas, quantidade de taxas etc.

- Estatísticas referentes às taxas podem ser consultadas pelo PROAD;

Fluxo Normal:

1. PROAD solicita Geração de Estatística;
2. Estatística é gerada;
3. PROAD visualiza estatística;
4. Caso de uso é finalizado.

4.4. Requisitos Não Funcionais

Confiabilidade das informações

O software deverá garantir que as informações fornecidas pelo componente sejam confiáveis.

Facilidade de utilização

O software deverá ser simples o suficiente para que o usuário não tenha dificuldade em utilizá-lo. Teclas de atalho, menu de opções e validações com os usuários são algumas das maneiras de garantir a usabilidade desejada.

4.5. Modelo de Domínio

O modelo abaixo mostra que uma taxa pode estar associada a vários períodos e em cada período ela possui um valor.

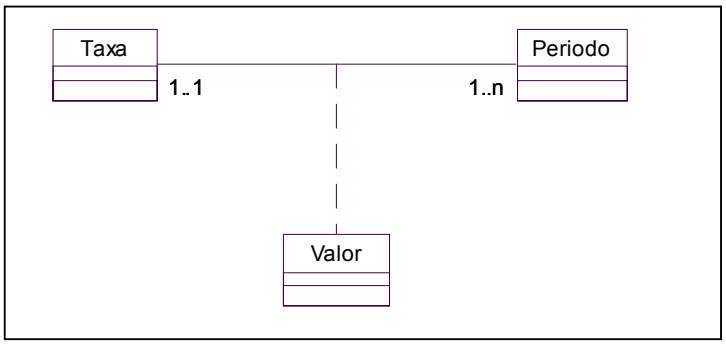

Figura 4.2 – Modelo de Domínio

4.6. Análise e Projeto

O diagrama de classes de análise mostra como é a relação entre uma taxa e seus respectivos períodos e valores.

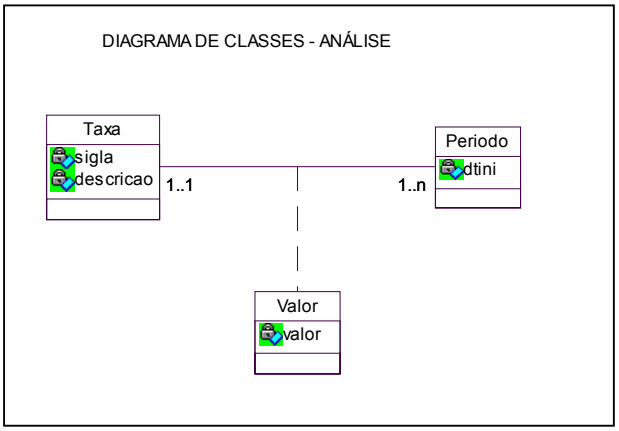

Figura 4.3.Diagrama de Classes - Análise

A figura abaixo mostra o diagrama de classes de projeto representando as associações e as relações de dependência entre as classes do componente.

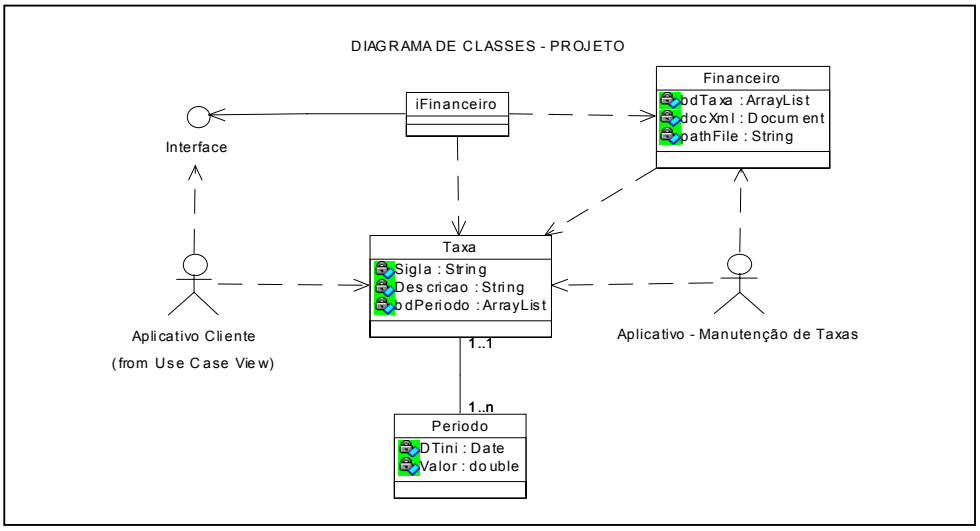

Figura 4.4. Diagrama de Classes - Projeto

A figura abaixo representa a seqüência da chamada aos métodos para que uma taxa seja adicionada no arquivo XML.

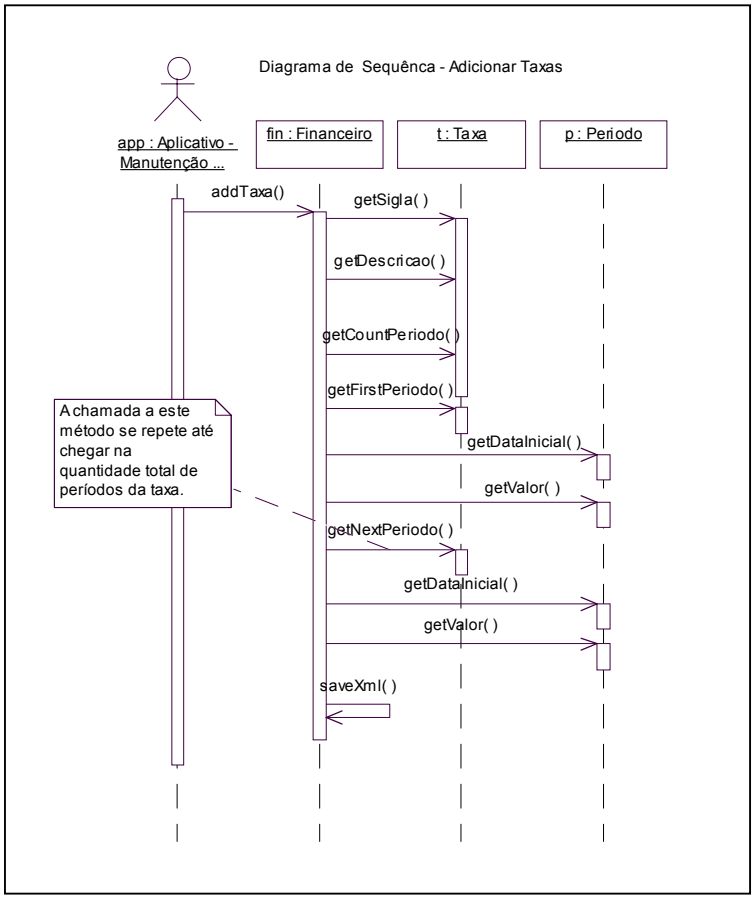

Figura 4.5 – Diagrama de Seqüência – Adicionar Taxas

A figura abaixo representa a seqüência da chamada aos métodos para que uma taxa seja editada no arquivo XML.

