UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ALEXANDRE PRADO BARBOSA

LUAN BUKOWITZ BELUZZO

UM PROCESSO DE EXTENSÃO DE FRAMEWORK DE DOMÍNIO: UM ESTUDO DE CASO NO FRAMEMK (FRAMEWORK PARA FORMAÇÃO DE PREÇO DE VENDA)

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2013

ALEXANDRE PRADO BARBOSA

LUAN BUKOWITZ BELUZZO

UM PROCESSO DE EXTENSÃO DE FRAMEWORK DE DOMÍNIO: UM ESTUDO DE CASO NO FRAMEMK (FRAMEWORK PARA FORMAÇÃO DE PREÇO DE VENDA)

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Coordenação do Curso de Análise e Desenvolvimento de Sistemas (COADS), da Universidade Tecnológica Federal do Paraná.

Orientadora: Profª. Drª. Simone Nasser Matos.

PONTA GROSSA

2013

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

Um Processo de Extensão de Framework de Domínio:
Um Estudo de Caso no FrameMK (Framework para Formação de Preço de Venda)

por

ALEXANDRE PRADO BARBOSA
LUAN BUKOWITZ BELUZZO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 20 de dezembro de 2013 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profª Drª Simone Nasser Matos
Orientadora

Prof. MSc Willian Massami Watanabe
Membro titular

Profª. Drª. Helyane B. Borges
Responsável pelos Trabalhos
de Conclusão de Curso

Prof. MSc Vinicius Camargo Andrade
Membro titular

Profª. Drª Simone de Almeida
Coordenadora do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

RESUMO

BARBOSA, Alexandre Prado; BUKOWITZ, Luan Beluzzo. **Um Processo de Extensão de Framework de Domínio: Um Estudo de Caso no FrameMK (Framework para Formação de Preço de Venda).** 2013. 103 f. Trabalho de Conclusão de Curso Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2013.

O FrameMK é um *framework* de domínio usado para formação de preço de venda, sendo assim, incorpora métodos de precificação de produto existentes na literatura. Este *framework* está sendo desenvolvido pelo grupo de pesquisa em Sistemas de Informação, Linha de Engenharia de Software, desta instituição. O FrameMK já passou por várias alterações e atualmente está disponível na web. Em sua arquitetura falta incorporar os métodos de custeio: Marginal, ROIC e parte do ABC. Este trabalho estudou e adaptou um processo para extensão de *framework* de domínio para incluir os métodos de precificação inexistentes no modelo do FrameMK. Realizou também testes para validar as novas funcionalidades inseridas em seu modelo.

Palavras-chave: *Framework*. Método de Preço de Venda. Processo de Extensão.

ABSTRACT

BARBOSA, Alexandre Prado; BUKOWITZ, Luan Beluzzo. **Um Processo de Extensão de Framework de Domínio: Um Estudo de Caso no FrameMK (Framework para Formação de Preço de Venda).** 2013. 103 f. Trabalho de Conclusão de Curso Tecnologia em Analise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2013.

The FrameMK is a domain framework used to establish the selling price and includes pricing methods existing in literature. This framework is being developed by the research group in Information Systems, in area Software Engineering. The FrameMK has already been through several modifications and nowadays it's available in the web. Its architecture does not modeled methods: Marginal, ROIC and some requirements of the ABC. This work has studied and adapted an extension process of domain framework which was used to include product pricing methods in the architectural model of the FrameMK. Tests were performed to verify its new functioning included.

Key words: Framework. Sales Princing Method. Extension Process.

LISTA DE ILUSTRAÇÕES

Figura 1 - Custos no Comércio.....24
Figura 2 - Custos no Comércio – Cálculo SEBRAE. 25
Figura 3 - Moffsoft Calculator 2 – Tela Inicial. 26
Figura 4 - Moffsoft Calculator 2 - Cálculo do Preço do Produto. 26
Figura 5 - Moffsoft Calculator 2 - Cálculo do Preço do Produto. 27
Figura 6 - SolveIt: Tela Inicial. 27
Figura 7 - SolveIt: Cálculo do ROI. 28
Figura 8 - Esquema de Extensão para Frameworks Transversais. 33
Figura 9 – Processo Proposto para Extensão de *Frameworks* de Domínio. 34
Figura 10 – Opções do Método ABC. 43
Figura 11 - Diagrama de classe Atividade e subsistema Gerenciar Linha de Produção. 43
Figura 12 - Adição de *Link* para *ProductionLine* em JSP. 44
Figura 13 - Adição de método para redirecionamento de *ProductionLine* em *windowsMenuAbcAction.* 44
Figura 14 – Adição de *forward* para *windowFindProductionLineAction* sem método especificado. 45
Figura 15 – Criação de método para receber requisições sem especificação de método em *Production Line.* 46
Figura 16 - Busca de atributos de *ProductionLine* e chamada para *success.* 46
Figura 17 - Inserção da *tag action* para *windowFindProductionLineAction* e criação de *foward* para *success.* 47
Figura 18 - Adição de *tag definition* para .*windowFindProductionLine* para extensão de interface. 47
Figura 19 - Criação de JSP *windowFindProductionLine*. 48
Figura 20 - Diagrama de classe do FrameMK para inserção do subsistema *Product Line* do método ABC. 48
Figura 21 - Diagrama de classe do FrameMK para inserção do subsistema *Product* do método ABC 49
Figura 22 - Criação da classe *windowFindActivity*.jsp 50
Figura 23 - Criação da classe *WindowAddActivity*.jsp para adição e/ou edição de Atividade 51
Figura 24 – Controle de busca de linhas de produção por *WindowAddActivityController* no subsistema *Activity.* 51
Figura 25 - Diagrama de classe do FrameMK para inserção do subsistema *Activity* do método ABC. 52
Figura 26 - Diagrama de Classe do FrameMK para Adição de Menus do Método Marginal 53
Figura 27 – Diagrama de classe do FrameMK para inserção do subsistema *Controle de Produto* do método Marginal. 55

Figura 28 – Diagrama de classe do FrameMK para inserção do subsistema *Gerenciamento de Matéria-prima* do método Marginal. 58
Figura 29 – Diagrama de Classe do FrameMK para o *Gerenciamento de Associação de Produto* com *Matéria-prima* do método Marginal. 59
Figura 30 – Diagrama de Classe do FrameMK para *Gerenciamento de Custos Fixos* do método Marginal.....61
Figura 31 – Diagrama de Classe do FrameMK para *Cálculo* do Método Marginal....62
Figura 32 – Diagrama de Classe do FrameMK para *Persistência de Produtos* no método Marginal.....64
Figura 33 – Diagrama de Classe do FrameMK para *Persistência de Matéria-Prima* no método Marginal.....64
Figura 34 – Diagrama de Classe do FrameMK para *Persistência de Associação de Produto* com *Matéria-Prima* no método Marginal. 65
Figura 35 – Diagrama de Classe do FrameMK *para Persistência de Custos Fixos* no método Marginal.....65
Figura 36 – Diagrama de Classe do FrameMK para Adição de Menus do Método ROIC 67
Figura 37– Diagrama de Classedo FrameMK para *Gerenciamento de Tipos de Ativos* do Método ROIC.....68
Figura 38 – Diagrama de Classe do FrameMK para *Gerenciamento de Ativos* do método ROIC. 70
Figura 39 – Diagrama de Classe do FrameMK para *Cálculo* do Método ROIC. 72
Figura 40 – Diagrama de Classe do FrameMK para *Persistência de Tipos de Ativos*. 73
Figura 41 – Diagrama de Classe do FrameMK para *Persistência de Ativos*.....73
Figura 42 – Menu de opções do método ABC com todas as opções habilitadas.....78
Figura 43 – Resultado da Implementação dos Requisitos para o Subsistema *ProductionLine.* 79
Figura 44 – Resultado da Implementação dos Requisitos para o Subsistema *Product* do método ABC. 79
Figura 45 – Resultado de adição ou edição de Produtos do método ABC do FrameMK.....80
Figura 46 - Resultado da Implementação dos Requisitos para o Subsistema *Activity* do método ABC 80
Figura 47 - Resultado de adição ou edição de Atividade. 81
Figura 48 - Diagrama de Classes do subsistema "Gerenciar Produto" do método ABC. 89
Figura 49 - Adição de *Link* para *Product* no JSP *windowMenuABC.jsp.* 90
Figura 50 – Adição do *forward* para *windowFindProduct* sem método especificado. 90
Figura 51 - Criação do método para receber requisições sem especificação de método. 91
Figura 52 – Redirecionamento do *Forward* para a classe *WindowFindProductAction.* 91
Figura 53 - Adição de uma *Tag Action* para *WindowFindProduct.* 92
Figura 54 - Criação da classe *windowFindProduct.jsp.* 92

Figura 55 - Definição da classe .*windowFindProduct* no arquivo *tiles-defs-specific*.xml. 93
Figura 56 - Adição da *Tag Action* para *WindowAddProduct.* 93
Figura 57 - Método para chamar *forward success* e abrir tela adição e/ou edição. ..94
Figura 58 - Criação da classe *WindowAddProduct*.jsp para adição e/ou edição de produtos. 94
Figura 59 – Definição criada no arquivo *tiles-defs-specific.*xml da classe *WindowFindProduct.* 95
Figura 60 - Diagrama de Classes do subsistema "Gerenciar Atividade". 97
Figura 61 - Adição de *Link* para *Activity* no JSP *WindowMenuABC.jsp.* 98
Figura 62 - Inserção do *forward* para *windowFindActivity* sem método especificado. 98
Figura 63 - Adicionando *forward* para *windowFindProduct* sem método especificado. 99
Figura 64 - Redirecionamento do *Forward* para a classe *WindowsFindActivityAction.* 99
Figura 65 - Adição de uma *Tag Action* para *WindowFindActivity.* 100
Figura 66 - Definição da classe *.windowFindActivity* no arquivo *tiles-defs-specific.xml.* 100
Figura 67 – Adição da *Tag Action* para *WindowAddActivity.* 101
Figura 68 - Método para chamar *forward success* e abrir tela adição e/ou edição. 101
Figura 69 – Definição criada no arquivo *tiles-defs-specific.*xml da classe *WindowFindActivity.* 102

Gráfico 1 – Total de testes realizados. 77
Gráfico 2 – Tempo utilizado para a realização dos testes de unidade. 77

LISTA DE QUADROS

Quadro 1 - Subsistemas do Método ABC. 36
Quadro 2 – Funcionalidades por subsistema do método ABC. 37
Quadro 3 - Subsistemas do Método Marginal. 38
Quadro 4 – Funcionalidades por Subsistema do método Marginal. 39
Quadro 5 - Elementos da Arquitetura do Método ROIC. 40
Quadro 6 – Funcionalidades por Subsistema do método ROIC. 40
Quadro 7 – Modelo de caso de teste 75
Quadro 8 – Caso de Teste para habilitar *link* para Linha de Produção. 76
Quadro 9 – Caso de Teste na adição da *Tag Action* para *WindowAddProduct* em *struts-config*.xml. 76

LISTA DE SIGLAS

ABC	Custo Baseado em Atividades
BD	Banco de Dados
CAS	Custo de Atividade por Setor
Ca	Custo de Aquisição
CFA	Custos Fixos Aplicados
CFDAP	Custos Fixos Diretamente Atribuíveis ao Produto
CFP	Custo Final do Produto
CIPV	Custo Indireto de Produção Variáveis
CMP	Custos Marginais do Produto
CPP	Custo Pleno do Produto
CTA	Custo Total da Atividade
Df	Despesas Fixas
Dv	Despesas variáveis
DVVA	Despesas Variáveis de Venda e de Administração
FTs	*Frameworks* Transversais
GA	Giro Ativo
GPES	Grupo de Pesquisa em Engenharia de Software
Li	Lucro Líquido
ML	Margem Lucro
MP	Matérias-primas
PUPS	Porcentagem de Utilização do Produto por Setor
RGS	Recurso Gasto no Setor
ROI	*Return On Investment*
ROIC	Retorno Sobre o Capital Investido
TDR	Total Disponível Recurso
TRI	Taxa de Retorno Sobre Investimento
TUS	Total de Utilização do Setor
UPS	Utilização do Produto por setor

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 13
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.2 ORGANIZAÇÃO DO TRABALHO 14
2 *FRAMEWORK* DE FORMAÇÃO DE PREÇO DE VENDA 15
2.1 *FRAMEWORKS*: UMA BREVE INTRODUÇÃO 15
2.1.1 *Framework* de domínio 16
2.2 *FRAMEWORK* DE DOMÍNIO EM PREÇO DE VENDA 17
2.2.1 Aplicativos de Formação de Preço de Venda 24
2.3 MODELO DO FRAMEMK 29
2.3.1 Equipe de Desenvolvimento 29
3 PROCESSO DE EXTENSÃO DO FRAMEMK 32
3.1 PROCESSOS DE EXTENSÃO DE *FRAMEWORKS* EXISTENTES NA LITERATURA 32
3.2 PROCESSO PROPOSTO PARA A EXTENSÃO DE *FRAMEWORK* 33
3.3 APLICAÇÃO DO PROCESSO PROPOSTO PARA EXTENSÃO DO FRAMEMK 35
3.3.1 Identificação e documentação de novos requisitos 35
3.3.2 Implementar Novo Requisito e Levantar Pontos de Junção no *Framework* ... 42
3.3.3 Testar novo Requisito 74
4 RESULTADOS 75
4.1 TESTE DE UNIDADE PARA OS NOVOS REQUISITOS 75
4.2 RESULTADO DA INSERÇÃO DOS NOVOS REQUISITOS NO PROTÓTIPO DO FRAMEMK 78
4.3 AVALIAÇÃO DO PROCESSO DE EXTENSÃO DO FRAMEMK 81
5 CONCLUSÃO 83
5.1 TRABALHOS FUTUROS 83
APÊNDICE A - MODELAGEM DO SUBSISTEMA *PRODUCT* 88
APÊNDICE B - MODELAGEM DO SUBSISTEMA *ACTIVITY* 96

1 INTRODUÇÃO

A criação de sistemas baseados nos conceitos de *frameworks* permite uma maior reusabilidade, manutenibilidade, extensibilidade e flexibilidade. Os *frameworks* podem ser classificados como de: domínio, aplicação e de suporte (TALIGENT, 1994).

Um *Framework de domínio* tem por objetivo focar em um contexto de domínio (TALIGENT, 1994), como por exemplo, o *Framework* para Formação de Preço de Venda (FrameMK) que se restringe ao domínio de preço de venda de produtos, sendo assim, tem como finalidade prover uma base para que métodos de precificação de produto possam ser implementados (GPES, 2013).

Para a construção do FrameMK, desenvolvido pelo grupo de pesquisa em Sistemas de Informação linha de pesquisa Engenharia de Software, foram estudados, modelados e codificados métodos de preficicação de produto, dentre eles: Sebrae (CRAZUSKI, FEITOSA, CORDEIRO, 2008), Pleno (SANTOS, 1994) e ABC (MARTINS, 2001). Porém, em sua arquitetura atual não são contemplados os métodos Marginal (SANTOS, 1994), ROIC (BARATA, 2003) e alguns subsistemas do ABC.

Neste trabalho foram estudados processos para extensão de *frameworks*, a saber, de Rajlich (1997), Saito e Yakamoto (2002) e Camargo e Masiero (2005) e criou uma adaptação para que fosse possível a extensão do FrameMK, incorporando a ele as funcionalidades faltantes. O processo proposto permitiu a adição de novos requisitos a estrutura do FrameMK, bem como a realização de testes para verificar se os requisitos implementados estavam corretos.

1.1 OBJETIVOS

Os objetivos geral e específicos estão descritos a seguir nas seções 1.1.1 e 1.1.2, respectivamente.

1.1.1 Objetivo Geral

Adaptar um processo de extensão de *frameworks* de domínio e aplicá-lo no *Framework* para Formação de Preço de Venda (FrameMK), para incorporar novas funcionalidades a sua estrutura funcionalidades de precificação que o tornem mais completo.

1.1.2 Objetivos Específicos

a) Identificar os processos para extensão de *frameworks* de domínio.
b) Analisar a arquitetura do FrameMK.
c) Levantar os requisitos faltantes do método ABC, ROIC e Marginal, para incorporar ao modelo FrameMK.
d) Codificar os requisitos identificados do método ABC.
e) Testar os requisitos implementados.

1.2 ORGANIZAÇÃO DO TRABALHO

O presente trabalho dispõe de cinco capítulos. O Capítulo 1 relata a motivação e os objetivos para o desenvolvimento desta pesquisa.

O Capítulo 2 apresenta uma abordagem geral sobre *frameworks*, relatando sobre os destinados a formação de preço de venda.

O Capítulo 3 descreve sobre os processos de extensão de *frameworks* e apresenta um processo adaptado para sua evolução e sua aplicação na extensão do *Framework* para Formação de Preço de Venda (FrameMK).

O Capítulo 4 narra alguns casos de teste usados para verificar se os requisitos foram inseridos corretamente ao FrameMK, bem como apresenta uma avaliação sobre o processo de extensão usado por este trabalho.

Por fim, o último capítulo expõe as conclusões finais e as propostas para trabalhos futuros que podem ser desenvolvidos a partir desta pesquisa.

2 *FRAMEWORK* DE FORMAÇÃO DE PREÇO DE VENDA

Este capítulo apresenta um apanhado geral sobre *frameworks*. A Seção 2.1 define *frameworks*. A Seção 2.2 enfatiza *frameworks* destinados a formação de preço de venda, descrevendo os métodos da literatura que podem ser usados na precificação. E por fim, a Seção 2.3 descreve o FrameMK (*Framework* de Formação de Preço de Venda) que está sendo desenvolvido pelo Grupo de Pesquisa em Sistema de Informação, linha Engenharia de Software, foco deste trabalho no qual serão adicionado novos métodos de formação de preço de venda.

2.1 *FRAMEWORKS*: UMA BREVE INTRODUÇÃO

Taligent (1994) assegura que *frameworks* têm a capacidade de resolver tanto os problemas que justificam sua criação quanto os que vão além da ideia inicial de seu desenvolvimento, sendo essa uma das razões pelas quais eles se diferem de aplicações independentes.

Alguns *frameworks* podem exibir características semelhantes a bibliotecas, contudo o que identificará realmente um *framework* é sua estrutura interna e a forma pela qual manipula as problemáticas, as quais são à base de sua formação (TALIGENT, 1994).

Um *framework* tem como meta prover uma estrutura básica referente a um determinado objetivo de sua criação e todos os sistemas que tirarem proveito dele, ou seja, derivarem desse, terão as características desta estrutura (SOUZA JUNIOR, 2002). Dessa forma, um *framework* consiste à base de um sistema.

Um *framework* pode até mesmo ser visto como um construtor, justamente pela sua característica de ser a base para uma grande quantidade de aplicações (MATTSSON, 1996).

Outra característica relevante é apontada por Silva (2000, p.32):

> *Frameworks* podem agrupar diferentes quantidades de classes, sendo assim, mais ou menos complexos. Além disto, podem conter o projeto genérico para

> um domínio de aplicação, ou construções de alto nível que solucionam situações comuns em projetos.

Dentre os *frameworks* existentes, tem-se uma classificação referente a seu reuso, sendo que os de caixa-branca (*White-box*) são estendidos usando conceitos de herança a partir de classes base do *framework,* os de caixa-preta (*Black-box*) provêm interfaces que possibilitam a composição de objetos para obtenção de um resultado final (FAYAD, SCHMID, 1997). Por fim, tem-se os de caixa-cinza (*Gray-box*) que podem ser chamados de híbridos e possibilitam que sejam estendidos a partir de herança e também permitem a utilização de interfaces para composição (YASSIN; FAYAD, 2000).