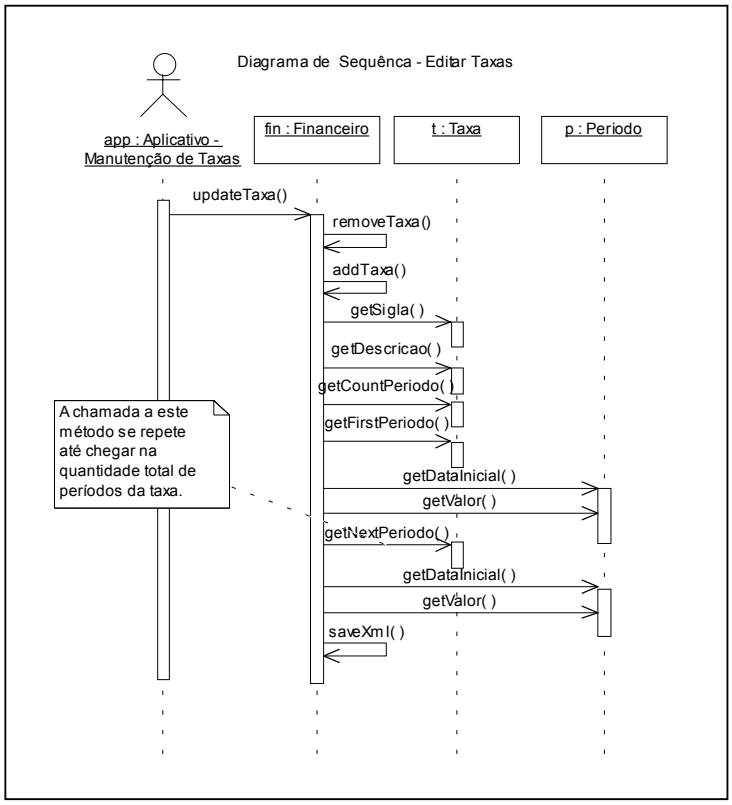

Figura 4.6 – Diagrama de Seqüência – Editar Taxas

A figura abaixo representa a seqüência da chamada aos métodos para que uma taxa seja removida do arquivo XML.

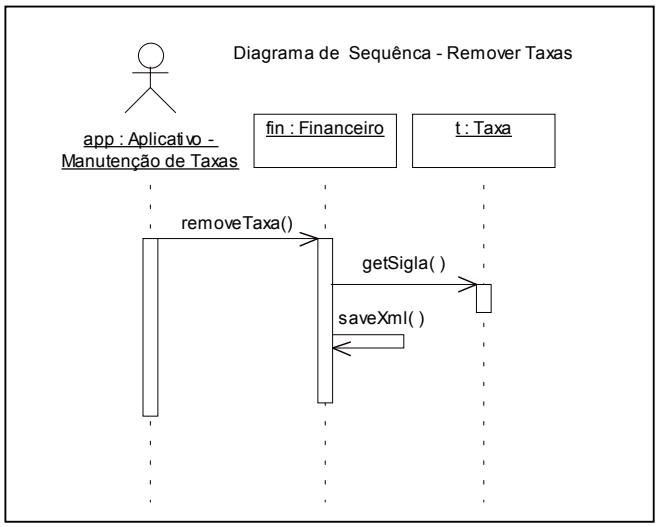

Figura 4.7 – Diagrama de Seqüência – Remover Taxas

A figura abaixo representa a seqüência da chamada aos métodos para que uma taxa seja consultada por um aplicativo cliente.

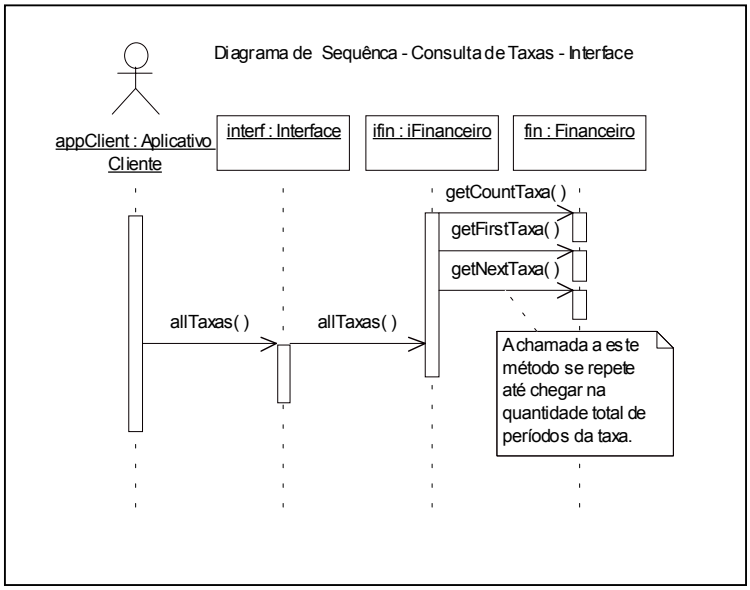

Figura 4.8 – Diagrama de Seqüência – Consulta de Taxas através da Interface

4.7. Modelo de Implantação

Abaixo segue o modelo de implantação, mostrando como será a estrutura que vai ser utilizada para que o componente seja acessado tanto internamente através da UFGNet, quanto externamente por aplicativos clientes através da internet.