Outra classificação está relacionada ao seu escopo, no qual os *frameworks* de aplicação envolvem determinada especialidade que pode ser utilizada por vários clientes de domínio, os *frameworks* de suporte provêm serviços em nível de sistema e os *frameworks* de domínio são criados para um determinado domínio e se tornam especialistas (TALIGENT, 1994). Mais detalhes sobre *frameworks* de domínio serão apresentados na próxima seção, pois é o foco deste trabalho.

2.1.1 *Framework* de domínio

Um *framework* de domínio é aquele que (SOUZA JUNIOR, 2002, p. 36):

> ...apresenta uma fatia vertical de funcionalidade referente a um domínio em particular, como aplicações financeiras, por exemplo. É possível observar que existe uma relação entre *frameworks* verticais e de aplicação de negócios, apesar de pertencerem a classificações diferentes.

Durante o desenvolvimento de *frameworks* de domínio podem ser encontradas dificuldades, pois a arquitetura gerada é grande e necessita de pessoas especializadas e com experiência no domínio (BOSCH et. al, 1997). Outro problema encontrado está relacionado ao fator tempo, pois o domínio tratado está em processo de mudança devido ao futuro incerto e de acordo com o processo de criação do *framework* construído, esse pode ser aumentado, porque outros requisitos podem não terem sido contemplados em sua primeira versão.

Durante o desenvolvimento de *frameworks* de domínio é importante a identificação dos pontos de estabilidade e flexibilidade entre as aplicações que estão sendo analisadas (MATOS, 2008).

- *Frozen Spots (pontos de estabilidade)* são os pontos comuns entre as aplicações analisadas.
- *Os Hot Spots* (pontos de flexibilidade) são pontos específicos de cada aplicação.

Matos (2008) apresenta o seguinte exemplo para que os conceitos supracitados sejam melhores explicados:

> ... ao se analisar três aplicações-exemplo no domínio de jogo de corrida de carros, pode-se verificar que o requisito ou caso de uso *Inserir Senha* do jogador é o mesmo para as três aplicações. Por esse motivo, pode-se classificá-lo como sendo um aspecto comum (*frozen spot*) entre as aplicações-exemplo. Por sua vez, identificou-se que o caso de uso *Comprar Acessórios* só existia em uma das aplicações-exemplo. Assim, esse caso de uso pode ser classificado como um aspecto específico (*hot spot*) de uma das aplicações-exemplo analisadas.

A seguir serão descritos *frameworks* de domínio de formação de preço de venda, objeto de estudo desta pesquisa.

2.2 *FRAMEWORK* DE DOMÍNIO EM PREÇO DE VENDA

A formação do preço, de forma justa como é apresentada por Leão (2008), pode ser feita conforme a fórmula (1), também, pode ser apresentada levando em conta custos diretos e indiretos da instituição como é mostrado na fórmula (2).

$$Preço = custo + lucro + impostos \quad (1)$$

$$Preço = custo\ direto + custo\ indireto + lucro + impostos \quad (2)$$

Independentemente da base que se utilize para formação do preço de venda, este cálculo é importante para a instituição em questão, pois caso isto não seja feito de forma correta, pode aos poucos, levar a instituição a falência como apontado por uma pesquisa realizada pelo IBGE (Instituto Brasileiro de Geografia e Estatística) na qual 48,4% das empresas com mais de oito (8) anos vão a falência e 20% sobrevivem após um ano no mercado (LEÃO, 2008).

Dentro do escopo de domínio de preço de venda é necessário estudar a utilização de vários métodos, cada um com resultados diferentes a partir de perspectivas distintas para a elaboração do preço final de um produto, e isto é explicado a partir de alguns métodos descritos a seguir.

Método ABC

O método de Custeio Baseado em Atividades (ABC, em inglês *Activity-Based Costing*) tem como enfoque as atividades desenvolvidas pela empresa, além de associar as relações entre produtos e essas atividades (BORNIA, 2002).

Padoveze (1997, p. 357) por sua vez define este método como aquele que "procura aprimorar o custeamento dos produtos, através de mensurações corretas dos custos fixos indiretos, em cima das atividades geradoras desses custos, para acumulação diferenciada ao custo dos diversos produtos da empresa".

Quando se fala de atividades, tem-se a noção que é uma parcela de um processo, definida por um conjunto de recursos, sejam eles humanos, materiais, tecnológicos e financeiros e produz algum tipo de bem ou serviço. Também é possível subdividir uma atividade em várias tarefas (MARTINS, 2001).

Quanto a definição de atividades, elas podem ser definidas em uma ou mais por departamento de serviço. O que será o fator de destaque é a relevância daquela atividade quanto a geração de custo na produção (PADOVEZE, 1997).

As etapas relacionadas a este método são apresentadas por Bornia (2002). São elas:

a) Mapeamento das atividades;
b) Alocação dos custos;
c) Redistribuição dos custos das atividades indiretas até as diretas;
d) Cálculo dos custos dos produtos.

A primeira etapa faz menção ao mapeamento de atividades, as quais serão consideradas conforme seu grau de relevância na definição do custo do produto, também é possível que esse levantamento de custos possua um grau de detalhamento maior, com isso, o mapeamento será mais refinado.

Na segunda etapa se tem a definição de custos por atividades, que pode ser definida como etapa de rastreamento.

Na terceira etapa, faz-se uma distinção entre as atividades que estão diretamente ligadas aos produtos e as que não, separa-se então, o custo ligado às atividades, estas por sua vez as atividades diretas e as indiretas aos produtos.

Por fim, no que consiste ao cálculo dos custos dos produtos são encontrados focos de determinada atividade, estes são chamados de direcionadores.

Como foi possível observar na descrição deste método, não é viável construir uma fórmula genérica que abranja todas as regras de negócio e setores existentes nas indústrias. Para que um preço seja atribuído a determinado produto no método ABC é necessário que seja definido um custo por ele gerado a cada atividade por setor dentro da empresa. Por exemplo, em Wernke (2005) faz-se um levantamento das atividades de determinada empresa, assim como os gastos apresentados por cada uma dessas atividades no período de um mês.

Para que seja possível estabelecer um custo para cada atividade faz-se necessário a utilização de critérios de rateio das atividades, por exemplo: Número de horas consumidas, Aluguel: Metros quadrados ocupados e Energia: Consumo kWh. Também é necessário estabelecer o quanto esses recursos são usados dentro de cada setor da empresa.

Com o intuito de separar os gastos de cada atividade de acordo com sua utilização dentro de cada setor, têm-se a fórmula (3). Nota-se que as variáveis apresentadas na fórmula são o CAS(Custo de Atividade por Setor), o CTA (Custo Total da Atividade), o RGS(Recurso Gasto no Setor) e por fim o TDR(Total Disponível do Recurso).

$$CAS = CTA\ x\ \frac{RGS}{TDR} \quad (3)$$

A partir da fórmula (3) é possível obter o custo de cada atividade por setor da empresa, porém o que ocorre muitas vezes é que a empresa fabrica/desenvolve mais de um produto. Para que seja possível calcular o preço de cada um, primeiramente deve-se dividi-los dentro dos setores conforme sua utilização, então para que se tenha uma porcentagem de utilização do setor para determinado produto, tem-se a fórmula (4) que se utiliza de variáveis como $PUPS$(Porcentagem de Utilização do Produto por Setor), a UPS (Utilização do Produto por Setor) e o TUS (Total de Utilização do Setor).

$$PUPS = 100\ x\ \frac{UPS}{TUS} \quad (4)$$

Por fim, tem-se a fórmula (5) que se utiliza das fórmulas (3) e (4) para se obter o custo de cada produto por setor, já que é desejado saber custo final do produto (CFP). O que se deve fazer é somar o custo de determinado produto por todos os setores que ele passa, para que se tenha este valor.

$$CFP = \sum PUPS\ X\ CAS \quad (5)$$

Método SEBRAE-PR

Observa-se que existem definições importantes para a compreensão desse método, sendo (SEBRAEPR, 2008):

- Custo de aquisição: faz menção ao montante resultante de todo o processo de requisição ao recebimento do produto.
- A margem de contribuição: tem o trabalho de filtrar gastos em produtos menos rentáveis.
- Ponto de equílibro: pode ser definido considerando o valor do produto necessário para que todas as despesas vinculadas a ele sejam supridas. O lucro desejado é o resultado do que foi investido na

empresa, o lucro líquido é o desejado retirando-se todas as despesas e custos envolvidos.

A partir das definições anteriores, pode-se prosseguir ao cálculo do método (SEBRAEPR, 2008) como apresentado na fórmula (6):

$$\mathrm{Pe} = \mathrm{Df}/\ \mathrm{Mc} \tag{6}$$

Conforme a fórmula (6) é possível a realização do cálculo de Ponto de equilíbrio (Pe), se a despesa fixa (Df) for definida como 100,00 e uma margem de contribuição (Mc) de 40%, tem-se que o valor de Pe é de 250,00.

Para o levantamento do custo do preço de venda (Pv) tem-se a fórmula (7), utilizando-se de valores do custo de aquisição (Ca), das despesas fixas (Df), despesas variáveis (Dv) e lucro líquido (Ll) do produto. A partir dos valores apontados, tem-se o cálculo para o preço de venda:

$$\mathrm{Pv} = \frac{\mathrm{Ca}}{100\% - (\%\mathrm{Df} + \%\mathrm{Dv} + \%\mathrm{Ll})} \tag{7}$$

Método Custo Pleno

Também chamado de custeio total, integral, absorção ou RKW. Trata-se de uma somatória de todos os valores envolvidos na produção e também aqueles da própria empresa, dividindo todos os gastos pelo número de produtos, obtendo o valor do produto e a partir desse, adiciona-se o lucro e tem-se o valor do preço de venda (MARTINS, 2001).

Padoveze (1997, p. 419) comenta o método do Custo Pleno como sendo o mais utilizado e também o define como: “Tomam-se como base os custos industriais por produto e adicionam-se as taxas gerais de despesas administrativas e comerciais, despesas financeiras e margem desejada”.

Santos (1994) apresenta um cálculo para obtenção do valor do custo pleno do produto (CPP) a partir das matérias-primas (MP), custos indiretos de produção variáveis (CIPV), despesas variáveis de venda e de administração (DVVA), custos fixos diretamente atribuíveis ao produto (CFDAP) e custos fixos aplicados (CFA) como observado na fórmula (8).

$$\mathrm{CPP} = \mathrm{MP} + \mathrm{CIPV} + \mathrm{DVVA} + \mathrm{CFDAP} + \mathrm{CFA} \quad (8)$$

Quando Martins (2001, p.237) discute sobre o método em questão também aponta o seguinte fato:

> O mercado é o grande responsável pela fixação dos preços, e não os custos de obtenção dos produtos. É mais provável que uma empresa análise seus custos e suas despesas para verificar se é viável trabalhar com um produto, cujo preço o mercado influencia marcantemente ou mesmo fixa, do que ela determinar o preço em função daqueles custos ou despesas.

Método Custo Marginal

Para Santos (1994, p.127) os custos marginais são “custos acrescidos que podem ser diretamente relacionados com o que é produzido e vendido. São custos que não seriam incorridos se um produto fosse eliminado”.

Há uma maior liberdade na construção de preços quando se refere ao método custo marginal, pois o fator limitante do preço mínimo a ser estabelecido a um produto é definido a partir do total de custos desembolsados para a criação do mesmo (SANTOS, 1994).

Levando em consideração as informações anteriores, quando são calculados todos os custos envolvidos na produção de um produto, caso seja desejado incluir o valor da entrega do produto como parte do custo mínimo do mesmo, o valor de entrega pode ser adicionado ao montante já calculado de custos de produção, fazendo com que o valor mínimo do produto a ser suprido seja igual ao montante total dos custos envolvidos em sua produção mais o valor do pacote de entrega, sendo assim, não haverá prejuízos na produção.

A fórmula (9) permite calcular o custo de um produto e/ou serviço usando os custos marginais do produto (CMP), os valores de matérias-primas (MP), custos indiretos de produção variáveis (CIPV), despesas variáveis de venda e de administração (DVVA) e os custos fixos diretamente atribuíveis ao produto (CFDAP) (SANTOS, 1994).

$$\mathrm{CMP} = \mathrm{MP} + \mathrm{CIPV} + \mathrm{DVVA} + \mathrm{CFDAP} \quad (9)$$

Método ROIC

O Retorno Sobre o Capital Investido (ROIC, em inglês *Return on Invested Capital* faz menção ao lucro que é obtido pela empresa e o Investimento é o montante utilizado com objetivo de ter o lucro final (BARATA, 2003).

Para que seja possível o cálculo da Taxa de Retorno Sobre Investimento (TRI) é necessário que dividir o lucro ($Lucro$) pelo ativo ($Ativo$), também dentro desta operação se pode utilizar vários tipos de lucro e ativo, mas é interessante que quando focados em determinado objetivo, ambos tem que referenciar ao mesmo foco, por exemplo, quando se utiliza o Lucro Operacional também se coloca o denominador Ativo Operacional. A fórmula (10) realiza o cálculo do preço se baseando no ROIC (BARATA, 2003):

$$TRI = \frac{Lucro}{Ativo} \quad (10)$$

Margem de Lucro Líquido (ML) e Giro Ativo (GA) são dois pontos a serem considerados dentro do ROIC, sendo o primeiro o lucro (em centavos) após todas as deduções feitas após a venda, já o segundo pode ser definido como o identificador de produtividade. Os dois cálculos são definidos segundo as fórmulas (11) e (12) (BARATA, 2003):

$$ML = \frac{Lucro\ Liquido}{Vendas\ Liquidas} \quad (11)$$

$$GA = \frac{Vendas\ Liquidas}{Ativo\ Total} \quad (12)$$

Barata (2003) também atenta ao fato de que empresas com a mesma taxa de retorno podem ter a Margem e Giro diferentes, desta forma, dependendo da

necessidade da empresa estes pontos podem ser focados distintamente para uma melhor taxa de lucro.

2.2.1 Aplicativos de Formação de Preço de Venda

Nas seções a seguir são apresentados alguns sistemas desenvolvidos utilizando métodos de formação de Preço de Venda. Dentre eles, foram selecionados aqueles que apresentavam uma maior facilidade ao cálculo do preço de venda segundo seus respectivos métodos. Os softwares escolhidos foram: Custos Comércio (CUSTOS COMÉRCIO, 2013), *Moffsoft Calculator 2* (MOFFSOFT, 2009) e *SolveIt* (SOLVEIT, 2009), os quais utilizam os métodos de custo SEBRAE, marginal e ROIC, respectivamente.

a) Aplicativo com implementação do método do Custo SEBRAE

O aplicativo ilustrado na Figura 1 foi desenvolvido no Excel e utiliza o método de precificação Sebrae.

Figura 1 - Custos no Comércio.
Fonte: CUSTOS COMÉRCIO (2013).

A opção de *Calcular Preços ICM* pode ser utilizada para o cálculo do preço de produto pelo método SEBRAE. Para que o cálculo seja efetuado é necessário que o usuário informe a descrição do produto e adicione informações como: custo de entrada, rateio dos custos fixos, desperdícios de comercialização, resultado líquido e

margem bruta. Após todos estes dados serem informados, o aplicativo retornará um preço para ser utilizado na venda, como é demonstrado pelo campo "Preço e Venda". A Figura 2 apresenta o cálculo de dois produtos por meio do aplicativo.

CÁLCULO DO PREÇO DE VENDA

ESTE NÃO É O SEU ENQUADRAMENTO

De acordo com o seu enquadramento, você terá que inserir na planilha CustosFixos os percentuais corretos das despesas de comercialização. Converse com o seu contador.

Voltar ao Menu | Gerar Lista de Preços

	Descrição do Produto	Custo entrada	Rateio Custos Fixos	Desp.de comercialização (DC)	Resultado Líquido	Margem bruta	Preço de venda	Atribuir Percentual de Lucro	Preço Sugerido o Mercado	Diferença
7	Sapato Samello ref 102	56,47	11,29	7,53	0,00	25,00%	75,30			75,30
8	Botina de camurça Zebu ref 678	77,99	15,60	10,40	0,00	25,00%	103,98			103,98
9	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
10	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
11	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
12	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
13	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
14	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
15	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
16	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
17	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
18	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
19	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
20	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
21	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
22	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
23	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
24	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
25	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
26	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
27	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!
28	0	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	25,00%	#DIV/0!			#DIV/0!

Figura 2 - Custos no Comércio – Cálculo SEBRAE.
Fonte: CUSTOS COMÉRCIO (2013).

A próxima subseção aborda a descrição de um *software* implementado com o método de custo Marginal.

b) Aplicativo com implementação do método do Custo Marginal

Segundo o seu fabricante a *Moffsoft Calculator 2* é uma calculadora rica de funções com uma interface simples, além de possuir funções que são encontradas em calculadoras comuns. Também apresenta funções financeiras, conversores de unidades, funções de cálculo de preço de produtos, entre outros. A Figura 3 ilustra a tela inicial de apresentação do software.

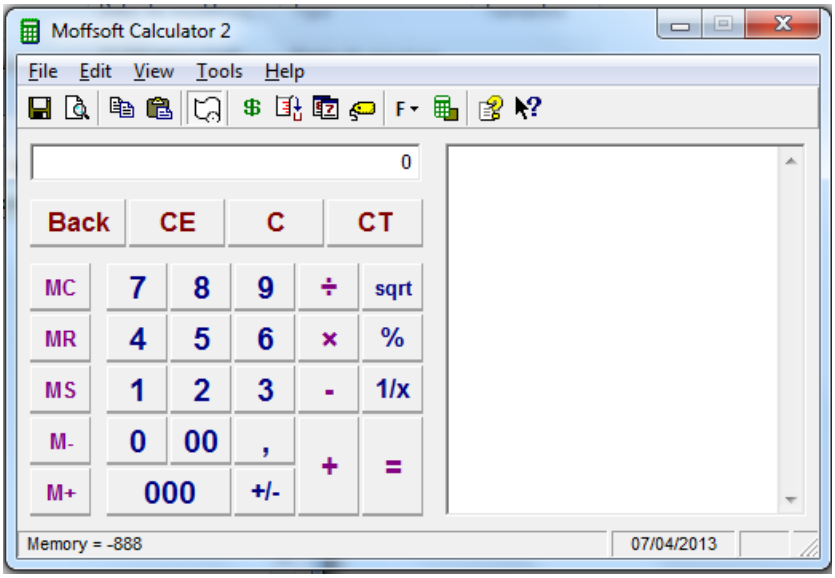

Figura 3 - Moffsoft Calculator 2 – Tela Inicial.
Fonte: Moffsoft (2009).

A função a qual será visualizada com mais abrangência é a de *Product Princing*, pois diz respeito ao cálculo do Lucro Marginal. Logo após, é possível calcular o preço marginal pela definição do custo do produto (*Cost of Goods Sold*) e também a margem de lucro (*Profit Margin*), como mostra a Figura 4.

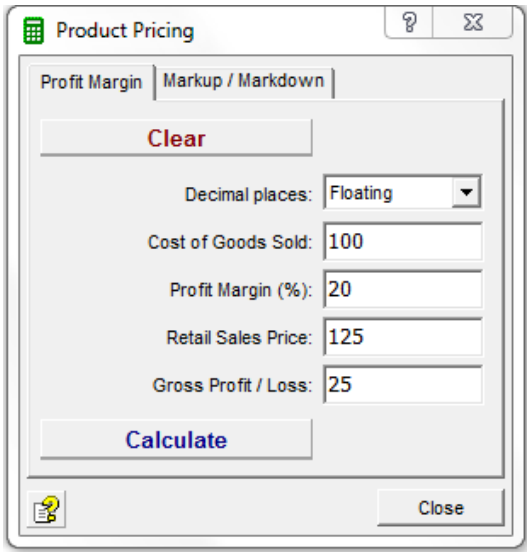

Figura 4 - Moffsoft Calculator 2 - Cálculo do Preço do Produto.
Fonte: Moffsoft (2009).

Após a inserção de todos os parâmetros necessários para o cálculo e o clique na opção “Calculate”, o resultado é apresentado na tela inicial como mostra a Figura 5.

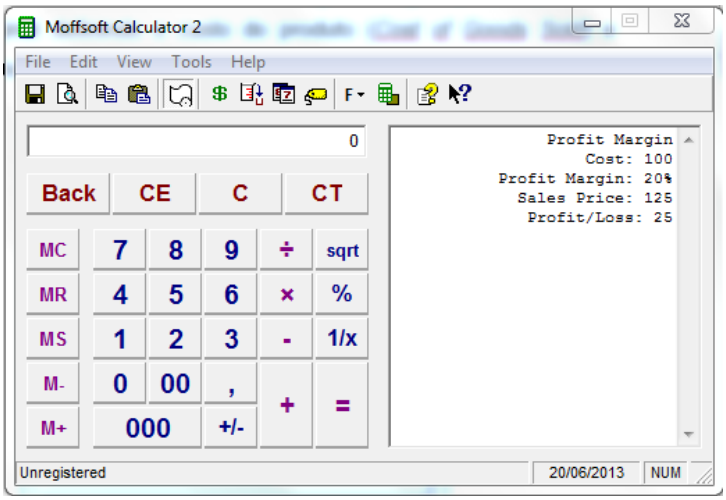

Figura 5 - Moffsoft Calculator 2 - Cálculo do Preço do Produto.
Fonte: Moffsoft (2009).

c) Aplicativo com implementação do método ROIC

SolveIt é apresentado como um software que não necessita de uma pessoa especializada para utilizá-lo, também é um conjunto de mais de quarenta calculadoras financeiras. A Figura 6 ilustra sua tela inicial.

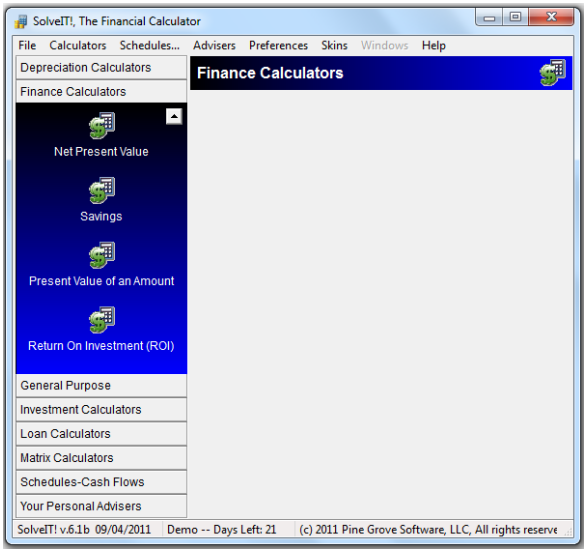

Figura 6 - SolveIt: Tela Inicial.
Fonte: SolveIt (2009).

A subseção a qual será mais focada é a de *Finance Calculatos* (Calculadoras Financeiras), pois tem como funcionalidade o cálculo do *Return On Investment* (ROI).

Após a execução do programa, é necessário acessar a subseção de *Finance Calculators*, assim será encontrado o método desejado em último lugar. Por fim, clica-se na opção que se deseja e tem-se a tela de apresentação do cálculo do ROI, mostrada na Figura 7.

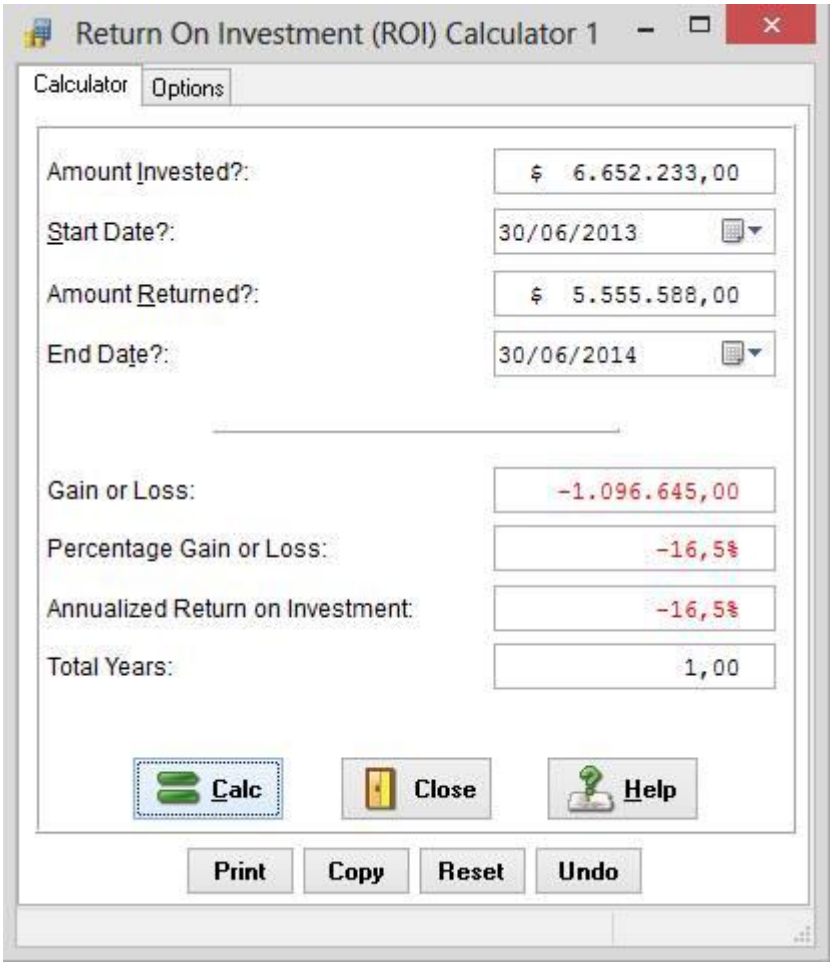

Figura 7 - SolveIt: Cálculo do ROI.
Fonte: SolveIt (2009).

Para que seja calculado o ROIC, são inseridos os valores de *Amount Invested* (Quantia Investida), *Amount Returned* (Quantia Retornada) e por fim os valores das datas de início (*Start Date*) e fim (*End Date*).

Conforme observado, os aplicativos gratuitos implementam um método de formação de preço de venda. Desta forma, o Grupo de Pesquisa em Sistema de Informação – Câmpus Ponta Grossa, linha Engenharia de Software, está desenvolvendo um *framework* de domínio para formação de preço de venda, denominado de FrameMK (*Framework* para Formação de Preço de Venda), descrito a seguir.

2.3 MODELO DO FRAMEMK

O FrameMK concentra vários métodos de preço de venda para produtos (Sebrae, Pleno e ABC) e tem por finalidade oferecer ao usuário um ambiente no qual pode comparar o preço gerado para um produto ou serviço em vários métodos.

2.3.1 Equipe de Desenvolvimento

O FrameMK é um projeto que está em desenvolvimento e envolve estudantes de graduação, especialização e mestrado. A seguir descreve-se um breve histórico dos trabalhos já desenvolvidos, bem como seus respectivos autores.

Anderson Crazuski, Leandro Botelho Feitosa e Thiago Luiz Cordeiro iniciaram o estudo do domínio em 2008 com o trabalho “Identificação dos pontos de estabilidade e de flexibilidade dos métodos para o estabelecimento de preço de venda” (CRAZUSKI et al., 2008). O trabalho se baseia em identificar os pontos de estabilidade e de flexibilidade para criar a modelagem de três métodos, Método de Custeio em Atividades (ABC), Método Sebrae e Método Custo Pleno, estes que auxiliam gestores para estabelecer preço de venda. Para a execução desta tarefa foi usado o processo dirigido por responsabilidades.

Em 2009 os alunos: Rafael Rudnik de Oliveira e Rudy José Crissi Crema continuaram a pesquisa do domínio desenvolvendo o trabalho “Definição dos pontos de estabilidade e de flexibilidade, em nível de requisitos, no domínio de preço de venda” (RUDNIK; CREMA, 2009). O trabalho teve como finalidade apresentar os conceitos sobre a formação de preço de venda e do lucro, modelando dois métodos de formação de preço de venda, sendo eles o método do Custeamento Marginal e o do Retorno Sobre o Capital (ROIC).

Em 2010 os alunos Paulo Eduardo Boeira Capeller e Vinícius Camargo Andrade iniciaram o desenvolvimento da arquitetura inicial do FrameMK com a pesquisa intitulada de “Uso do processo dirigido a responsabilidade no desenvolvimento da arquitetura e modelagem do *framework* de preço de venda” (CAPELLER; ANDRADE, 2010). O trabalho desenvolvido teve como finalidade propor uma arquitetura inicial para o FrameMK. Conforme Capeller e Andrade (2010), o *framework* proposto utiliza 3 métodos de domínio, Custo Pleno, Custeio Baseado em Atividade e SEBRAE, os quais já tinham sidos modelados individualmente pelo trabalho dos alunos Anderson et al. (2008). A modelagem do *framework* desenvolvida, com base no processo voltado a responsabilidades possibilitou identificar e modelar os pontos de estabilidade e flexibilidade entre os três métodos.

Ainda em 2010, voltado para *Web*, desenvolveu-se o trabalho, “Um *web service* para busca de preço de produto”, realizado pelo aluno Claudinei Rodrigues Junior. A ideia foi criar um serviço de código aberto para a busca de preços de venda de produtos em sítios de e-commerce e desta forma auxiliar o grupo de pesquisa para a implementação do método baseado nas decisões das empresas concorrentes (JUNIOR, 2010).

Em 2011, tendo como base o trabalho de Capeller e Andrade (2010), foi realizado a “Refatoração da camada de apresentação do *framework* de preço de venda (Framemk)” por Renato Ramos (RAMOS, 2011). Para a realização desse processo de refatoração foram utilizados os *frameworks Struts* e o *Tiles.* O *Struts* permitiu a integração da camada de apresentação com as camadas de regra de negócio e persistência. O *Tiles* ajudou a dividir de forma mais adequada os pontos comuns e específicos na camada de apresentação.

No ano de 2012 o acadêmico Leandro Siqueira da Silva desenvolveu um módulo de login, utilizado no FrameMK, baseado em aspectos para gerenciamento de acesso com a pesquisa “Um método para identificação de aspectos em nível de análise baseado em atributos de requisitos não-funcionais” (SIQUEIRA, 2012). O método proposto identifica aspectos na fase de análise usando uma matriz adjacência. A vantagem que o método proposto oferece, é que os analistas não

necessitam ter conhecimentos sobre aspectos, pois esses serão identificados a partir das similaridades com requisitos não funcionais.

No mesmo ano, o acadêmico Victor Schnepper Lacerda desenvolveu o trabalho de refatoração do sítio ArcaboMK, denominado “Refatoração do aplicativo gerenciador de menus dinâmicos do sítio Arcabomk”. O sítio é destinado a armazenar as informações dos trabalhos realizados pelo grupo de pesquisa e de acadêmicos envolvidos nos projetos (LACERDA, 2012). Para a refatoração do sítio, foram estudadas algumas metodologias e técnicas de refatoração, as quais foram adaptadas para a criação das atividades usadas no processo de refatoração do Manipulador ArcaboMK, e assim, gerando uma estrutura que facilita a sua manutenibilidade. A refatoração foi realizada para *framework Struts* e sua validação foi obtida por meio do uso de métricas de qualidade e desempenho.

Em 2013, foi desenvolvido em *web service* para o FrameMK por Mazur (2013).

Conforme relatado o FrameMK implementa atualmente somente três métodos de formação de preço de venda. Desta forma, necessita-se adicionar mais métodos de precificação para torná-lo completo.

3 PROCESSO DE EXTENSÃO DO FRAMEMK

O presente capítulo apresenta o processo usado para a evolução do FrameMK proposto por este trabalho. A Seção 3.1 faz menção a alguns processos de extensão de *frameworks* existentes na literatura. A Seção 3.2 descreve a adaptação de um destes processos. Por fim, a Seção 3.3 aplica o processo proposto para inserção de novos requisitos ao FrameMK.

3.1 PROCESSOS DE EXTENSÃO DE *FRAMEWORKS* EXISTENTES NA LITERATURA

Existem na literatura alguns processos que se preocupam com a evolução de *frameworks*. Dentre elas se destaca a de Rajlich (1997), que foca sobre erros ou inconsistências encontradas na estrutura do *framework*, de Saito e Yakamoto (2002), que se preocupam como a regra de negócio foi implementada, e de Camargo e Masiero (2005), que propõe em identificar como novos requisitos podem ser alocados ao sistema já em produção. Este tipo de abordagem será utilizada neste trabalho justamente por se preocupar em como novas funções podem ser inseridas ao *framework*. Os autores propõem quatro etapas, sendo elas:

- Identificar e documentar novo requisito.
- Analisar a documentação da biblioteca de FTs (*Frameworks* Transversais) e identificar um ou mais *frameworks* candidatos que talvez possam ser utilizados para instanciar o novo requisito.
- Analisar documentação do FT recuperado.
- Implementar novo requisito.

A Figura 8 apresenta um esquema de ligação entre estas etapas para que o processo de extensão seja melhor compreendido.

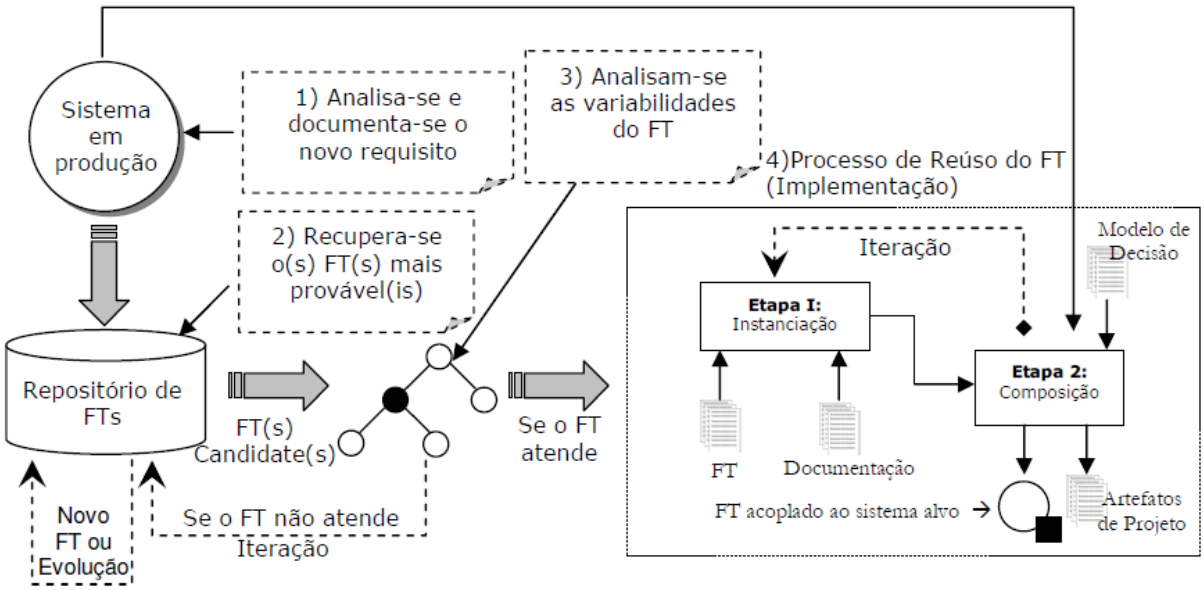

Figura 8 - Esquema de Extensão para Frameworks Transversais.
Fonte: Camargo e Masiero (2005).

A primeira etapa pode ser feita através de metodologias de análise/projetos, ou até mesmo técnicas orientados a aspectos.

Na segunda etapa se buscam outros *frameworks* cujos recursos sejam úteis a nova funcionalidade desejada, sendo assim, instancia-se este novo *framework* em meio a aplicação.

A análise da documentação do *framework* que será instanciado, averiguando se realmente as funcionalidades desejadas são apresentadas é realizada na terceira etapa. Caso não seja encontrado um FT que se adeque as características desejadas, então ou pode ser desenvolvido um novo *framework*, ou estendido um já existente para que seja adequado ou até mesmo implementar o novo requisito de forma *ad-hoc.*

Na quarta e última etapa o novo requisito é implementado (CAMARGO; MASIERO, 2005).

3.2 PROCESSO PROPOSTO PARA A EXTENSÃO DE *FRAMEWORK*

A partir da metodologia de Camargo e Masiero (2005), uma adaptação foi desenvolvida visando atender a evolução do FrameMK, como pode ser observado

na figura 9. Alguns passos foram retirados do processo original, como por exemplo, “Analisar a documentação da biblioteca de FTs e identificar um ou mais *frameworks* candidatos que possam ser utilizados para instanciar o novo requisito”. Esta remoção foi feita, pois não foram encontrados até o momento *frameworks* específicos a implementação de precificação de produtos. A metodologia de Camargo e Masieiro (2005) é voltada para *frameworks* transversais, neste caso, a metodologia não se encaixa totalmente para o contexto do trabalho, pois o FrameMK é um *framework* de domínio.

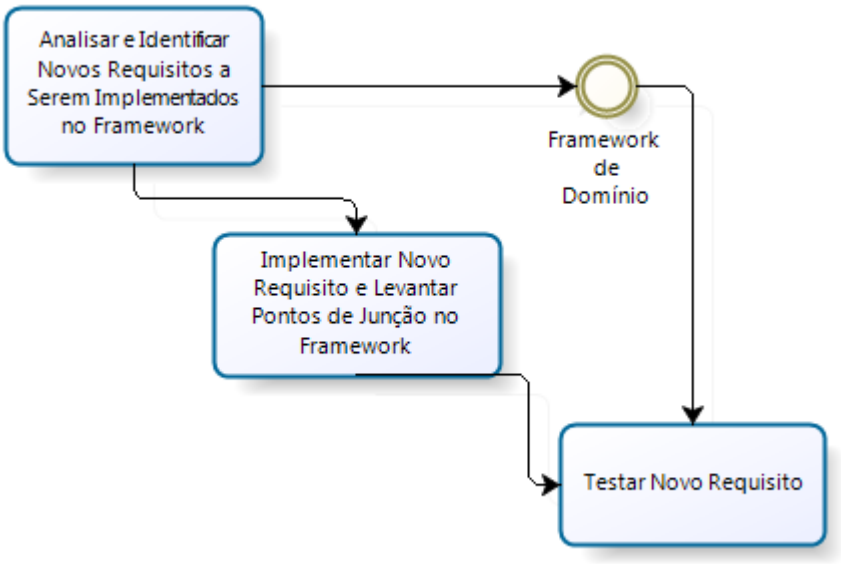

Figura 9 – Processo Proposto para Extensão de *Frameworks* de Domínio.
Fonte: Autoria própria.

Tomando por base a Figura 9, têm-se as descrições de cada uma das etapas:

- A Primeira etapa é apresentada como “Analisar e Identificar Novos Requisitos a Serem Implementados no *Framework*” e consiste em fazer um levantamento de todos os requisitos necessários para a implementação de novas funcionalidades ao *framework* em questão.