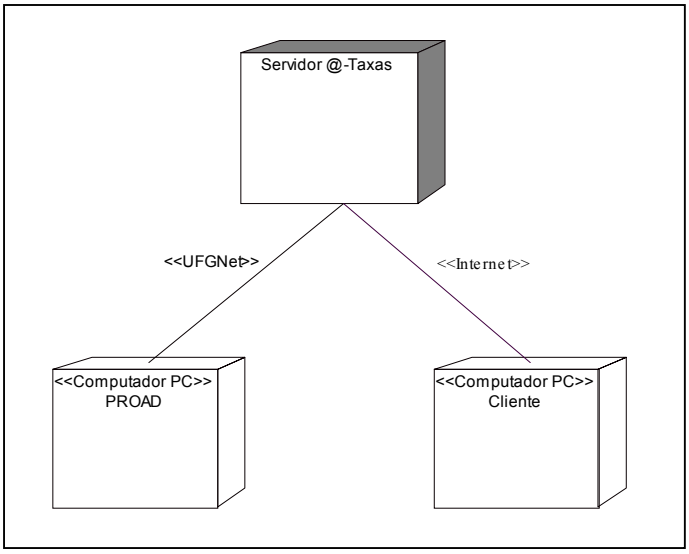

Figura 4.9 – Modelo de Implantação

4.8. Implementação

O código do componente é constituído por 12 classes, incluindo as classes que fazem parte do aplicativo de manutenção das Taxas. No anexo A são apresentadas as classes Taxa e Período, responsáveis por guardar as informações do componente.

4.9. Testes

Os testes foram implementados com o apoio da ferramenta *JUnit* 3.8 [8]. Este *framework*, escrito por Erich Gamma e Kent Beck, facilita a implementação de testes de unidade em *Java* e pode ser obtido gratuitamente (*open source*). Exemplos do emprego deste *framework* estão disponíveis no anexo C, que documenta os casos de testes realizados para o componente @-taxas.

A UML [4] foi a linguagem utilizada para fazer a modelagem em todas as atividades do projeto e a ferramenta de programação utilizada foi JAVA [5] acessando um arquivo XML para fazer manutenção e consulta dos dados.

4.10. Interface Gráfica de Software

Seguem abaixo as telas do software de manutenção das taxas:

A figura abaixo mostra a tela principal do aplicativo enfatizando o menu “Dados”.

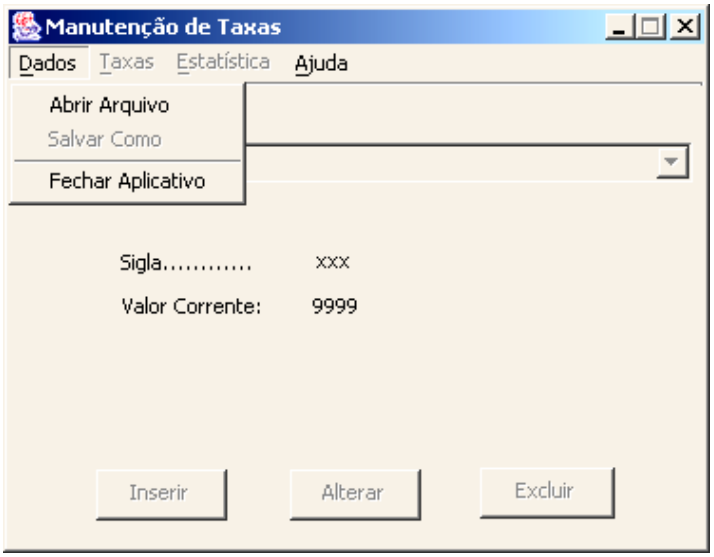

Figura 4.10 – Tela principal do aplicativo

A figura abaixo mostra a tela de inserção e alteração de uma taxa.

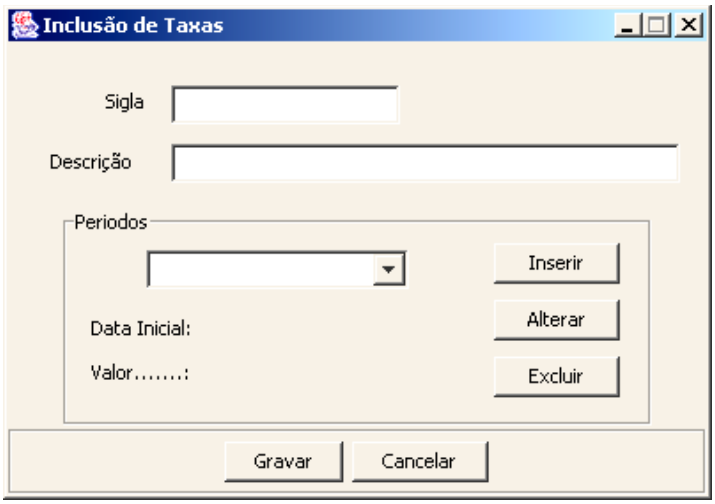

Figura 4.11 – Tela de inclusão de taxas

A figura abaixo mostra a tela de inclusão ou alteração de um período de uma determinada taxa.

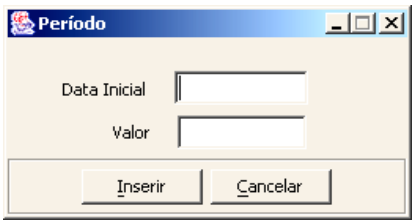

Figura 4.12 – Tela de manutenção de um período

A figura abaixo mostra a tela de consulta da Estatística das taxas cadastradas.

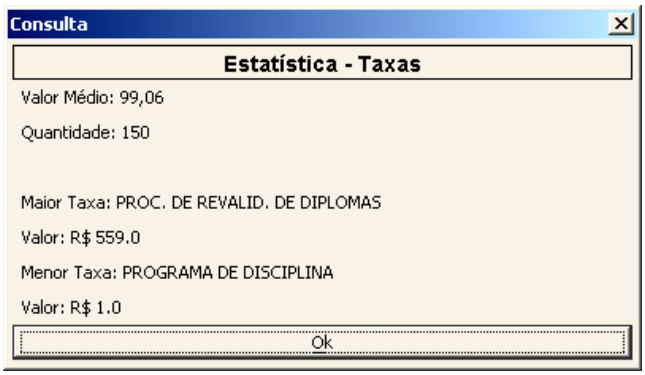

Figura 4.13 – Tela de consulta de Estatística

A figura abaixo mostra a tela de informações sobre o sistema.

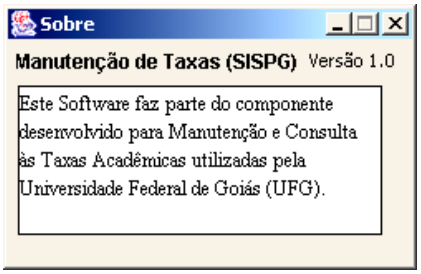

Figura 4.14 – Tela de informações sobre o aplicativo

5. PROJETO DE SOFTWARE

Neste capítulo estão contidas todas as funcionalidades internas do software, com explicação do funcionamento de todas as classes utilizadas no projeto, para que no caso de necessidade de se fazer manutenção, diminua a possibilidade de acontecer erros.