- Conforme a etapa de “Implementar Novo Requisito e Levantar Pontos de Junção no *Framework*”, uma implementação dos novos requisitos

a serem adicionados ao FrameMK é feita a partir da análise e identificação de seus requisitos.

- A etapa de “Testar Novo Requisito” permite realizar testes para verificar se os requisitos foram implementados corretamente. Existem vários métodos para testes de software, dentre eles, *White-box Testing* (Testes de Caixa-branca), *Incremental Testing* (Testes Incrementais), *Extreme Unit Testing* (Testes Unitários Extremos), etc (MYERS, 2004). Cada um dos testes foca em pontos diferentes do sistema sejam estes processamentos importantes ou um módulo como um todo. Além dos métodos mencionados acima, há o teste de aceitação, que centraliza esforços para garantir o funcionamento correto de cada funcionalidade adicionada ao *software* objetivando atender as espectativas definidas pelo cliente (ASSIS, 2012). Este trabalho usará o método de teste de aceitação em tempo de execução para validar os novos requisitos que serão adicionados.

3.3 APLICAÇÃO DO PROCESSO PROPOSTO PARA EXTENSÃO DO FRAMEMK

A partir da readaptação do processo de Camargo e Masiero (2005) serão realizados alguns passos para incorporar novos requisitos ao FrameMK. Os passos consistem em analisar e documentar um novo requisito descrito na seção 3.3.1, desenvolver o novo requisito e levantar os pontos que possibilitem a sua junção ao FrameMK relatados na seção 3.3.2, e por fim, a seção 3.3.3 apresenta a implantação do novo requisito.

3.3.1 Identificação e documentação de novos requisitos

Os novos requisitos a serem inseridos no *framework* estão relacionados ao método ABC, Marginal e ROIC. O ABC foi modelado e implementado por Crazuski et. al (2008). Posteriormente, o método foi reanalisado unindo os pontos semelhantes dos métodos estudados formando subsistemas, em seguida, foi

modelado e codificado por Capeller e Andrade (2010) no FrameMK. Já os métodos, Marginal e ROIC, foram modelados e codificados por Oliveira e Crema (2009).

Os três métodos foram analisados neste trabalho possibilitando identificar os requisitos a serem inseridos no FrameMK.

O primeiro método estudado foi o ABC que contém em sua arquitetura os seguintes subsistemas: *Attributes, FoodSystem, Calculation, ProductLine*, *Product* e *Activity,* porém os três últimos não haviam sido incorporados ao *framework* de preço de venda.

Já o segundo e o terceiro método avaliados foram o Marginal e ROIC. Os subsistemas do método Marginal são: Cálculo do Preço de Venda, Conexão e Cadastro. O ROIC apresenta os seguintes subsistemas: Cálculo do Preço de Venda, Gerenciar Produto, Gerenciar Custo, Gerenciar Matéria-Prima e Conexão. Nenhum dos subsistemas haviam sido implementados no FrameMK.

Descrevem-se as funcionalidades de cada um dos métodos, as quais foram obtidas por meio de um estudo dos trabalhos já desenvolvidos.

Método ABC

Ao analisar o método ABC foram identificados sete (7) subsistemas em sua versão atual, ilustrados no Quadro 1.

Subsistemas do Método ABC Crazuski et al. (2008)	**Subsistemas do Método ABC por Capeller e Andrade (2010)**
Cálculo de Preço de Venda	*Calculation*
Gerenciar Linha de Produção	*ProductionLine*
Gerenciar Produto	*Product*
Gerenciar Atividade	*Activity*
Gerenciar Atributo	*Attributes*
Gerenciar dados do Atributo	*FoodSystem*
Gerenciar Dados da Atividade	
Conexão	Camada de Persistência

Quadro 1 - Subsistemas do Método ABC.
Fonte: Autoria Própria.

Após a análise e identificação dos subsistemas, elaborados por Crazuski et al., (2008), foi realizado por Capeller e Andrade (2010) o refinamento das informações de cada subsistema com o objetivo de entender melhor o seu funcionamento para o *framework*. Ao realizar tal refinamento, foram identificados

sete subsistemas dos quais foram realizados os diagramas de caso de uso para, *Calculation*, *Attributes* e *FoodSystem*. Já os casos de uso dos subsistemas *ProductionLine, Product* e *Activity* não faziam parte da estrutura do FrameMK.

O Quadro 2 ilustra as funcionalidades dos subsistemas encontrados.

Subsistema	**Funcionalidades**	**Implementado? (Sim/Não)**
Calculation	Acessar Tela de Calculo	Sim
	Informar Números Produzidos	Sim
	Escolher Linhas de Produção	Sim
	Buscar Produtos	Sim
	Calcular	Sim
	Inserir Margem de Lucro	Sim
	Salvar	Sim
	Tela de Cálculo ABC	Sim
	Tela de Cálculo Sebrae	Sim
	Tela de Cálculo Custo Pleno	Sim
	Buscar Linha de Produção	Sim
Attributes	Cadastrar Atributo	Sim
	Editar Atributo	Sim
	Selecionar Atributo	Sim
	Informar Dados	Sim
	Inserir Dado do Tipo Atributo	Sim
	Inserir Dado do Tipo Variável	Sim
	Verificar Validade dos Dados	Sim
	Armazenar Valores	Sim
	Desabilitar Atributo	Sim
	Buscar Atributo	Sim
	Filtrar Pesquisa	Sim
	Buscar Código	Sim
	Buscar Variável	Sim
	Buscar Descrição	Sim
FoodSystem	Buscar Atributos	Sim
	Buscar Dados do Método ABC	Sim
	Buscar Dados do Método Sebrae	Sim
	Buscar Dados do método Custo Pleno	Sim
	Editar Valores	Sim
	Verificar Validade dos Dados	Sim
	Salvar Alterações	Sim
	Efetuar Cálculo	Sim
ProductionLine	Consultar Linha	Não
	Cadastrar Linha	Não
	Editar Linha	Não
	Excluir Linha	Não
Product	Cadastrar Produto	Não
	Consultar Produto	Não
	Editar Produto	Não
	Excluir Produto	Não
Activity	Cadastrar Atividade	Não
	Editar Atividade	Não
	Consultar Atividade	Não
	Excluir Atividade	Não

Quadro 2 – Funcionalidades por subsistema do método ABC.
Fonte: Autoria Própria.

Método Marginal

Analisando o método descrito no trabalho de Oliveira e Crema (2009), composto pelos pacotes do diagrama de dependência de pacotes da UML 2.0, foi possível identificar os subsistemasQuadro 3 necessários para a implementação do método marginal, os mesmos estão ilustrados no Quadro 3.

Subsistemas do Método Marginal
Cálculo de Preço de Venda
Controle de Cadastros
Conexão

Quadro 3 - Subsistemas do Método Marginal.
Fonte: Própria.

O subsistema “Cálculo de Preço de Venda” destina-se a calcular o preço de venda, levando em consideração todos os valores referentes aos custos envolvidos do produto ou serviço.

O subsistema “Controle de Cadastros” é destinado ao cadastro de produtos e de custos e posteriormente são associados a produto. Além disto, é possível realizar consulta, desativação e a atualização dos dados relacionados aos custos da empresa. Também oferece suporte a busca e exibição de produtos por meio de uma interface de consulta, alteração e exclusão dos produtos cadastrados.

No Quadro 3, o subsistema de Cadastros é mostrado de forma geral. No Quadro 4, o subsistema é mostrado de forma mais detalhada, exibindo os dois controles de cadastro pertencentem a este subsistema.

O subsistema “Conexão” é responsável por estabelecer a conexão o sistema e o banco de dados.

O Quadro 4 apresenta as funcionalidades por subistemas, bem como se haviam sido codificadas no FrameMK.

Subsistema	Funcionalidades	Implementado? (Sim/Não)
Cálculo de Preço de Venda	Selecionar Produto	Não
	Informar dados	Não
	Calcular Custos	Não
	Exibir Resultado	Não
Controle de Cadastros (Custo e Produto)	Incluir novo custo	Não
	Desativar custos	Não
	Consultar custos	Não
	Alterar custos	Não
	Salvar	Não
	Exibir aba Pesquisada de custos	Não
	Cancelar	Não
	Carregar Dados	Não
	Voltar a aba Gerenciamento	Não
	Incluir Novo produto	Não
	Desativar produto	Não
	Consultar produto	Não
	Alterar produto	Não
	Remover custos	Não
	Cancelar	Não
	Salvar	Não
	Exibir aba Pesquisa Produto	Não
	Exibir aba Inclusão de custos produto	Não
	Carregar e voltar aba Gerenciamento	Não
	Voltar aba Gerenciamento	Não
	Buscar custos	Não
	Adicionar custos ao produto	Não
Conexão	Controle de Cadastro	Não
	Calcular Preço de Venda	Não
	Desconectar do Banco de Dados	Não

Quadro 4 – Funcionalidades por Subsistema do método Marginal.
Fonte: Autoria Própria.

Método ROIC

Seguindo a análise realizada pelo trabalho de Oliveira e Crema (2009), foram identificados os subsistemas para o método ROIC, descritos no Quadro 5.

Subsistemas do Método ROIC
Cálculo do Preço de Venda
Gerenciar Produto
Gerenciar Custo
Gerenciar Matéria-prima
Conexão

Quadro 5 - Elementos da Arquitetura do Método ROIC.
Fonte: Própria.

O subsistema de “Cálculo do Preço de Venda” é responsável por calcular o preço do produto ou do serviço levando em consideração os custos envolvidos.

O “Gerenciar Custo” tem a finalidade de realizar o cadastro, consulta, exclusão e a atualização dos dados relacionados aos custos da empresa.

Já o subsistema “Gerenciar Produto” é voltado para o cadastro, consulta, exclusão e atualização dos dados dos produtos para serem posteriormente realizados os cálculos de seus preços de venda.

O subsistema “Gerenciar Matéria-prima” tem o objetivo de realizar o cadastro, consulta, exclusão e a atualização dos dados sobre as matérias-primas para serem relacionadas posteriormente com os produtos.

Por fim, o subsistema “Conexão” possibilita uma conexão entre o sistema e o banco de dados. O Quadro 6, ilustra as funcionalidades para os subsistemas descritos anteriormente.

Subsistema	Funcionalidades	Implementado? (Sim/Não)
Cálculo do Preço de Venda	Inserir Novo Cálculo	Não
	Selecionar Produto	Não
	Consultar Cálculo	Não
	Alterar Cálculo	Não
	Excluir Cálculo	Não
	Salvar Cálculo	Não
	Informar Dados	Não
	Calcular Preço de Venda	Não
	Cancelar	Não
	Adicionar Custo	Não
	Remover Custo	Não
	Remover Todos Custos	Não
	Filtrar Consulta	Não
	Exibir Aba Consulta	Não
	Selecionar Custos	Não
	Exibir Tela Busca Custo	Não
	Filtrar Custo	Não
	Ver Detalhes	Não
	Voltar para Cadastro	Não

Quadro 6 – Funcionalidades por Subsistema do método ROIC.
Fonte: Autoria Própria.

Subsistema	Funcionalidades	Implementado? (Sim/Não)
Gerenciar Custo	Inserir novo Custo	Não
	Consultar custo	Não
	Alterar custo	Não
	Excluir custo	Não
	Salvar custo	Não
	Cancelar	Não
	Informar Dados	Não
	Exibir aba Consulta	Não
	Filtrar	Não
	Ver Detalhes	Não
	Voltar para Cadastro	Não
	Inserir Nova Matéria-prima	Não
	Consultar Matéria-prima	Não
	Alterar Matéria-prima	Não
	Excluir Matéria-prima	Não
	Salvar Matéria-prima	Não
	Cancelar	Não
	Informar Dados	Não
	Exibir aba Consulta	Não
	Filtrar	Não
	Ver Detalhes	Não
	Voltar para Consulta	Não
Produto	Inserir Nono Produto	Não
	Consultar Produto	Não
	Alterar Produto	Não
	Excluir Produto	Não
	Salvar Produto	Não
	Informar Dados	Não
	Cancelar	Não
	Adicionar Matéria-prima	Não
	Remover Matéria-prima	Não
	Filtrar Consulta	Não
	Exibir aba consulta	Não
	Exibir Tela Busca Matéria-prima	Não
	Remover Todas Matérias-primas	Não
	Selecionar Matéria-prima	Não
	Filtrar Matéria-prima	Não
	Ver Detalhes	Não
	Voltar para Cadastro	Não

Quadro 6 – Funcionalidades por Subsistema do método ROIC.
Fonte: Autoria Própria.

Após a identificação dos novos requisitos a serem inseridos no FrameMK, foi finalizado a primeira etapa do processo. Logo, a segunda etapa, descrita na seção 3.3.2, refere-se a identificar os pontos de junções que podem ser utilizados para a inserção dos novos métodos ao *framework*.

3.3.2 Implementar Novo Requisito e Levantar Pontos de Junção no *Framework*

Esta etapa encontra os pontos propícios para inserção de novos requisitos no *framework*. Conforme mencionado na etapa 1, o método ROIC e Marginal não tinham nenhum subsistema implementado e desta forma, optou-se por iniciar o processo de inclusão pelo subsistema que já tinha parte implementado, a saber, ABC.

Inserção dos Requisitos Pertencentes ao Método ABC

Nesta seção é relatada a criação e alteração das classes pertences aos subsistemas, *Product Line*, *Product* e *Activity*, do método ABC que foram inseridos ao FrameMK.

a) Subsistema ProductionLine

Este subsistema gerencia a linha de produção do método de cálculo ABC, o que possibilita ao usuário cadastrar, editar ou desabilitar linhas. É o único que não depende de nenhum outro subsistema, pois para cadastrar uma nova linha de produção é necessário apenas o nome desta.

Atualmente, o FrameMK está com a opção desabilitada destes subsistemas no sitio onde fica hospedado, como ilustrado na Figura 10. Após a inserção do novo requisito a opção estará disponível.

Figura 10 – Opções do Método ABC.
Fonte: FrameMK (2013).

A identificação de um ponto de junção foi realizada após análise do código fonte do FrameMK e sua documentação. Usou-se o modelo criado por Crazuski et al. (2008), ilustrado na Figura 11, com o objetivo de melhor compreender as suas funcionalidades.

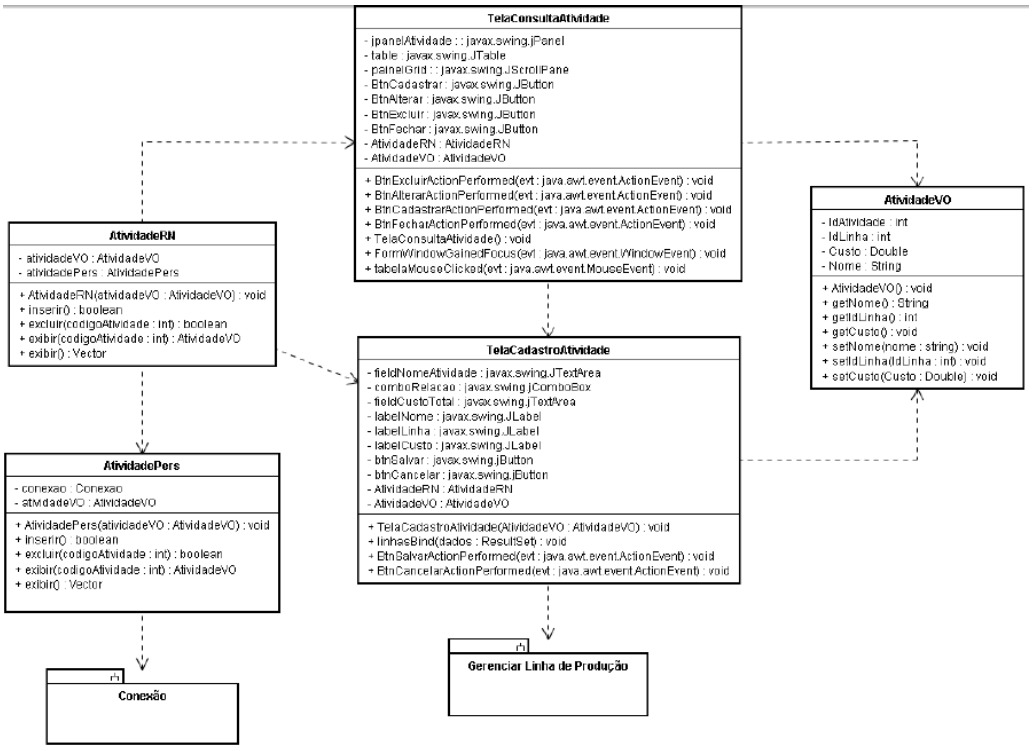

Figura 11 - Diagrama de classe Atividade e subsistema Gerenciar Linha de Produção.
Fonte: Crazuski et al. (2008, p. 18).

Outro ponto analisado foi o Banco de Dados (BD) do FrameMK. Com isso, foi possível entender a estrutura dos dados e como trabalhar com a arquitetura já estabelecida. Neste estudo, verificou-se que o BD já possuía a tabela Linha, necessária para a inclusão deste subsistema. Esta tabela têm duas dependências, a saber, tabela ATIVIDADE_LINHA e PRODUTO_ABC.

Posteriormente, foi realizado um estudo dos códigos e visto o que seria necessário para inserir o novo requisito. Primeiramente, foi adicionado um *link* para *ProductionLine* no JSP *windowsMenuAbc*, a Figura 12 ilustra esta alteração.

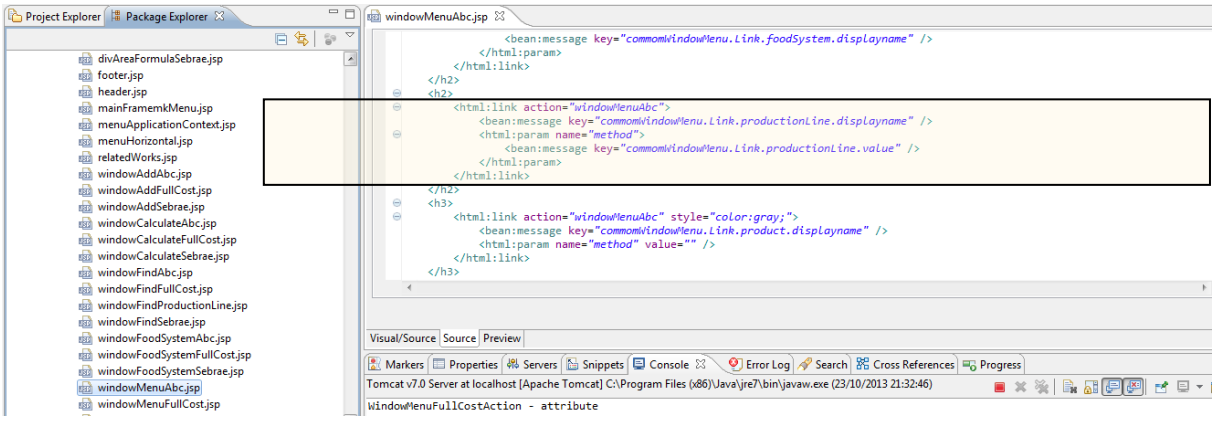

Figura 12 - Adição de *Link* para *ProductionLine* em JSP.
Fonte: Autoria Própria.

Em seguida, foi adicionado um método para redirecionamento de *ProductionLine* na classe *windowMenuAbcAction*, a Figura 13 ilustra a alteração realizada.