5.1. Apresentação das Classes

O componente para manutenção de taxas é composto por 7 (sete) classes, apresentadas abaixo:

Taxa	Classe base do software utilizada para armazenar as informações referentes a uma taxa.
Período	Classe base do software utilizada para armazenar as informações referentes a um período de uma determinada taxa.
Financeiro	Classe onde se encontram todos os métodos responsáveis pela manutenção e consulta das taxas, conseqüentemente manipulação do arquivo *XML* onde se encontra armazenados todos os dados.
XMLUtils	Esta classe é utilizada pela classe Financeiro para atualizar o arquivo *XML* no disco rígido.
App	Classe onde se encontra o método principal do software de manutenção de Taxas.
Frame	Classe que contém a interface gráfica da tela principal do software.
ManutTaxa	Classe que contém a interface gráfica para inserção e alteração das taxas.
ManutPeriodo	Classe que contém a interface gráfica para inserção e alteração dos períodos da taxa.
MostraEstat	Classe que contém a interface gráfica para consulta à Estatística das taxas.

Ifinanceiro	Classe que contém a implementação da interface do componente
Sobre	Classe que mostra informações sobre o sistema.
XmlFileFilter	Filtra na caixa de diálogo para mostrar somente arquivo com extensão ".xml".

Figura 5.1 – Classes de Projeto

5.2. Detalhamento das Classes

5.2.1. Classe Taxa

OBJETIVO PRINCIPAL

1. Armazenar as informações referentes a uma taxa.
2. Retornar os valores armazenados.

MÉTODOS DEFINIDOS

Taxa(String s, String desc, ArrauList a)

Constructor da classe Taxa, que recebe como parâmetro a sigla, descrição e os períodos em uma lista.

addPeriodo(Periodo p)

Recebe como parâmetro uma instância da classe período e acrescenta no atributo bdPeriodo.

getCountPeriodo()

Retorna a quantidade de períodos existentes na taxa.

getDescricao()

Retorna a descrição da taxa.

getFirstPeriodo()

Retorna o primeiro período da taxa.

getNextPeriodo()

Retorna o próximo período da taxa.

getSigla()

Retorna a sigla da taxa.

getValorCorrente()

Retorna o valor corrente da taxa.

setTaxa(String fSigla, String fDesc, ArrayList a)

Altera o valor dos atributos de taxa pelos valores passados como parâmetro.

setTaxa(String fSigla, String fDesc)

Altera o valor dos atributos de taxa pelos valores passados como parâmetro.

sortPeriodo()

Ordena os períodos da taxa em ordem decrescente de data.

5.2.2. Classe Periodo

OBJETIVO PRINCIPAL

1. Armazenar as informações referentes a um periodo.
2. Retornar os valores armazenados.

MÉTODOS DEFINIDOS

Periodo(Date d, double v)

Constructor da classe Periodo, que recebe como parâmetro a data inicial e o valor correspondente.

getDataInicial()

Retorna a data inicial do período.

getValor()

Retorna o valor correspondente ao período.

setPeriodo(Date fdtini, double fValor)

Altera os atributos do período.

5.2.3. Classe Financeiro

OBJETIVO PRINCIPAL

1. Manipular as informações referentes às taxas.
2. Interpretar arquivo *XML*.
3. Retornar informações referentes às taxas.

MÉTODOS DEFINIDOS

Financeiro(String file)

Constructor da classe Financeiro que recebe como parâmetro o caminho do arquivo *XML* e carrega todas as taxas armazenadas para a memória.

instancia(String file)

Faz parte do Design Pattern Singleton, que verifica se já existe uma instância da classe na memória e se caso exista, não é criada outra instância da mesma classe.

addTaxa(Taxa t)

Adiciona uma nova taxa no *DOM* e posteriormente no arquivo *XML*.

getCountTaxa()

Retorna a quantidade de taxas cadastradas.

getFirstTaxa()

Retorna a primeira taxa da lista de taxas.

getNextTaxa()

Retorna a próxima taxa da lista.

getTaxaPorSigla(String fSigla)

Retorna a taxa que a sigla seja igual ao parâmetro informado.

removeTaxa(Taxa t)

Remove a taxa passada como parâmetro.

removeTaxa(String fSigla)

Remove a taxa em que a sigla seja igual ao parâmetro informado.

saveXml(File file, boolean indenting, String publicID, String systemID)

Chama o método “serialize” da classe XMLUtils para atualizar o arquivo *XML* em disco.

setPathFile(String path)

Altera o valor do atributo “pathFile”, modificando o caminho do arquivo *XML* a ser manipulado.

updateTaxa(Taxa t)

Altera a taxa cadastrada na lista, posteriormente atualiza o arquivo *XML*.

5.2.4. Classe App

OBJETIVO PRINCIPAL

1. Iniciar o aplicativo de Manutenção de Taxas

MÉTODOS DEFINIDOS

App()

Constructor da classe App que instancia a classe Frame.

main(String[] args)

Método principal do aplicativo que instancia a classe App.

5.2.5. Classe Frame

OBJETIVO PRINCIPAL

1. Abrir janela principal do aplicativo.

MÉTODOS DEFINIDOS

Frame()

Constructor da classe Frame que chama o método “jbnit”.

processWindowEvent()

Finaliza o aplicativo caso a janela seja fechada.

alterar_actionPerformed(ActionEvent e)

Abre uma janela para alteração da classe selecionada, através da classe ManutTaxa.

excluir_actionPerformed(ActionEvent e)

Abre a janela de confirmação de exclusão da taxa selecionada.

inserir_actionPerformed(ActionEvent e)

Abre a janela de inclusão de taxa, através da classe ManutTaxa.

jItemAbrir_actionPerformed(ActionEvent e)

Chama o método "openFile" para abrir uma caixa de diálogo para selecionar o arquivo *XML* a ser manipulado.

jItemSalvar_actionPerformed(ActionEvent e)

Abre uma caixa de diálogo para salvar uma cópia do arquivo *XML* que estava sendo manipulado.

carregaTaxas()

Acessa a classe Financeiro para inserir todas as taxas cadastradas numa lista juntamente com um componente ComboBox, que será origem para as consultas no aplicativo.

gera_Estatistica()

Calcula a estatística das taxas cadastradas e as mostra na tela através da instanciação da classe MostraEstat.

jCbTaxa_itemStateChanged()

Atualiza as informações da taxa selecionada na tela após uma alteração do item selecionado na ComboBox.

JbInit()

Inicializa a interface gráfica da classe Frame.

openFile()

Abre uma caixa de diálogo para selecionar o arquivo *XML* a ser manipulado.

saveFile()

Abre uma caixa de diálogo para escolher onde será gerada uma cópia do arquivo *XML* que está aberto.

sortTaxa()

Ordena a lista das taxas em ordem crescente do atributo "Descrição" da classe "Taxa".

5.2.6. Classe Sobre

OBJETIVO PRINCIPAL

1. Abre janela com informações do sistema.

MÉTODOS DEFINIDOS

Sobre()

Constructor da classe Sobre que abre a janela com informações sobre o sistema.

5.2.7. Classe XmlFileFilter

OBJETIVO PRINCIPAL

1. Classe que filtra os arquivos a serem visualizados nas caixas de diálogo.

MÉTODOS DEFINIDOS

accept()

Filtra os arquivos no diretório selecionado.

getDescription()

Retorna a descrição a ser mostrada na caixa de diálogo para o tipo do arquivo.

5.2.8. Classe MostraEstat

OBJETIVO PRINCIPAL

1. Abre janela com informações estatísticas referentes às taxas.

MÉTODOS DEFINIDOS

MostraEstat()

Constructor da classe MostraEstat que abre a janela com informações estatísticas das taxas.

5.2.9. Classe ManutTaxa

OBJETIVO PRINCIPAL

2. Abre janela para manutenção das taxas.

MÉTODOS DEFINIDOS

ManutTaxa(Taxa t)

Constructor da classe ManutTaxa que abre a janela para manutenção da taxa que foi passada como parâmetro, caso o parâmetro seja nulo o aplicativo considera uma inserção de taxa.

btnAlterar_actionPerformed()

Abre a janela de manutenção do período selecionado.

btnExcluir_actionPerformed()

Abre a janela de confirmação de exclusão do período selecionado.

btnInserir_actionPerformed()

Abre a janela de inclusão de um novo período.

btnGravar_actionPerformed(ActionEvent e)

Chama os métodos da classe Financeiro para atualizar o arquivo *XML*.

JCbPeriodo_itemStateChanged()

Atualiza as informações do período selecionado na tela após uma alteração do item selecionado na ComboBox.

5.2.10. Classe ManutPeriodo

OBJETIVO PRINCIPAL

1. Abre janela para manutenção dos períodos de uma determinada taxa.

MÉTODOS DEFINIDOS

ManutPeriodo(Período p)

Constructor da classe ManutPeriodo que abre a janela para manutenção do período que foi passado como parâmetro, caso o parâmetro seja nulo o aplicativo considera uma inserção de período.

btnGravar_actionPerformed(ActionEvent e)

Insere ou altera o período na lista da classe que está sendo alterada.

5.2.11. Classe XMLUtils

OBJETIVO PRINCIPAL

1. Salvar o arquivo *XML* no disco.

MÉTODOS DEFINIDOS

Os métodos definidos nesta classe são padrões, e não precisam ser alterados, mesmo em caso de alteração do arquivo *XML*.

5.2.12. Classe Ifinanceiro

OBJETIVO PRINCIPAL

1. Implementar os métodos para a interface com outros aplicativos.

MÉTODOS DEFINIDOS

allTaxas()

Método que retorna a lista de todas as taxas.

getTaxaPorSigla(String fsigla)

Método que retorna a taxa correspondente à sigla passada como parâmetro.

6. CONCLUSÃO

O @-taxas contribui com o desenvolvimento do SISPG ao disponibilizar serviços que este sistema necessita para a realização de suas atribuições. Convém ressaltar que parte das atribuições do SISPG compreende os requisitos funcionais descritos na seção 4.2. O presente trabalho não só identificou estes requisitos, validados com os usuários, como também implementou um protótipo que serve para ilustrar a operação destas atribuições do SISPG. Embora seja questionável o emprego da implementação produzida por este trabalho como parte dos serviços do SISPG, pois se trata de um protótipo, ele contribui para validar os requisitos e ilustrar uma possível abordagem para uma implementação futura.

Um aspecto importante da implementação, e que deve ser ressaltado, é a utilização da tecnologia XML junto com a linguagem de programação Java, pois trata-se de uma tecnologia que vem despontando no meio da informática, principalmente na internet (veja detalhes da teoria na seção 2 e detalhes da implementação no anexo B e C).

A eliciação dos requisitos e o protótipo correspondente são elementos que implementam esta contribuição. Em particular, o protótipo auxilia a validar uma proposta de arquitetura para serviços computacionais de interesse de toda a UFG (veja detalhes na seção 4.8).

Vários outros componentes do SISPG ainda não foram definidos. O desenvolvimento destes outros componentes poderá se beneficiar do conteúdo deste texto quanto à forma, por exemplo, ou melhorando a abordagem empregada para descrever componentes que fazem parte do SISPG. Para facilitar o entendimento do que é o componente @-Taxas e a sua relação com os outros componentes do SISPG, segue abaixo o diagrama de componentes do SISPG, onde o @-Taxas pode ser visto como parte do pacote “financeiro”:

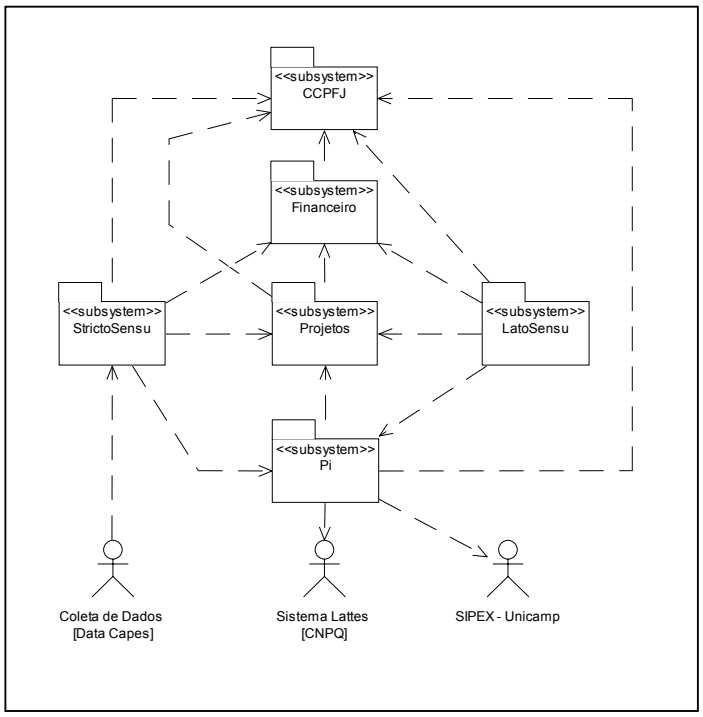

Figura 6.1. Diagrama de Componentes SISPG

Este trabalho teria pouca utilidade, contudo, sem os trabalhos futuros mencionados abaixo.