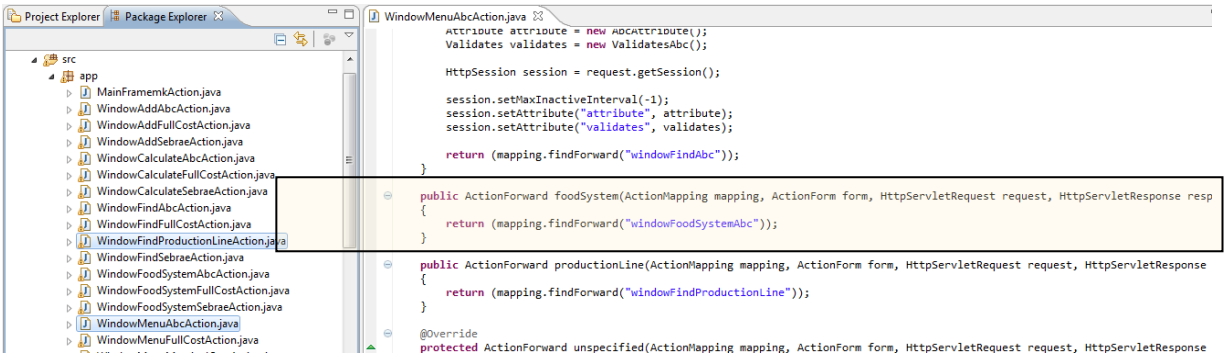

Figura 13 - Adição de método para redirecionamento de *ProductionLine* em *windowsMenuAbcAction.*
Fonte: Autoria Própria.

Outra tarefa realizada foi a inserção de *forward* para *action windowFindProductionLineAction* sem método especificado, ou seja, quando é retornado um *ActionForward* (observar retorno de método Figura 13) em uma classe que herda *Action,* o *framework struts* procura dentro do arquivo *struts.conf* o *forward* correspondente ao requisitado e chama o *path* relacionado a ele, sendo assim, também foi alterado o *struts-config.xml* para detecção deste novo *forward* e abertura de um novo arquivo, conforme exibe a Figura 14.

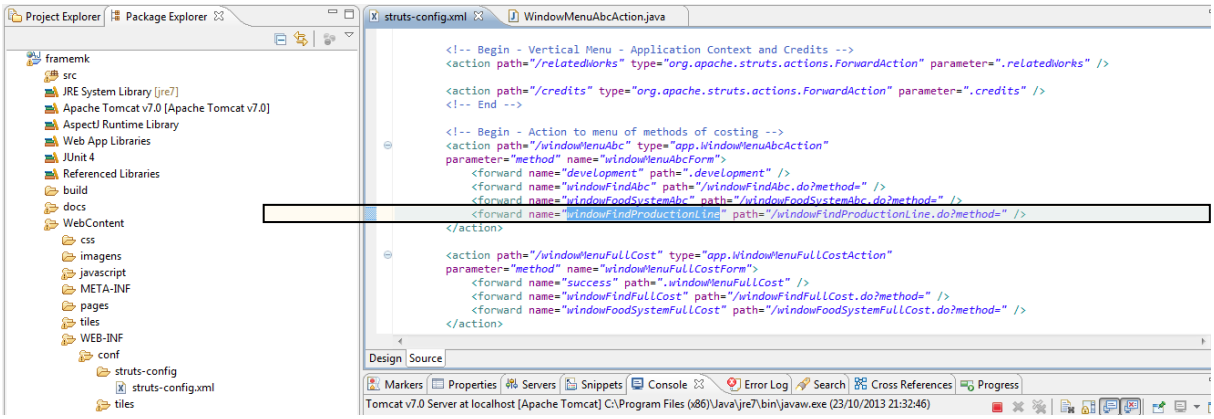

Figura 14 – Adição de *forward* para *windowFindProductionLineAction* sem método especificado.
Fonte: Autoria Própria.

Sabe-se que uma classe que herda *Action* será chamada em um *path* (seleção dentro da Figura 14), quando a extensão *.do* sucede o nome da classe requerida. Também, quando é enviado um método vazio (seleção dentro da Figura 14), a classe pai *Action* tem um método chamado *unespecified* que recebe este tipo de requisição, desta forma, foi criado um método na *WindowFindProductionLineAction* para receber as requisições de método sem especificação, conforme mostra a Figura 15.

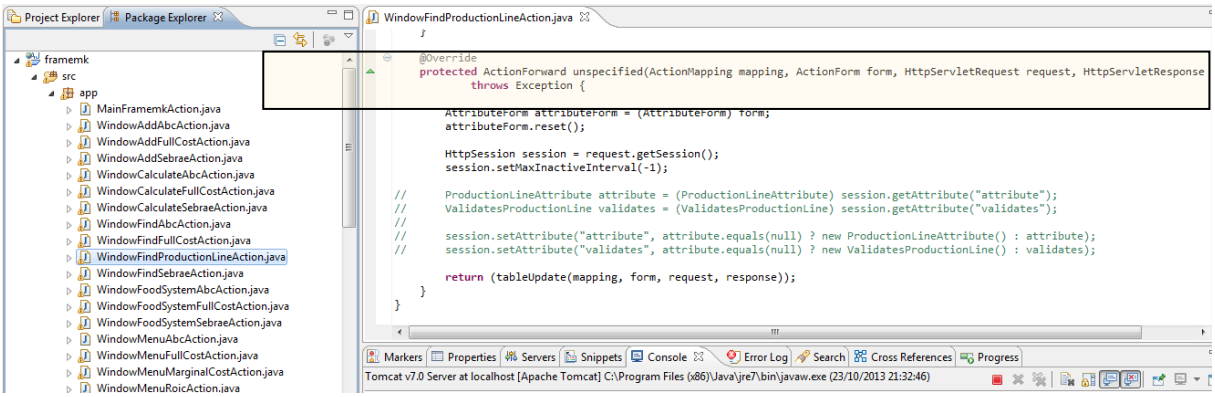

Figura 15 – Criação de método para receber requisições sem especificação de método em *Production Line.*
Fonte: Autoria Própria

A Figura 16 ilustra a busca de linhas de produção cadastradas no banco de dados, retornando uma chamada de *forward success*.

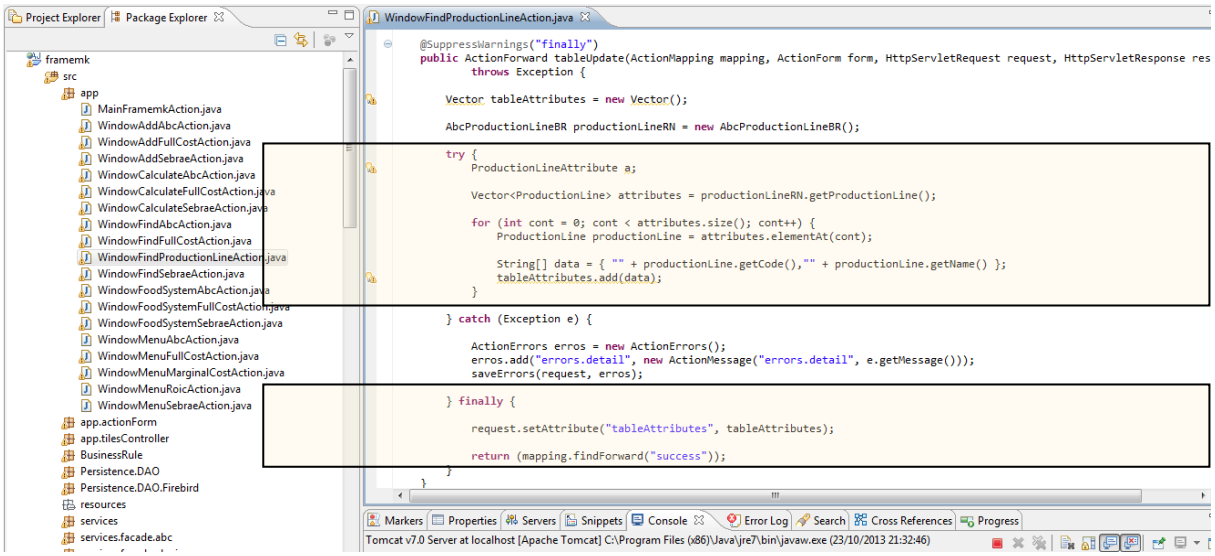

Figura 16 - Busca de atributos de *ProductionLine* e chamada para *success.*
Fonte: Autoria Própria.

A Figura 17 ilustra a inserção da *tag action* para *windowFindProductionLineAction* e a criação de *forward* para *success*.

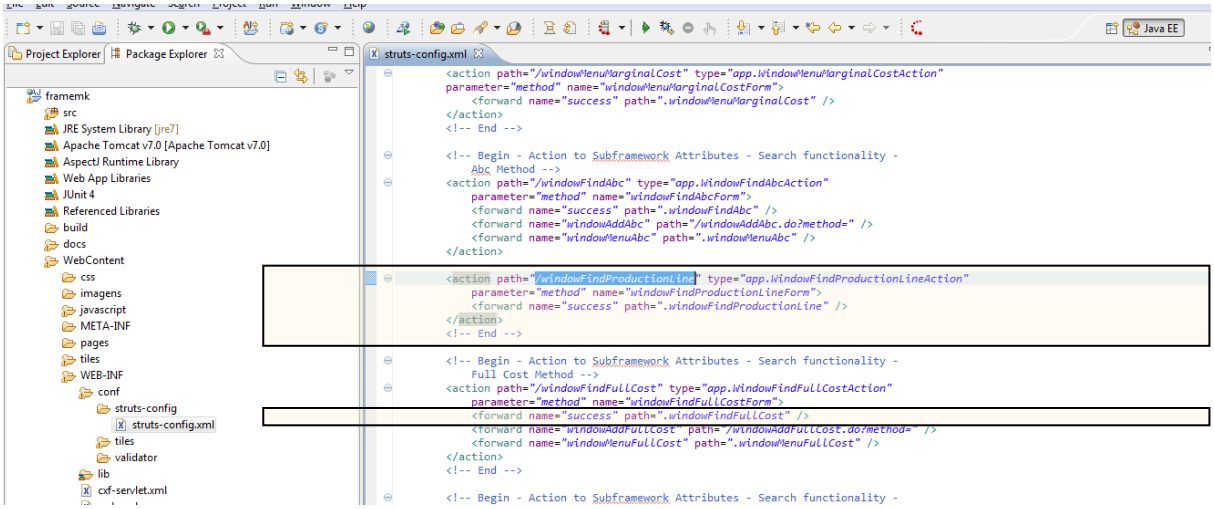

Figura 17 - Inserção da *tag action* para *windowFindProductionLineAction* e criação de *foward* para *success.*
Fonte: Autoria Própria.

A tarefa posterior foi inserir a *tag definition* para .*windowFindProductionLine* para extensão de interface. Esta *tag* é chamada através do *Path* da Figura 17. A Figura 18 exibe a *tag definition*.

Figura 18 - Adição de *tag definition* para *.windowFindProductionLine* para extensão de interface.
Fonte: Autoria Própria.

A Figura 19 ilustra a criação de JSP *windowFindProductionLine* definindo quais as colunas serão apresentadas na listagem de Linhas de Produção e também quais métodos serão disparados em *windowFindProductionLineAction* quando as opções de inserção, deleção, busca e fechamento da funcionalidade forem requisitadas.

Figura 19 - Criação de JSP *windowFindProductionLine*.
Fonte: Autoria Própria.

A Figura 20 ilustra as classes que já existiam, as que foram modificadas e as inseridas no FrameMK para a inserção das funcionalidades dos subsistemas *Product Line*, *Product* e *Activity* do método ABC. As classes na cor amarela representam as já existentes, as em azul-escuras são as que foram modificadas e as azul-claras são as que foram adicionadas.

Figura 20 - Diagrama de classe do FrameMK para inserção do subsistema *Product Line* do método ABC.
Fonte: Autoria Própria.

Contudo, para a inserção do subsistema *Product Line* foram alteradas nove (9) classes, criadas mais cinco (5) e reusadas seis (6) classes já existentes no projeto para atender as necessidades de execução do subsistema.

b) Subsistema *Product*

O subsistema *Product* é responsável por exibir todos os produtos cadastrados, podendo o usuário efetuar as funções de cadastrar, editar ou desabilitar um novo produto. A inserção deste subsistema está documentada no Apêndice A, pois seu processo é similar ao que foi descrito para o subsistema *Product Line*.

A Figura 21, ilustra as classes que já existiam, as que foram modificadas e as inseridas no FrameMK para o subsistema *Product*. As classes na cor amarela representam as já existentes, as em azul-escuras são as que foram modificadas e as azul-claras são as que foram adicionadas.

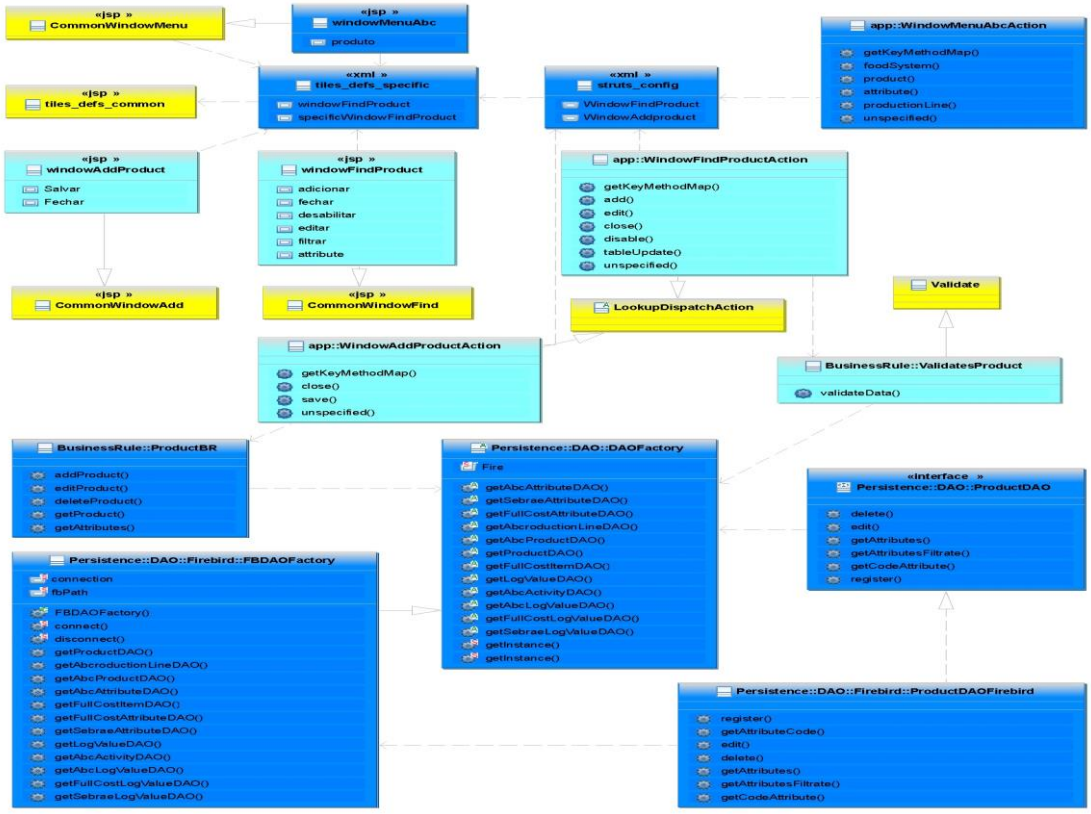

Figura 21 - Diagrama de classe do FrameMK para inserção do subsistema *Product* do método ABC
Fonte: Autoria Própria.

Desta forma, foram alteradas nove (9) classes, criadas mais cinco (5) e reusadas seis (6) classes sem qualquer tipo de alteração para atender as necessidades deste subsistema.

c) Subsistema *Activity*

O subsistema *Activity* tem a função principal de exibir todas as atividades existentes, podendo o usuário efetuar o cadastro, editar ou desabilitar uma atividade. A seguir é descrito algumas características deste subsistema. A descrição do processo de inserção se encontra em sua totalidade no Apêndice B.

Além dos procedimentos usados para a inserção do *Product Line*, o subsistema *Activity* necessitou a criação da classe *windowFindActivity.jsp* para atender ao menu Atividade, ilustrado na Figura 22.

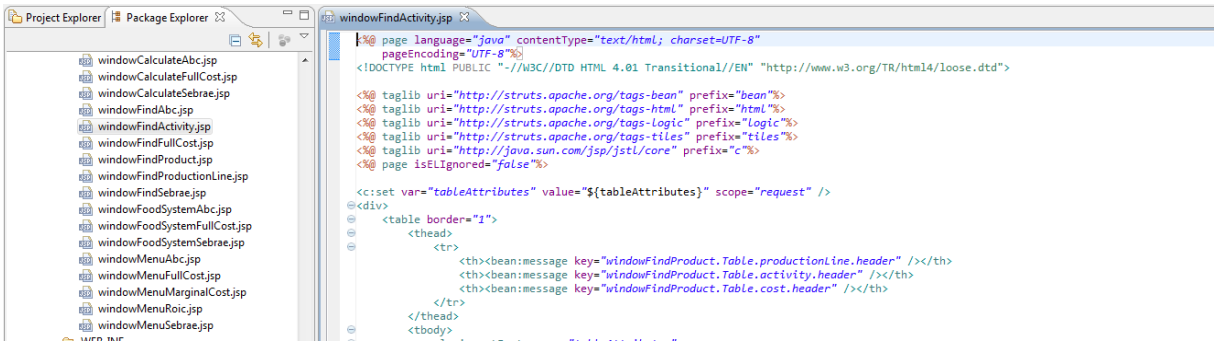

Figura 22 - Criação da classe *windowFindActivity*.jsp
Fonte: Autoria Própria.

Posteriormente, foi criado a classe *WindowAddActivity*.jsp para as opções de adição e/ou edição de produtos. Figura 23 ilustra essa tarefa e ressalta-se que o *jsp* possui poucas linhas de código, pois as únicas *tags* necessárias para a adição de produto são as mesmas do *jsp* base *WindowCommonAdd.jsp.*

Figura 23 - Criação da classe *WindowAddActivity*.jsp para adição e/ou edição de Atividade.
Fonte: Autoria Própria.

Em seguida, é ilustrado através da Figura 24, a busca de linhas de produção que serão listadas em *windowAddActivity.jsp.* A classe *WindowAddActivityController* é quem controla essa tarefa.

Figura 24 – Controle de busca de linhas de produção por *WindowAddActivityController* no subsistema *Activity.*
Fonte: Autoria Própria.

A Figura 25 ilustra as classes que já existiam, as que foram modificadas e as inseridas no FrameMK para o subsistema *Activity*. As classes na cor amarela representam as já existentes, as em azul-escuras são as que foram modificadas e as azul-claras são as alteradas.