6.1. Trabalhos Futuros

Primeiro. Validar a forma com que o componente foi desenvolvido exige a validação da arquitetura de software para serviços da tecnologia da informação para a comunidade da UFG. Naturalmente este é um empreendimento maior do qual este componente é apenas um dos elementos. Contudo, os produtos do presente trabalho, auxiliam na identificação de tal arquitetura, seja no sentido de validar aquela empregada e fazer com que esta seja utilizada ao implementar novos serviços ou como referência de arquitetura inadequada e, portanto, que não deverá ser utilizada na implementação dos serviços computacionais no âmbito da UFG.

Segundo. A analise de desempenho e segurança são elementos que não podem ser negligenciados, embora não tenham sido considerados neste trabalho. Estes tópicos deverão ser abordados por trabalho que vise fornecer uma infra-estrutura, por exemplo, de segurança, para acesso confiável à considerável quantidade de serviços oferecidos pelo SISPG.

Naturalmente, muitos outros trabalhos são necessários ao desenvolvimento do SISPG. Aqueles destacados acima, portanto, estão apenas mais "próximos" do componente fruto do presente trabalho.

6.2. Considerações Finais

O presente trabalho também foi útil no processo de solidificação dos conhecimentos teóricos obtidos pelo autor ao longo do curso de especialização ao mesmo tempo que oferece uma abstração para a administração das taxas para os demais esforços de desenvolvimento do SISPG em andamento.

7. BIBLIOGRAFIA

[1] PRESSMAN, Roger S. Engenharia de Software. 3. ed. São Paulo: Makron Books, 1995.

[2] LUCENA, Fábio Nogueira de. Visão do SISPG, versão 0.9b

[3] LUCENA, Fábio Nogueira de. Development Case do SISPG, versão 0.4

[4] FURLAN, José Davi. Modelagem de Objetos através da UML. Makron Books, 1998.

[5] DEITEL, H. M.. Java Como Programar. 3. ed. Porto Alegre: Bookman, 2001.

[6] BRAY, T. e DeRose. Disponível em: http://www.icmc.sc.usp.br/~wdl/xml/monografia/sec2.htm. Visitado em: 20 fev. 2003.

[7] LARMAN, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and The Unified Process. Prentice-Hall, 2002.

[8] GAMMA, Erich, Disponível em: http://www.junit.org. Visitado em 22 fev. 2003.

[9] UFG Ofício Circular/PROAD nº 017/2002. Goiânia, 2002.

ANEXO A: Código em Java das principais classes do componente "@ - Taxas"

Segue abaixo o código fonte das classes "Taxa" e "Periodo", responsáveis por guardar todas as informações pertencentes às taxas e que são utilizadas por todas as outras classes do componente.

Taxa.Java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.text.SimpleDateFormat;

public class Taxa {
  private String sigla;
  private String descricao;
  private ArrayList bdPeriodo;

  private int cont;

  public Taxa() {
      bdPeriodo = new ArrayList();
  }

  public Taxa(String s, String desc, ArrayList a)
  {
    bdPeriodo = new ArrayList();
    setTaxa(s,desc,a);
  }

  public void setTaxa (String fsigla, String fdesc, ArrayList a)
  {
    sigla = fsigla;
    descricao = fdesc;
    a = sortPeriodo( a );
    bdPeriodo.addAll( a );
  }

  public void setTaxa (String fsigla, String fdesc )
  {
    sigla = fsigla;
    descricao = fdesc;
  }

  public String getSigla () {
    return sigla;
  }

  public String getDescricao () {
    return descricao;
  }

  public Periodo getFirstPeriodo() {
    cont = 0;
    return ( Periodo )bdPeriodo.get( 0 );
  }
```

```
  public Periodo getNextPeriodo() {
    cont++;
    Periodo p = new Periodo();
    p = ( Periodo )bdPeriodo.get( cont );
    return p;
  }

  public double getValorCorrente() {
    return (( Periodo )bdPeriodo.get( 0 )).getValor();
  }

  public void addPeriodo ( Periodo p ) {
    bdPeriodo.add( p );
  }

  public int getCountPeriodo() {
      return bdPeriodo.size();
  }

  public ArrayList sortPeriodo ( ArrayList bdPerAux ) {
    ArrayList aux = new ArrayList();
    ArrayList auxPer = new ArrayList();
    Periodo p = new Periodo();
    for ( int i = 0; i < bdPerAux.size(); i++) {
      p = (( Periodo )bdPerAux.get( i ));
      SimpleDateFormat formatter;
      Date fData = p.getDataInicial();
      formatter = new SimpleDateFormat("yyyy-MM-dd");
      String dataAux = formatter.format( fData );
      aux.add( dataAux );
    }
    Collections.sort( aux , Collections.reverseOrder() );
    for ( int i = 0; i < aux.size(); i++) {
      String data = (aux.get( i )).toString();
      for ( int j = 0; j < bdPerAux.size(); j++) {
        p = (( Periodo )bdPerAux.get( j ));
        SimpleDateFormat formatter2;
        Date fData2 = p.getDataInicial();
        formatter2 = new SimpleDateFormat("yyyy-MM-dd");
        String dataAux2 = formatter2.format( fData2 );
        if ( dataAux2.equals( data ) ) {
          auxPer.add( i , p );
          j = bdPerAux.size();
        }
      }
    }
    return auxPer;
  }
}
```

Periodo.Java

```
import java.util.Date;

public class Periodo {

  private Date dtini;
  private double valor;

  public Periodo() {
  }

  public Periodo(Date d, double v)
  {
    setPeriodo(d,v);
  }

  public void setPeriodo (Date fdtini, double fvalor) {
   dtini = fdtini;
   valor = fvalor;
  }

  public Date getDataInicial () {
    return dtini;
  }

  public double getValor() {
    return valor;
  }
}
```

ANEXO B: Modelo do arquivo XML utilizado no projeto

Segue abaixo um exemplo do arquivo *XML* utilizado para armazenar as informações das taxas. Este arquivo é composto pelas tags: xml, taxa, sigla, descricao, periodo, data e valor, seguindo a hierarquia e estrutura que é mostrada no modelo abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<XML>
   <TAXA>
      <Sigla>01D</Sigla>
      <Descricao>CURSOS DE ESPECIAL./APERF.</Descricao>
      <PERIODO>
         <Data>2002-07-01</Data>
         <Valor>41.0</Valor>
      </PERIODO>
   </TAXA>
   <TAXA>
      <Sigla>01E</Sigla>
      <Descricao>CURSO DE MESTRADO (SEM/TRIM)</Descricao>
      <PERIODO>
         <Data>2002-07-01</Data>
         <Valor>42.0</Valor>
      </PERIODO>
   </TAXA>
   <TAXA>
      <Sigla>01F</Sigla>
      <Descricao>CURSOS DE DOUTORADO (SEM/TRIM)</Descricao>
      <PERIODO>
         <Data>2002-07-01</Data>
         <Valor>42.0</Valor>
      </PERIODO>
   </TAXA>
   <TAXA>
      <Sigla>01G</Sigla>
      <Descricao>MATRÍCULA DE ALUNO ESPECIAL</Descricao>
      <PERIODO>
         <Data>2002-07-01</Data>
         <Valor>62.0</Valor>
      </PERIODO>
   </TAXA>
   <TAXA>
      <Sigla>02A</Sigla>
      <Descricao>TRANCAMENTO DE MATRÍCULA</Descricao>
      <PERIODO>
         <Data>2002-07-01</Data>
         <Valor>1.0</Valor>
      </PERIODO>
   </TAXA>
</XML>
```

ANEXO C: Casos de Teste do componente "@ - Taxas"

Seguem abaixo todos os casos de teste simulados no componente "@ - Taxas". Para que esses testes fossem efetuados utilizou-se a ferramenta *JUNIT* 3.8 e para cada classe testada foi criada uma classe de teste correspondente com as iniciais "CT"(caso de teste) mais o nome da classe, por exemplo: para a classe "Periodo" foi criada uma classe correspondente com o nome "CTPeriodo" contendo as validações de todos os métodos referentes à classe "Periodo". Para cada classe de teste será mostrado o resultado do teste e o seu respectivo código fonte.

CT1 : Periodo

A classe de teste abaixo tem dois métodos que validam, respectivamente, o método que atribui valores aos atributos da classe "Periodo" e o constructor da classe que já recebe como parâmetro os valores que deverão ser preenchidos na instanciação da classe.

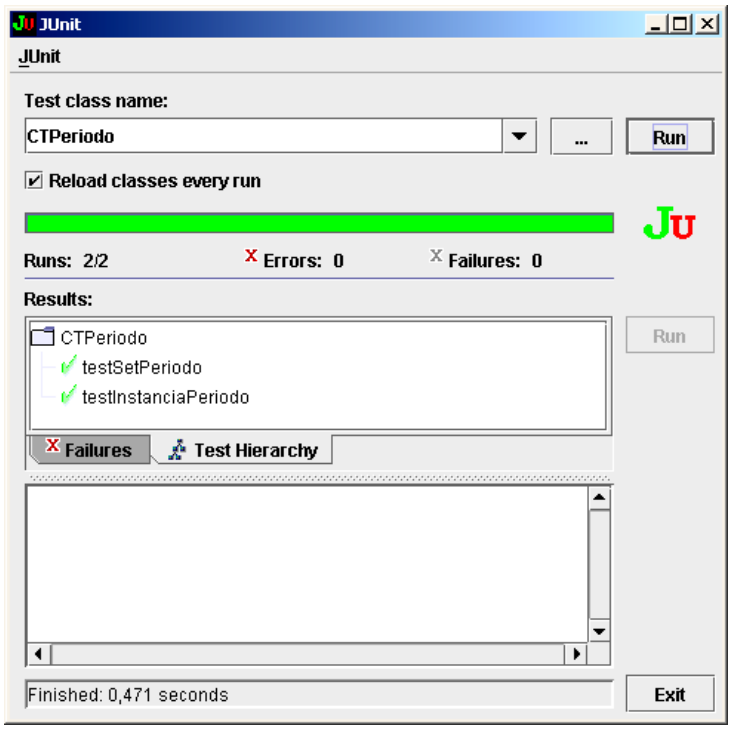

Figura C.1 – Execução do teste da classe "Periodo"

```
import junit.framework.*;
import java.util.GregorianCalendar;
import java.util.Date;

public class CTPeriodo extends TestCase {
      private Periodo p1;
      private Periodo p2;
      private Date data1;
      private Date data2;
      private double val1;
      private double val2;
```

```
	public static void main(String args[]) {
		junit.swingui.TestRunner.run(CTPeriodo.class);
	}

	protected void setUp() {
		p1 = new Periodo();
		GregorianCalendar gc1 = new GregorianCalendar(2005,0,10);
		GregorianCalendar gc2 = new GregorianCalendar(1995,0,15);
		data1 = gc1.getTime();
		data2 = gc2.getTime();
		val1 = 120.00;
		val2 = 100.00;
		p1.setPeriodo( data1 , val1 );
		p2 = new Periodo( data2 , val2 );
	}

	public void testSetPeriodo() {
		assertTrue( p1.getDataInicial() == data1 );
		assertTrue( p1.getValor() == val1 );
	}

	public void testInstanciaPeriodo() {
		assertTrue( p2.getDataInicial() == data2 );
		assertTrue( p2.getValor() == val2 );
	}

}
```

CT2 : Taxa

A classe de teste abaixo tem quatro métodos que validam, respectivamente, o método que atribui valor ao atributo "Sigla", o método que atribui valor ao atributo "Descrição", o método que retorno o valor atual da taxa consultada e o método que retorna a quantidade de períodos de uma determinada taxa.