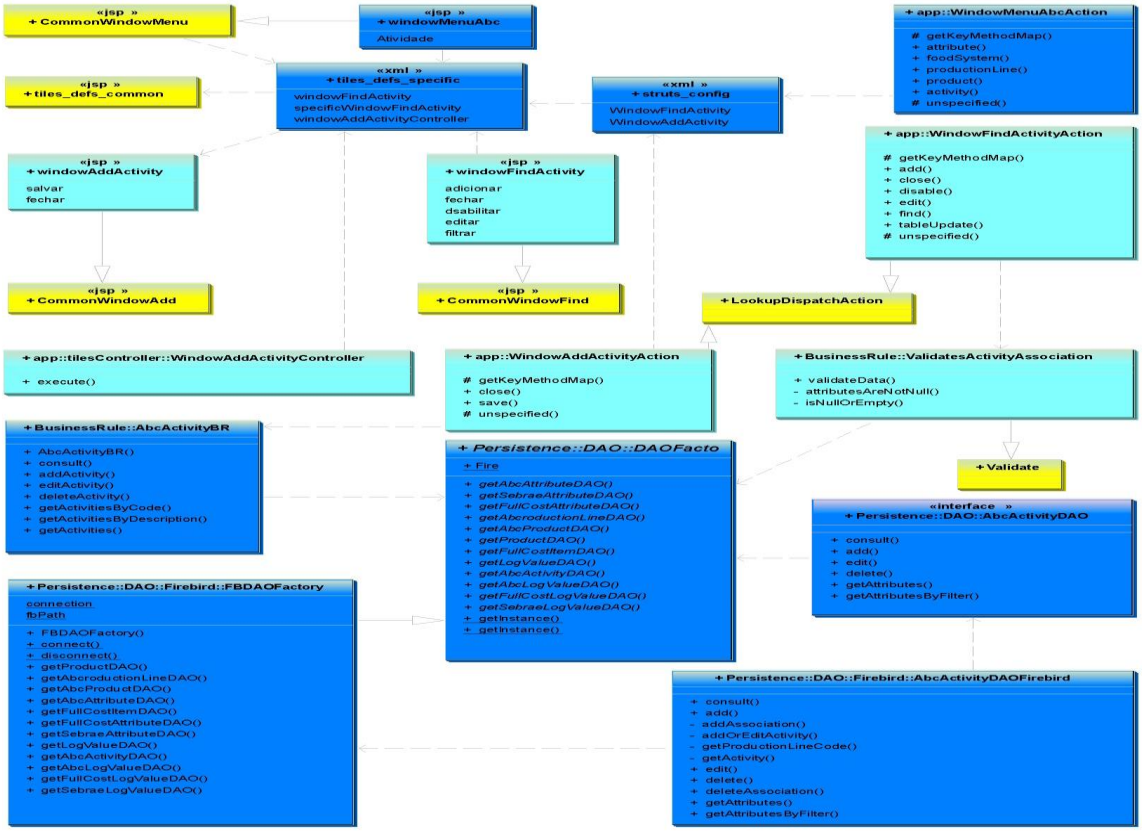

Figura 25 - Diagrama de classe do FrameMK para inserção do subsistema *Activity* do método ABC.
Fonte: Autoria Própria.

Portanto, foram alteradas nove (9) classes, criadas mais seis (6) e reusadas seis (6) para atender as necessidades de execução do subsistema.

Inserção dos Requisitos Pertencentes ao Método Marginal

Conforme relatado na primeira etapa da metodologia, o método Marginal, modelado e implementado por Oliveira e Crema (2009) ainda não havia sido implementado no FrameMK.

Após o estudo de seus subsistemas, notou-se que o módulo de Controle de Cadastros poderia ser subdividido em: Gerenciamento de Custos e Controle de Produto para melhor separar as funcionalidades. O restante dos subsistemas permaneceram inalterados, a saber, Cálculo de Preço de Venda e Conexão.

O processo de inclusão deste método foi realizado inicialmente adicionando um menu para seu acesso. A Figura 26 ilustra o diagrama de classe que mostra as classes necessárias para sua criação.

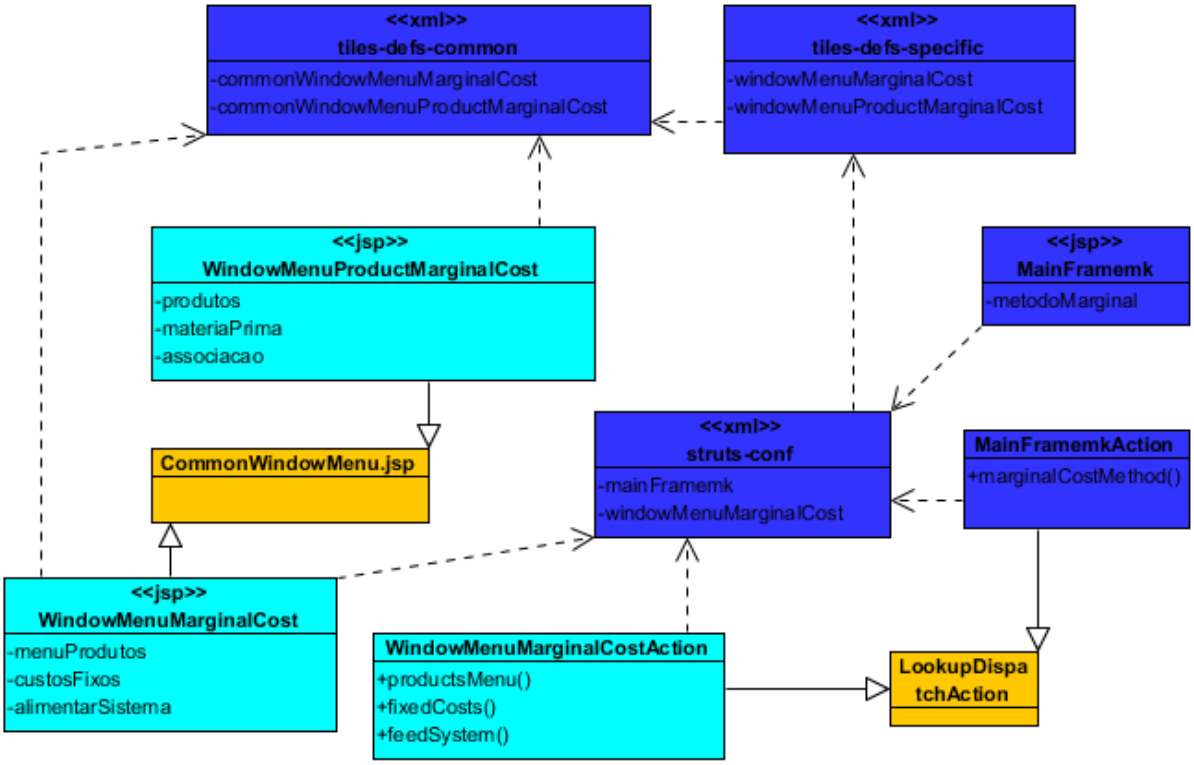

Figura 26 - Diagrama de Classe do FrameMK para Adição de Menus do Método Marginal.
Fonte: Autoria Própria.

Esta figura também apresenta o submenu para o método Marginal, pois ambos são implementados a partir do mesmo conceito. Foram criados JSP's que em suas *Tag's form* tem um atributo *action*, quando é feita uma requisição no JSP em questão, é verificado no arquivo XML *struts-config* se existe um atributo *path* de uma *Tag action* que contém o mesmo valor enviado pelo atributo *action* da requisição do JSP.

A *Tag action* se utiliza dos atributos *path* para o *action* especificado no JSP como explicado anteriormente, *type* para definição do caminho da classe *action*, *parameter, no caso do Framemk,* utilizado para definir o método a ser chamado dentro da classe *Action* especifiacada. Dentro da *Tag Action* são definidos *forward*s que recebem o resultado do método chamado na classe *Action*.

Uma vez que um *forward* é requisitado, é disparado o conteúdo do atributo *path* dentro dele, este pode ser uma chamada a um JSP ou a uma classe *Action*. É possível tomar como exemplo as primeiras implementações de métodos do FrameMK, quando é definido no atributo *path* do *forward* o nome de uma classe *Action*, mas sem sua terminação “Action” e sim “.do”. Observa-se então que a classe é disparada, mas em outros, simplesmente é definido um JSP e este é disparado.

Os arquivos XML *tiles-defs-common* e *tiles-defs-specific*, são utilizados pelo *framework* Tiles para organizar as interfaces com o usuário. O *tiles-defs-common* faz menção aos JSP’s base do sistema para a criação de novos menus, novos módulos para adição e edição de atributos, entre outros. Por sua vez, o *tiles-defs-specific* contém todos os JSP’s que herdam funcionalidades dos JSP’s base.

Sendo assim, para os menus do método custeio Marginal, tem-se:

1) Primeiramente é adicionado um *link* para o método custo Marginal no JSP que centraliza a chamada a todos os outros métodos de custeio (*MainFramemk*.jsp).
2) Em segundo lugar é adicionado um método *marginalCostMethod* na classe *Action* (*MainFramemkAction*) vinculada a este JSP. Este método processa a requisição enviada pelo JSP e retorna uma requisição a um *forward* relacionado a *Tag action* (dentro do XML *struts-config*) vinculada aquela classe *Action.*
3) Em terceiro lugar, o *forward* requisitado dispara uma chamada a um JSP que está no arquivo *tiles-defs-specific* na *Tag definition* com atributo *name* de valor *.windowMenuMarginalCost*. Já que é um menu, é necessário que seja adicionada uma *Tag definition* também no arquivo *tiles-defs-common*, e para este o *name* da *Tag definition* será *.commonWindowMenuMarginalCost*.

O mesmo processo de adição descrito anteriormente foi usado criação de seus submenus. As próximas subseções relatam como os requisitos dos subsistemas pertencentes a este método foram inseridos ao FrameMK.

a) Subsistema de Controle de Produto – Gerênciamento de Produto

Este subsistema é responsável pelo gerenciamento de produtos, matéria-prima e associação entre produto e matéria-prima. A Figura 27, 28 e 29, apresentam a modelagem proposta para o subsistema controle de produto.

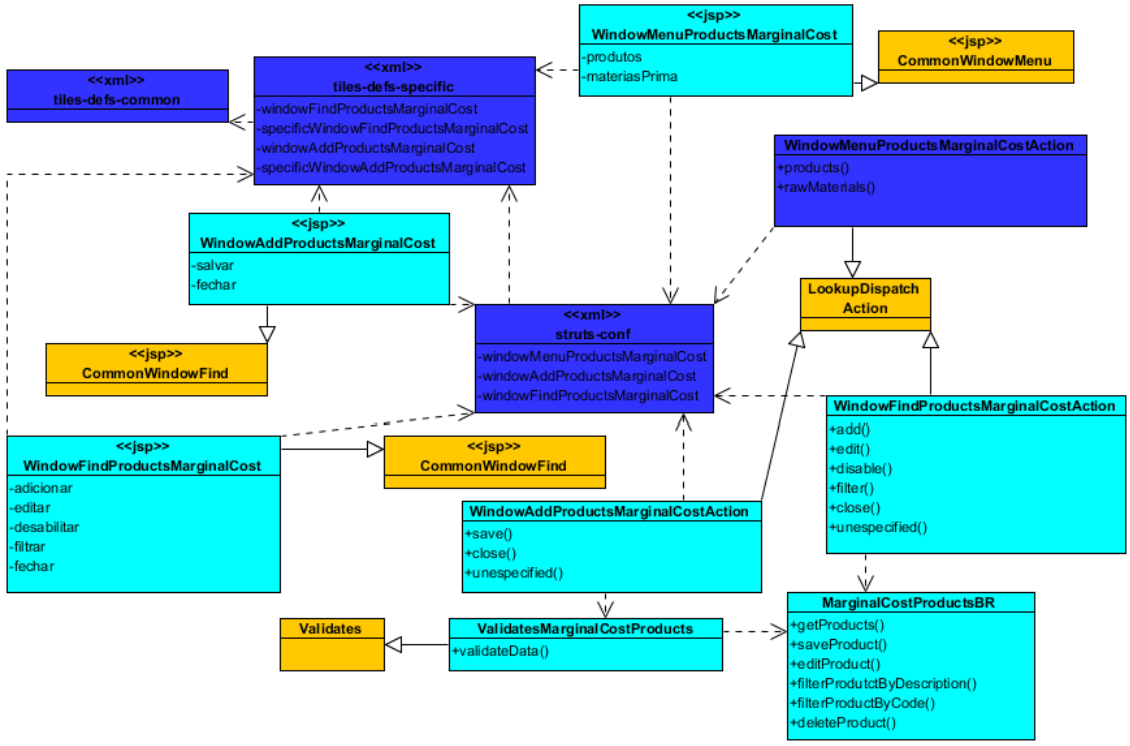

Figura 27 – Diagrama de classe do FrameMK para inserção do subsistema *Controle de Produto* do método Marginal.
Fonte: Autoria Própria.

A partir do menu de produtos criado pela Figura 26, é adicionado um *link* para produtos no JSP *WindowMenuProductsMarginalCost*, sendo assim, criou-se um método *products* na classe *WindowMenuProductsMarginalCostAction* que é chamado após o *click* no novo *link* do JSP. Quando este método é chamado, ele busca todos os produtos cadastrados no sistema através da classe de regra de negócio *MarginalCostProductsBR*, adiciona estes produtos na requisição e dispara um *forward* que é definido no *struts-config* para a *Tag action* de *WindowMenuProductsMarginalCostAction*. O *forward* disparado chama outras *Tags definition* dentro do arquivo *tiles-defs-specific*, sendo elas: *WindowFindProductsMarginalCost* e *SpecificWindowFindProductsMarginalCost*, as

quais fazem com que o JSP *WindowFindProductsMarginalCost* seja disparado, apresentando os produtos buscados anteriormente.

Após todos os produtos estarem exibidos para o usuário no JSP *WindowFindProductsMarginalCost*, são disponibilizadas as opções de adição, edição, deleção, filtro e fechar a página.

Para as opções de filtro e deleção, foi feita uma chamada a classe *WindowFindProductsMarginalCostAction* é realizada por meio dos métodos *filter* e *delete*, respectivamente. Ambos fazem uma chamada aos métodos *filterProductByDescription* ou deleteProduct da classe *MarginalCostProductsBR*, e posteriormente, retornam os dados para a tela.

O fechamento da tela é realizado pela classe *WindowFindProductsMarginalCostAction* chamando o método *close()* e este fará uma requisição ao *forward* para voltar ao menu método Marginal.

Por fim, nas opções de adição e edição realiza-se uma chamada ao método *add* ou *edit* da classe *WindowFindProductsMarginalCostAction*, e diferentemente das outras opções, estes métodos são redirecionadas para outro JSP de adição e deleção. Caso seja edição, os atributos do produto ficam na sessão e serão repassados para a próxima tela. Após receber a requisição o *Action* retornará um *forward* e o *struts-config* buscará a *Tag action* vinculada a classe *WindowFindProductsMarginalCostAction* e encontra o *forward* solicitado. Logo após, dispara o *path* descrito, sendo ele o *WindowAddProductMarginalCost*, e chama o *SpecifcWindowAddProductMarginalCost* que irá requisitar o JSP *WindowAddProductsMarginalCost*.

Para receber as requisições do JSP *WindowAddProductsMarginalCost*, foi criada uma *Tag action* no arquivo *struts-config* e os *forwars* relacionados ao mesmo. A classe *WindowAddProductsMarginalCostAction* têm os métodos *unespecified*, *save* e *close*. Quando a tela de adição/deleção é chamada pela primeira vez, é feito um *forward* na *Tag action* de *WindowFindProductsMarginalCost*, este por sua vez faz uma chamada direta a classe *action WindowAddProductsMarginalCostAction*. Caso não seja informado o método a ser chamado, a classe base *Action* já tem um método a ser sobrescrito para este tipo de chamadas, ou seja, o método *unespecified*.

Uma vez que o método *unespecified* é chamado, será buscado na sesão um produto associado a ela, caso exista, os atributos do produto pré-existente serão

colocados no *Form* da requisição e será chamado o *forward*. O *forward* por sua vez irá disparar o *path* descrito, *WindowAddProductMarginalCost* como foi definido anteriormente.

Após o clique do usuário no botão de fechar na tela de adição/edição de produtos, o método *close* da classe *WindowAddProductsMarginalCost* será chamado, retornando com um *forward* que redireciona novamente para classe de busca de produtos no sistema.

Para a opção de salvar, é buscado no *Form* procedente da requisição, os atributos do produto, estes atributos são repassados para a classe *ValidatesMarginalCostProducts*. Esta classe tem a finalidade de chamar a regra de negócio vinculada ao produto (*MarginalCostProductsBR*), sendo que a operação pode ser de adição ou deleção e retorna uma mensagem que será exibida ao usuário.

b) Subsistema de Controle de Produto - Gerenciamento de Matéria-Prima

O Gerenciamento de Matéria-Prima se assemelha ao de Produto, pois ambos os gerenciamentos não dependem de outras inserções no sistema. Este subsistema permite inserir, editar, deletar e filtrar matérias-primas no sistema. A Figura 28 ilustra o diagrama de classe resultante do processo de inserção deste subsistema ao FrameMK.

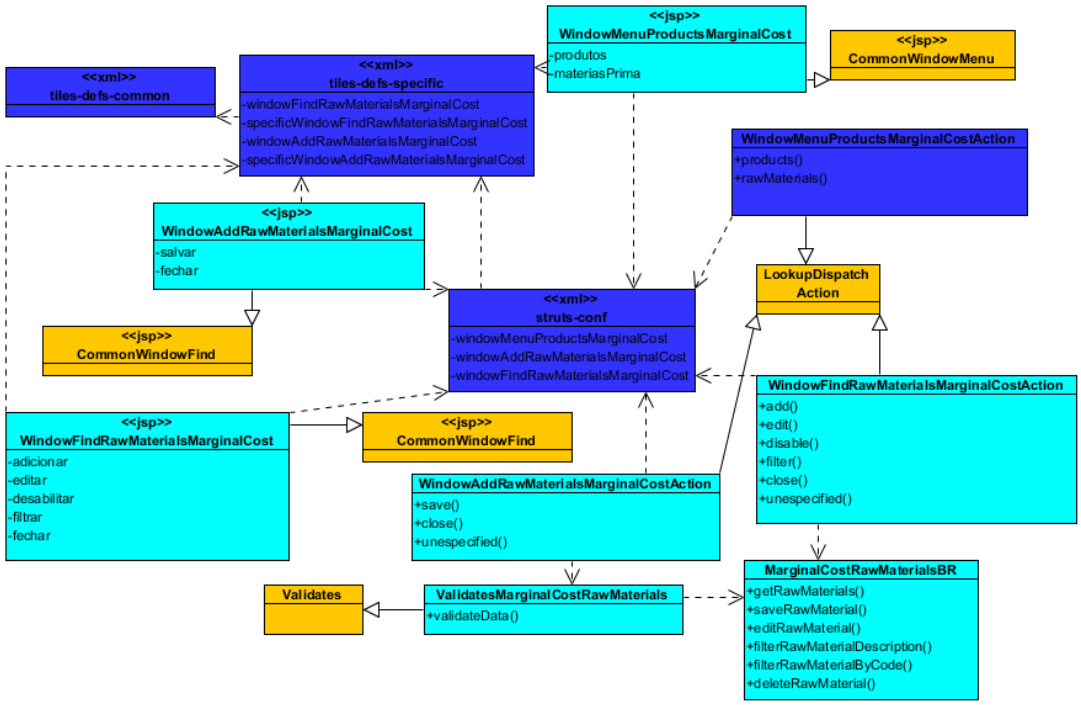

Figura 28 – Diagrama de classe do FrameMK para inserção do subsistema *Gerenciamento de Matéria-prima* do método Marginal.
Fonte: Autoria Própria.

Com objetivo de mostrar todas as matérias-primas cadastradas no sistema se tem o JSP *WindowFindRawMaterialsMarginalCost*. Para manipular as requisições feitas a partir do JSP, foi estabelecida a classe *WindowFindRawMaterialsMarginalCostAction*, que apresenta os mesmos métodos que do Controle de Produtos: *add*, *edit*, *disable*, *filter*, *close* e *unespecified*.

A classe *WindowAddRawMaterialsMarginalCostAction* tem por objetivo processar as requisições para adição de novas matérias-primas, requisições estas vindas do JSP *WindowAddRawMaterialsMarginalCost*. A classe *Action* contém os mesmos métodos da de adição/edição de produtos, a saber, *save*, *close* e *unespecified*.

Para o filtro de requisições e redirecionamento entre os subsistemas, foi alterado o arquivo *struts-config*, adicionando as *Tags action*: *windowAddRawMaterialsMarginalcost* e *windowFindRawMaterialsMarginalCost.* Foi alterado a *Tag action windowMenuProductsMarginalCost* para a definição dos novos JSP's e suas extensões. Novas *Tags definition* foram criadas dentro do arquivo *tiles-defs-specific* tais como: *windowFindRawMaterialsMarginalCost, specifcwindowFindRawMaterialsMarginalCost,*

windowAddRawMaterialsMarginalCost e *specifcwindowFindRawMaterialsMarginalCost.* As duas primeiras *Tag definition* fazem menção ao JSP *WindowFindRawMaterialsMarginalCost* e as outras se referem ao JSP *WindowAddRawMaterialsMarginalCost.*

Finalmente, as classes *MarginalCostRawMaterialsBR* e *ValidatesMarginalCostRawMaterials* foram adicionadas para o controle das regras de negócio do gerenciamento de matérias-primas.

c) Subsistema de Controle de Produto - Gerênciamento de Associação de Produto x Matéria-Prima

Foi necessário criar um módulo de Gerenciamento de Associação de Produto com Matéria-Prima (Figura 29), porque um Produto pode ter várias Matérias-Primas e uma Matéria-Prima pode ter vários Produtos.

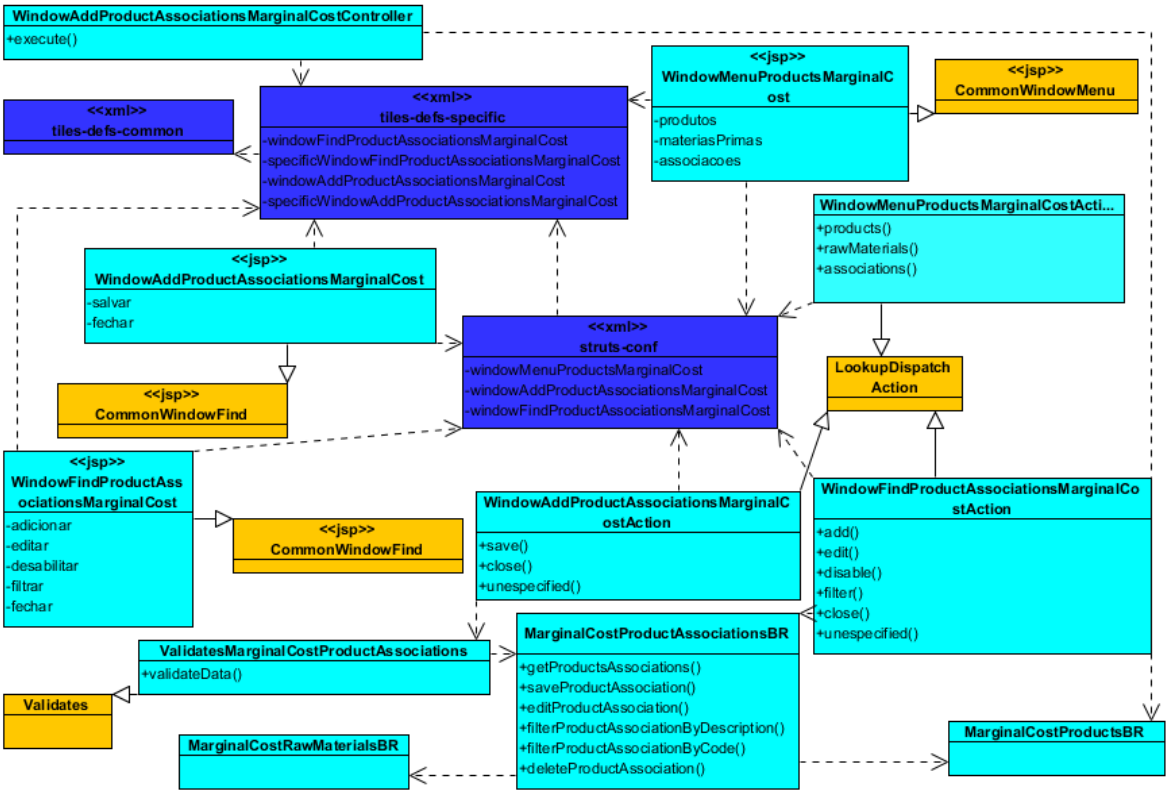

Figura 29 – Diagrama de Classe do FrameMK para o *Gerenciamento de Associação de Produto* com *Matéria-prima* do método Marginal.
Fonte: Autoria Própria.

Este subsistema foi implementado da mesma maneira que os subsistemas de Produto e Matéria-prima, porém, o único ponto em que este se difere dos demais é a adição e edição da Associação, sendo assim, esta diferença será mostrada a seguir.

Para que sejam associadas entidades dentro do *framemk*, foi necessário que adicionar um *combobox* no JSP *WindowAddProductAssociationMarginalCost* porque o JSP base (*CommonWindowAdd*) somente fornece um campo para nome da entidade que está sendo atualizada, adicionando mais um *combobox* no novo JSP foi possível fazer a associação necessária.

Foi preciso criar a classe *WindowAddProductAssociationMarginalCostController* que descende de *ControllerSupport* e *Controller* para buscar os produtos do sistema e adicioná-los ao *ComponentContext*. Para que a busca de produtos seja efetuada a classe *Controller* se utiliza da classe de regra de negócio *MarginalCostProductAssociationsBR*. A classe *Controller* é vinculada ao JSP por meio da *Tag definition specifcwindowAddProductAssociationsMarginalCost* do arquivo *tiles-defs-specific.*

d) Subsistema de Controle de Custo

A Figura 30 apresenta a modelagem proposta para a inserção deste subsistema. Um ponto relevante a se ressaltar é o fato que só podem ser adicionados custos no sistema caso haja algum produto no mesmo.

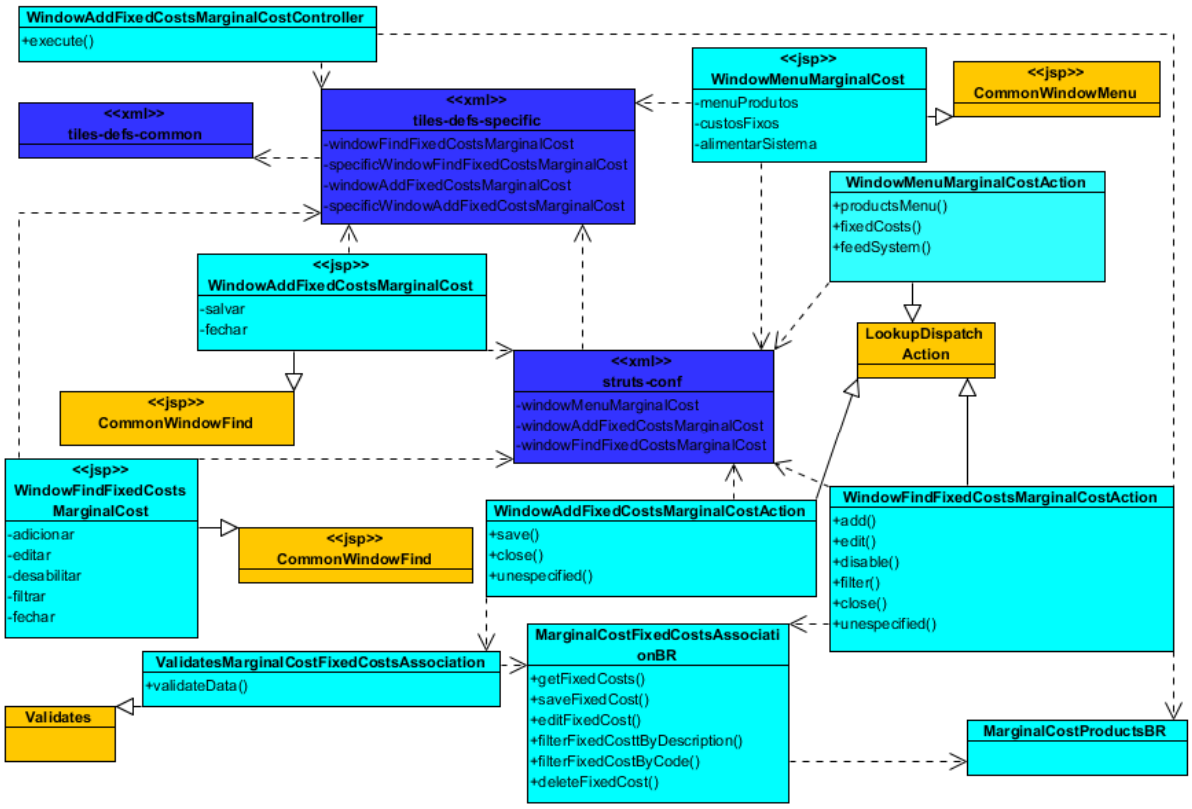

Figura 30 – Diagrama de Classe do FrameMK para *Gerenciamento de Custos Fixos* do método Marginal.
Fonte: Autoria Própria.

Os custos fixos são vinculados diretamente a um produto, não sendo possível a adição de um mesmo custo a mais de um produto.

A modelagem deste subsistema se assemelha ao Gerenciamento de Associação de Produtos com Matérias-Primas da seção anterior. A classe que descende de *Controller* (*WindowAddFixedCostsMarginalCostController*) buscará por produtos, e estes serão adicionados a um *combobox* que se encontra no JSP *WindowAddFixedCostsMarginalCost*.

e) Subsistema de Cálculo de Preço de Venda

O subsistema de preço de venda se preocupa em calcular o preço de venda a partir da fórmula (9) apresentada na seção 2.2, assim, para um determinado produto é feito um somatório de todos os valores de matéria-prima, custos-fixos, custos indiretos de produção, despesas variáveis de venda e administração para obtenção do custo marginal requerido.

Os custos indiretos de produção e despesas variáveis de venda e administração são adicionados no momento do cálculo do método.

A Figura 31 apresenta todas as alterações necessárias no sistema para que este cálculo seja efetuado com sucesso.

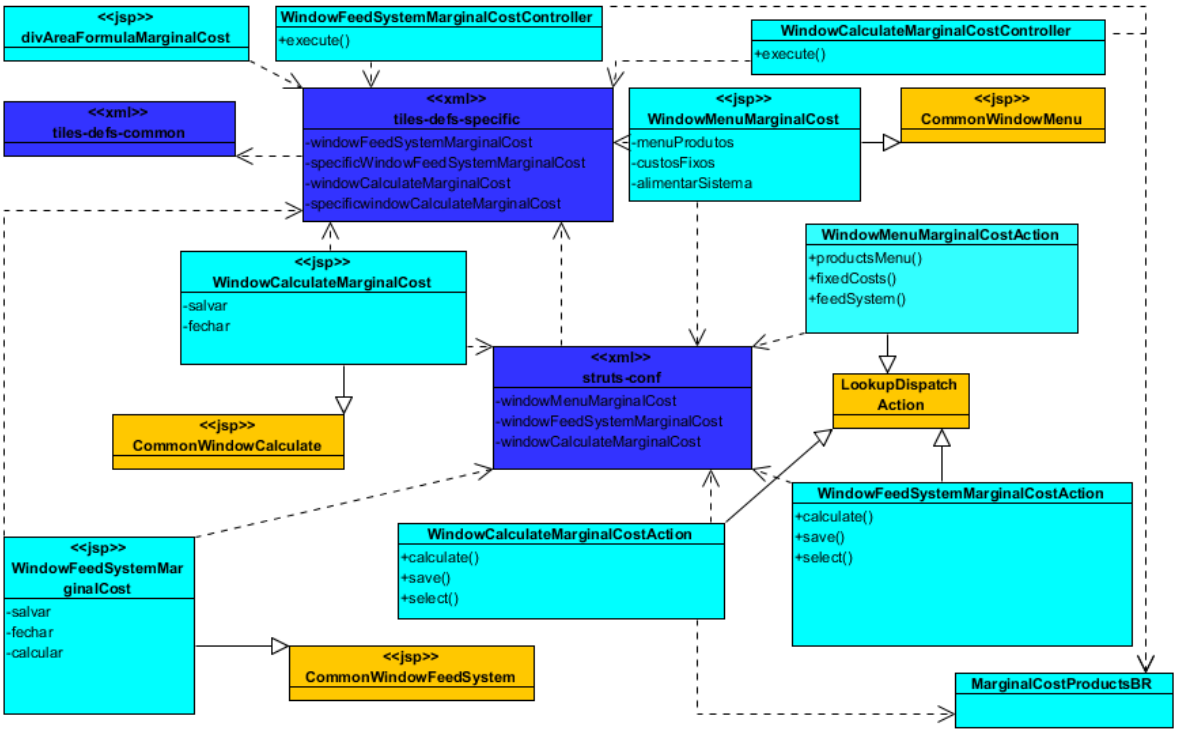

Figura 31 – Diagrama de Classe do FrameMK para *Cálculo* do Método Marginal.
Fonte: Autoria Própria.

Após o clique na opção de Alimentar o Sistema no JSP *WindowMenuMarginalCost*, o método *feedSystem* da classe *WindowMenumarginalCostAction* é chamado e retorna uma chamada de *forward*, essa é processada pelo *struts-config*, que por sua vez redireciona para o método *unespecified* da classe *WindowFeedSystemMarginalCostAction* e esta fará uma nova chamada de *forward* para struts-config, que agora redireciona para o JSP *WindowFeedSystemMarginalCost*.

Para que o JSP *WindowFeedSystemMarginalCost* pudesse descender de *WindowCommonFoodSystem*, são adicionadas duas *Tags definition* no arquivos *tiles-defs-specific* sendo elas: *windowFeedSystemMarginalCost* e *specificwindowFeedSystemMarginalCost*. A primeira *Tag definition* é chamada quando o *forward* para Alimentar o Sistema é disparado. Também são adicionadas

mais duas *Tags definition*, a primeira *windowCalculateMarginalCost* e a segunda *specificWindowCalculateMarginalCost*.

Da mesma forma que quando se quer adicionar um *combobox* a um JSP é necessário que uma classe descenda de Controller, desta maneira é criado uma classe *WindowFeedSystemMarginalCostController*, que tem por objetivo buscar os produtos existentes no sistema e os adicionar no *ComponentContext* para ser usado pelo *combobox*.

Com os produtos definidos no JSP *WindowFeedSystemMarginalCost*, o próximo passo é criar o *link* para cálculo do método custeio Marginal, este por sua vez é feito da seguinte forma: após pressionar o botão “calcular” do JSP, este chama o método *calculate* da classe *WindowFeedSystemMarginalCostAction* que retorna um *forward* o qual é processado e chama o método *unespecified* da classe *WindowCalculateMarginalCostAction*. Este método calcula o custo marginal do produto selecionado e retorna um *forward* para mostrar o resultado do cálculo, este *forward* chama a *Tag definition windowCalculateMarginalCost* que requisita o JSP *WindowCalculateMarginalCost*.

Por fim, para que a fórmula do método de custeio Marginal seja exibida ao usuário juntamente com o resultado obtido do cálculo, é criado o JSP *divAreaFormulaMarginalCost*, o qual é vinculado ao resultado na *Tag definition specificWindowCalculateMarginalCost*.

f) Subsistema de Conexão

Considerando todos os módulos envolvidos no cálculo do método de custeio Marginal, foram criados quatro (4) módulos de persistência, sendo um (1) deles para o subsistema de Gerenciamento de Custos e os demais para o subsistema de Controle de Produtos.

As Figura 32, Figura 33, Figura 34 e Figura 35 apresentam o modelo criado para este subsistema.

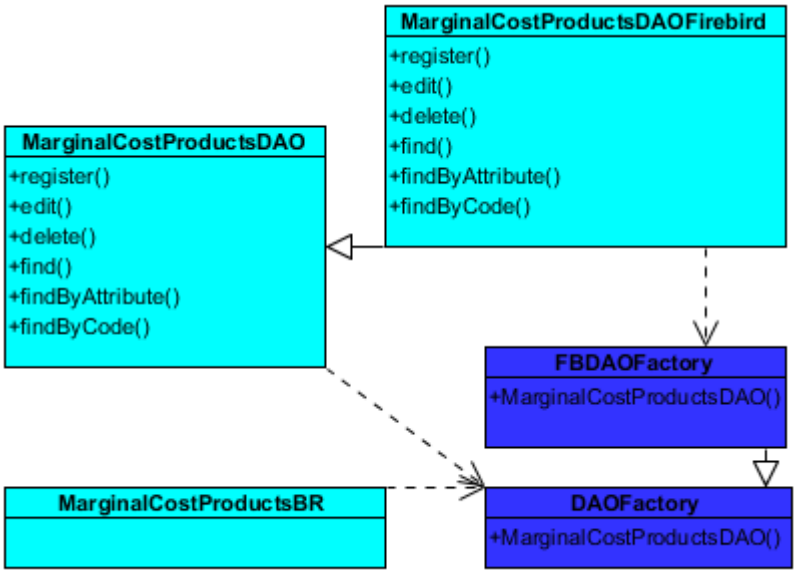

Figura 32 – Diagrama de Classe do FrameMK para *Persistência de Produtos* no método Marginal.
Fonte: Autoria Própria.

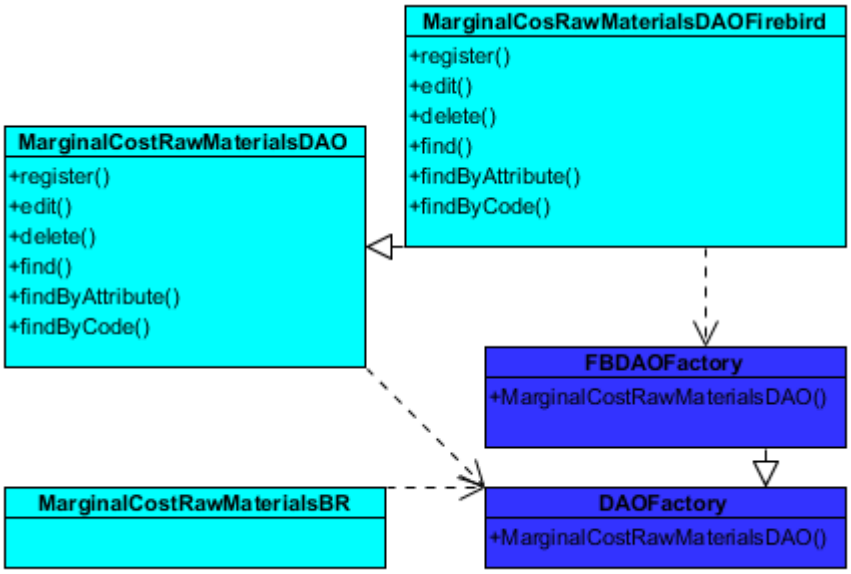

Figura 33 – Diagrama de Classe do FrameMK para *Persistência de Matéria-Prima* no método Marginal.
Fonte: Autoria Própria.