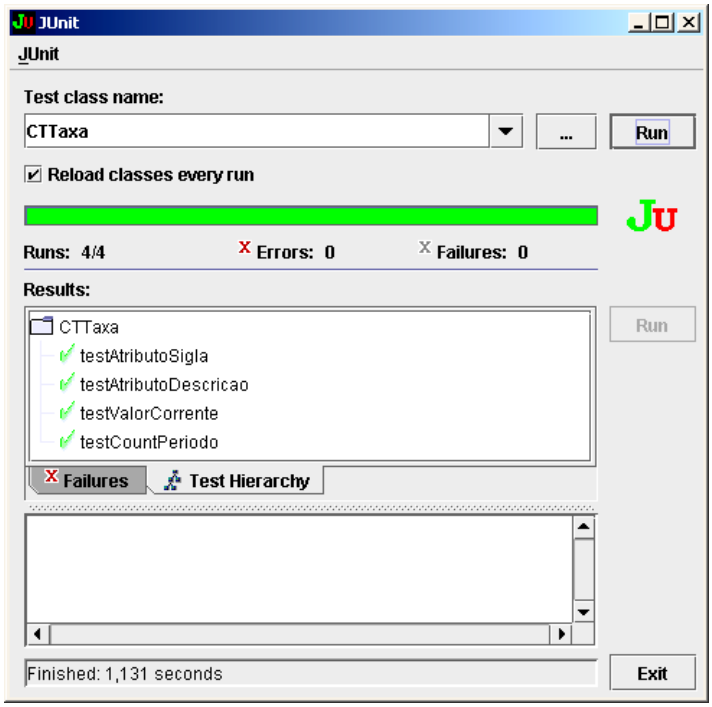

Figura C.2 – Execução do teste da classe "Taxa"

```
import junit.framework.*;
import java.util.GregorianCalendar;
import java.util.Date;

public class CTTaxa extends TestCase {
      private String sigla;
      private String descricao;
      private Taxa t;
      private Periodo p1;
      private Periodo p2;
      private Date data1;
      private Date data2;
      private double val1;
      private double val2;
      private int cont;

      public static void main(String args[]) {
            junit.swingui.TestRunner.run(CTTaxa.class);
      }

      protected void setUp() {
```

```
            cont = 0;
            t = new Taxa();
            p1 = new Periodo();
            p2 = new Periodo();
            GregorianCalendar gc1 = new GregorianCalendar(2000,0,1);
            GregorianCalendar gc2 = new GregorianCalendar(1998,0,1);
            data1 = gc1.getTime();
            data2 = gc2.getTime();
            val1 = 120.00;
            val2 = 100.00;
            p1.setPeriodo( data1 , val1 );
            p2.setPeriodo( data2 , val2 );
            t.setTaxa( sigla , descricao );
            t.addPeriodo( p1 );
            cont = cont+1;
            t.addPeriodo( p2 );
            cont = cont+1;
      }

      public void testAtributoSigla() {
            assertEquals( sigla , t.getSigla() );
      }

      public void testAtributoDescricao() {
            assertEquals( descricao , t.getDescricao() );
      }

      public void testValorCorrente() {
            assertTrue( val1 == t.getValorCorrente() );
      }

      public void testCountPeriodo() {
            assertTrue( t.getCountPeriodo() == cont );
      }
}
```

CT3 : Financeiro

A classe de teste abaixo tem três métodos que validam, respectivamente, o método que retorna a quantidade de taxas armazenada num arquivo XML, o método que inclui uma taxa e o método que remove uma taxa.

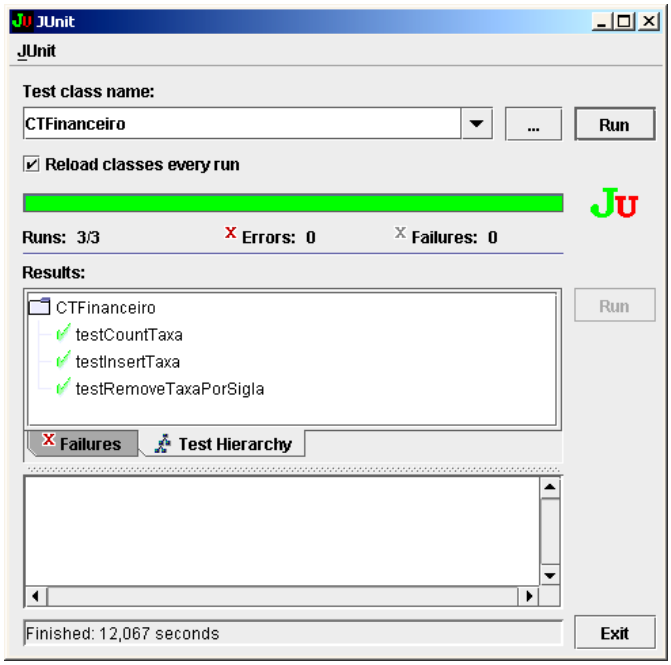

Figura C.3 – Execução do teste da classe "Financeiro"

```
import junit.framework.*;
import java.util.GregorianCalendar;
import java.util.Date;
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.xerces.parsers.DOMParser;

public class CTFinanceiro extends TestCase {
      private String pathFile;
      private String sigla;
      private String descricao;
      private Taxa t;
      private Taxa t2;
      private Periodo p1;
      private Periodo p2;
      private Date data1;
      private Date data2;
      private double val1;
      private double val2;
      private int cont;
      private Financeiro fin;
      private Document docXml;
      private Element root;
      private DOMParser parser = null;
```

```
public static void main(String args[]) {
      junit.swingui.TestRunner.run(CTFinanceiro.class);
}

protected void setUp() {
      pathFile = "C://Taxas//CTTaxas.xml";
      fin = Financeiro.instancia( pathFile );
      t = new Taxa();
      p1 = new Periodo();
      p2 = new Periodo();
      GregorianCalendar gc1 = new GregorianCalendar(2000,0,1);
      GregorianCalendar gc2 = new GregorianCalendar(1998,0,1);
      data1 = gc1.getTime();
      data2 = gc2.getTime();
      val1 = 120.00;
      val2 = 5.00;
      p1.setPeriodo( data1 , val1 );
      p2.setPeriodo( data2 , val2 );
      sigla = "t1";
      descricao = "descricao";
      t.setTaxa( sigla , descricao );
      t.addPeriodo( p1 );
      t.addPeriodo( p2 );
      t2 = new Taxa();
      t2.setTaxa( "XXXXXX" , "ddddddddddddddddddd" );
      t2.addPeriodo( p1 );
      fin.addTaxa( t2 );
}

public void testCountTaxa() {
      try {
        File xmlFile = new File( pathFile );
        DocumentBuilderFactory             factory             =
        DocumentBuilderFactory.newInstance();
        DocumentBuilder db = factory.newDocumentBuilder();
        docXml = db.parse( xmlFile );
        root = docXml.getDocumentElement();
        NodeList taxaList = root.getElementsByTagName( "TAXA" );
        cont = fin.getCountTaxa();
        assertTrue( cont == taxaList.getLength() );
      }
      catch (SAXException sxe) {
            Exception  x = sxe;
            if (sxe.getException() != null)
              x = sxe.getException();
            x.printStackTrace();
      }
      catch (ParserConfigurationException pce) {
            pce.printStackTrace();
      }
      catch (IOException ioe) {
            ioe.printStackTrace();
      }
}

public void testInsertTaxa() {
      cont = fin.getCountTaxa();
      for (int i= 0; i < 5; i++) {
            fin.addTaxa( t );
      }
```

```
            int cont1 = fin.getCountTaxa();
            assertTrue( cont1 == 5+cont);
      }


      public void testRemoveTaxaPorSigla() {
            cont = fin.getCountTaxa();
            fin.removeTaxa( t2 );
            int cont2 = fin.getCountTaxa();
            assertTrue( cont2 == (cont-1));
      }
}
```