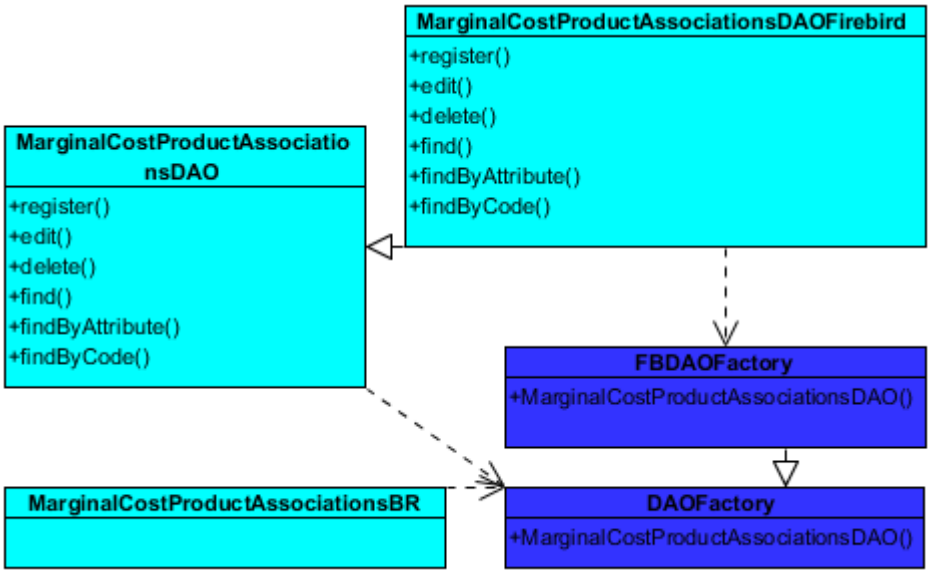

Figura 34 – Diagrama de Classe do FrameMK para *Persistência de Associação de Produto* com *Matéria-Prima* no método Marginal.
Fonte: Autoria Própria.

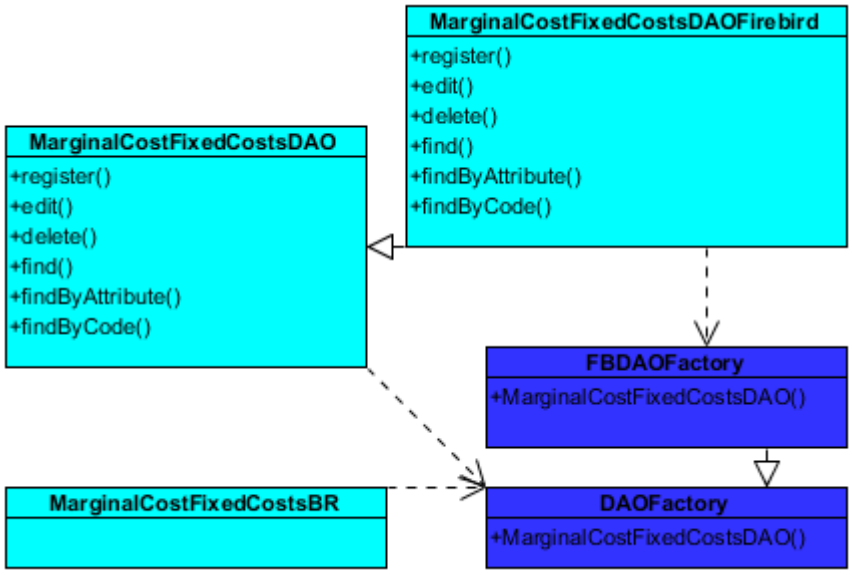

Figura 35 – Diagrama de Classe do FrameMK *para Persistência de Custos Fixos* no método Marginal.
Fonte: Autoria Própria.

As classes com terminação “BR” são as de regra de negócio e fazem o papel intermediário entre a camada de interface e persistência, para que não haja interação direta entre os mesmos.

DAOFactory consiste em um padrão que permite a adição de múltiplas instâncias de diferentes banco de dados, e este é usado neste projeto, sendo que

FBDAOFactory é a única instância de banco de dados encontrada atualmente no FrameMK e faz menção a base para as classes que acessam ao *Firebird*.

As classes *MarginalCostProductsDAO*, *MarginalCostRawMaterialsDAO*, *MarginalCostProductAssociationsDAO* e *MarginalCostFixedCostsDAO* apresentam a generalização para as várias instâncias de diferentes bancos de dados, mais especificamente para o gerenciamento de Produtos, Matérias-Primas, Associação de Produtos com Matérias-Primas e por fim Custos Fixos.

Após as generalizações citadas anteriormente, as mesmas foram criadas para o banco de dados Firebird, e são apresentadas da seguinte maneira: *MarginalCostProductsDAOFirebird*, *MarginalCostRawMaterialsDAOFirebird*, *MarginalCostProductAssociationsDAOFirebird*, *MarginalCostFixedCostsDAOFirebird*. Todas implementam os mesmos métodos de suas classes pai e fazem acesso as suas respectivas tabelas no banco de dados de acordo com a necessidade do FrameMK.

Na seção posterior é abordada a descrição de inserção para o Método ROIC.

Inserção dos Requisitos Pertencentes ao Método ROIC

Assim como na seção anterior, a modelagem do método ROIC também precisou ser reformulada em termos de subsistemas, onde os subsistemas Gerenciar Custo, Matéria-Prima e Produto foram reunidos em um único subsistema denominado de Controle de Ativos.

Inicialmente foi criado o menu para o acesso ao método ROIC, conforme ilustrado na Figura 36.

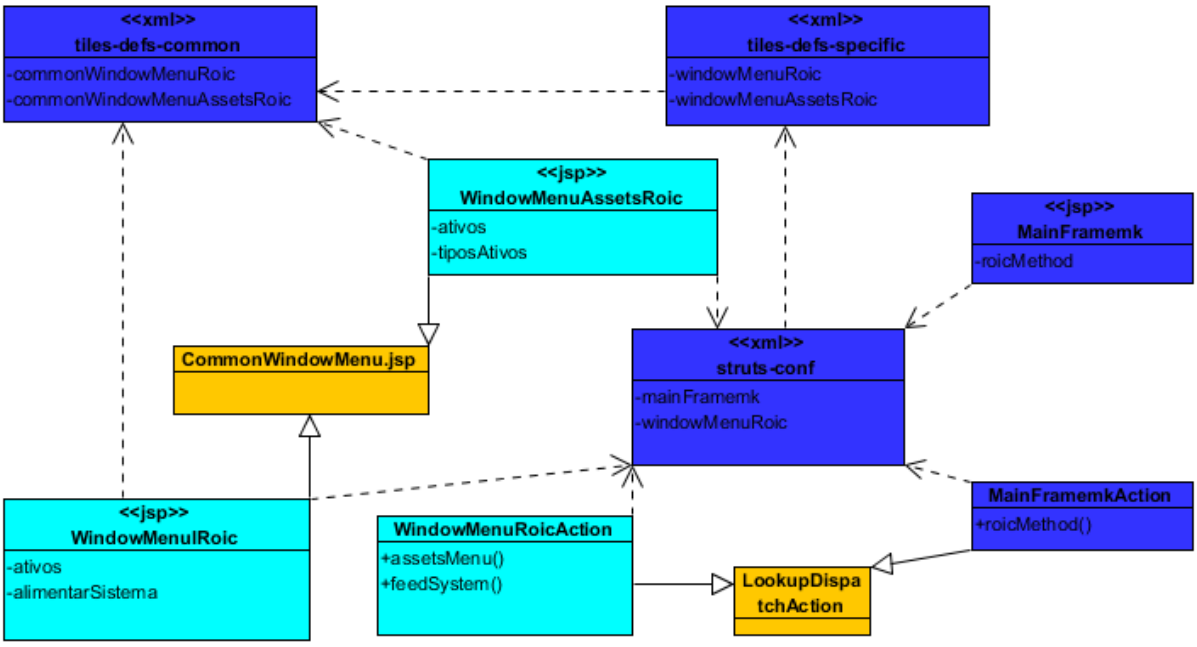

Figura 36 – Diagrama de Classe do FrameMK para Adição de Menus do Método ROIC
Fonte: Autoria Própria.

Além da adição de um menu para o método ROIC, a Figura 35 também apresenta a adição de um menu para Controle de Ativos, ambos os menus descendem do mesmo JSP.

A modelagem para adição destes menus se assemelha a criação de menus para o método de custeio Marginal (Figura 26).

Em primeiro lugar foram criados JSP's para os menus, ou seja, *WindowMenuRoic* e *WindowMenuAssetsRoic*. Para o *WindowMenuRoic* foram criados *links* para ativos e alimentar sistema, e *WindowMenuAssetsRoic* contempla os *links* para ativos e tipos de ativos.

Para que os JSP's descendam de *CommonWindowMenu* foi necessário adicionar *Tags definition* no arquivo *tiles-defs-specific*, definidas as *Tags windowMenuRoic* e *windowMenuAssetsRoic*, e para o arquivo *tiles-defs-common*, criadas as *Tags commonWindowMenuRoic* e *commonWindowMenuAssetsRoic*; as duas últimas *Tags* farão referência aos JSP's *windowMenuROIC* e *windowMenuAssetsRoic*, respectivamente.

O arquivo *struts-config* deve ser alterado para receber requisições dos JSP's já existentes. Após o clique no Menu Lateral para o método ROIC, o método *roicMethod* é chamado na classe *MainFramemkAction*, esta classe chama o *forward* referente ao método requisitado e é disparado o JSP *windowMenuRoic*. Por fim,

quando é clicado no menu para Ativos no menu ROIC, chama-se o método *assetsMenu* dentro da classe *WindowMenuRoicAction* o qual requisita o *forward* referente ao menu requisitado e é disparado o JSP *windowMenuAssetsRoic*.

As próximas subseções descrevem o processo de inserção dos requisitos de cada subsistema deste método.

a) Subsistema de Controle de Ativos

Este subsistema permite o gerenciamento de ativos, bem como os seus tipos. As

Figura 37 e 38 apresentam a modelagem proposta para o controle de ativos.

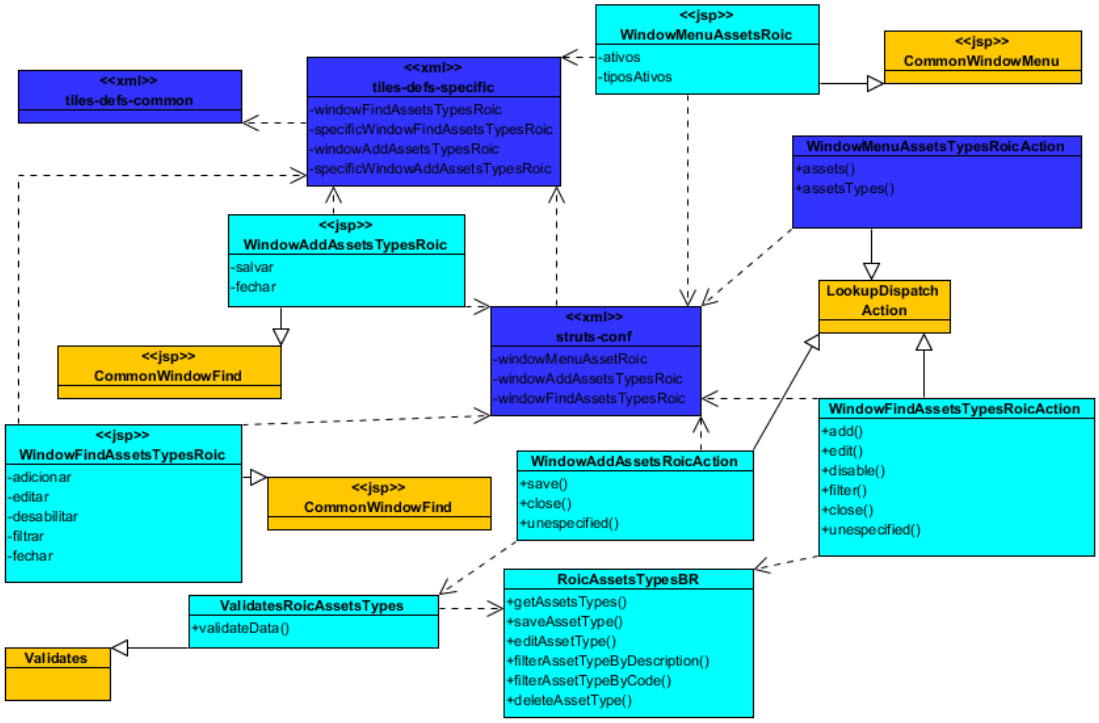

Figura 37– Diagrama de Classedo FrameMK para *Gerenciamento de Tipos de Ativos* do Método ROIC.
Fonte: Autoria Própria.

Este módulo apresenta a mesma característica que os módulos de gerenciamento de produtos e de matérias-primas do subsistema de controle de produto do método Marginal, pois da mesma forma que esses módulos não

precisam de outras inserções no sistema para funcionarem, o módulo de gerenciamento de tipos de ativos também não.

Os Tipos de Ativos são necessários, pois para o cálculo do método, não é possível adicionar um Ativo de maneira genérica, já que o mesmo não teria um resultado correto, é necessário dar um foco ao Ativo, por exemplo Ativo Operacional, Ativo Financeiro, Ativo Circulante, Ativo Não-Circulante e etc; estes são os Tipos de Ativos.

A arquitetura deste subsistema é a mesma dos subsistemas já apresentados, sendo assim, seguem as modificações:

- Criação de classes *Action*: *WindowAddAssetsTypesRoicAction* e *WindowFindAssetsTypesRoicAction*.
- Criação de JSP's para gerenciamento de Tipos de Ativos: *WindowAddAssetsTypesRoic* e *WindowFindAssetsTypesRoic*.
- Criação de classes *ValidatesRoicAssetsTypes* e *RoicAssetsTypesBR* para controle das regras de negócio.
- Alteração de XML *tiles-defs-specific* com adição de *Tags definition* para criação de descendência entre JSP's e adição de parâmetros (*windowFindAssetsTypesRoic*, *specificWindowFindAssetsTypesRoic*, *windowAddAssetsTypesRoic* e *specificwindowAddAssetsTypesRoic*).
- Alteração de XML *struts-config* com adição de *Tags action*, para recebimento de requisições de JSP's e redirecionamento de requisições (*windowAddAssetsTypesRoic* e *windowFindAssetsTypesRoic*).

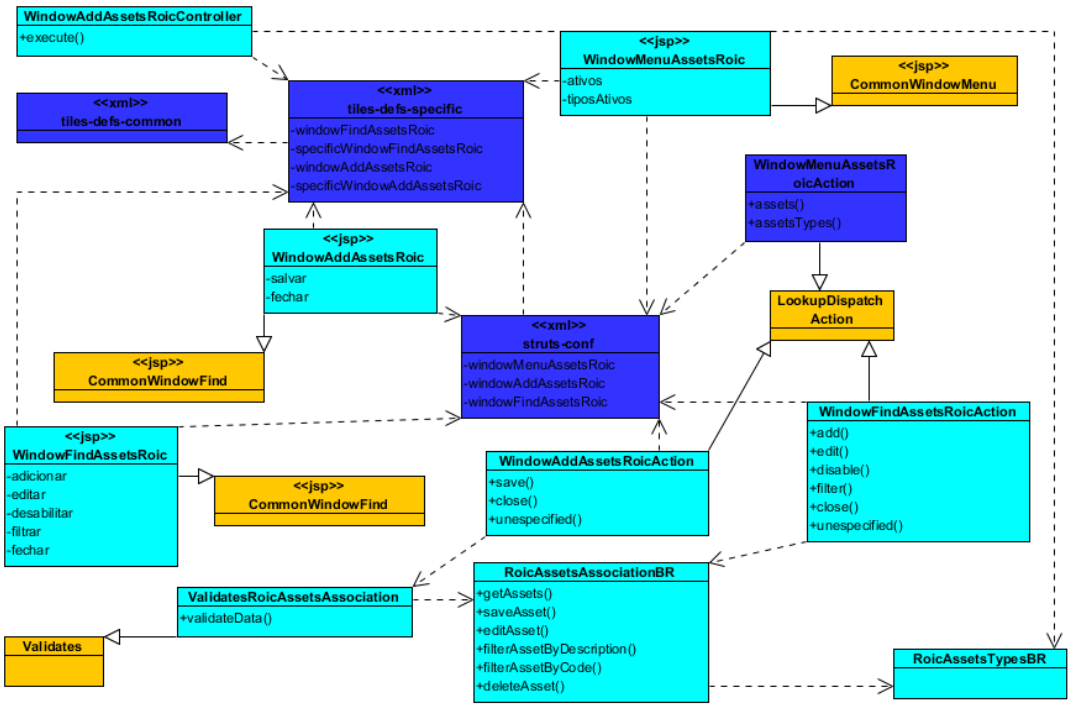

Figura 38 – Diagrama de Classe do FrameMK para *Gerenciamento de Ativos* do método ROIC. Fonte: Autoria Própria.

Com os tipos de ativos definidos, é possível que se criem ativos que sejam agrupados cada um por seu tipo, por exemplo, para o Tipo de Ativo “Cirlulante” é possível que sejam adicionados Ativos com descrição “depósitos bancários”, “dívidas de clientes”, “mercadorias para venda”, entre outros. Outro exemplo é que para o Tipo de Ativo “Não-Circulante”, podem-se ter os Ativos com descrição “Edificios”, “Ferramentas”, “Terrenos”, “Marcas” e etc.

Da mesma maneira que é possível comparar o gerenciamento de produtos com o gerenciamento de tipos de ativos, também, é possível observar o gerenciamento de custos fixos e concluir que possuem a mesma arquitetura. Portanto, seguem as alterações feitas, tomando como base o gerenciamento de custos fixos do método Marginal:

- Criação de classes *Action*: *WindowAddAssetsRoicAction* e *WindowFindAssetsRoicAction*.
- Criação de JSP’s para gerenciamento de Ativos: *WindowAddAssetsRoic* e *WindowFindAssetsRoic*.

- Criação de classes *ValidatesRoicAssetsAssociation* e *RoicAssetsBR* para controle das regras de negócio.
- Alteração de XML *tiles-defs-specific* com adição de *Tags definition* para criação de descendência entre JSP's e adição de parâmetros (*windowFindAssetsRoic*, *specificWindowFindAssetsRoic*, *windowAddAssetsRoic* e *specificwindowAddAssetsRoic*).
- Alteração de XML *struts-config* com adição de *Tags action*, para recebimento de requisições de JSP's e redirecionamento de requisições (*windowFeedSystemRoic*, *windowCalculateRoic*).

b) Subsistema de Cálculo de Preço de Venda

Este subsistema faz o cálculo do método a partir da fórmula (10) apresentada na seção 2.2, sendo assim para um determinado produto é feito um somatório de todos os valores dos ativos referentes a um determinado tipo.

Vale lembrar que não somente os Ativos devem estar agrupados de acordo com o Tipo de Ativo, mas também, quando for fornecido determinado valor pré-calculado de Lucro para o cálculo da fórmula, ele deve estar calculado a partir do mesmo Tipo de Ativo.

A Figura 39 apresenta todas as alterações necessárias no sistema para que este cálculo seja efetuado com sucesso. Em que teve-se:

- Criação de classes *Action*: *WindowCalculationRoicAction* e *WindowFeedSystemRoicAction*.
- Criação de JSP's para alimentar o sistema e mostrar os resultados obtidos juntamente com a fórmula: *WindowFeedSystemRoic*, *WindowCalculateRoic* e *divAreaFormulaRoic*.
- Criação de classes *RoicAssetsBR* e *RoicAssetsTypesBR* para controle das regras de negócio.
- Alteração de XML *tiles-defs-specific* com adição de *Tags definition* para a criação de descendência entre JSP's e adição de parâmetros (*windowFeedSystemRoic*, *specificWindowFeedSystemRoic*, *windowCalculateRoic* e *specificWindowCalculateRoic*).

- Alteração de XML *struts-config* com adição de *Tags action*, para recebimento de requisições de JSP's e redirecionamento de requisições (*windowAddAssetsRoic* e *windowFindAssetsRoic*).

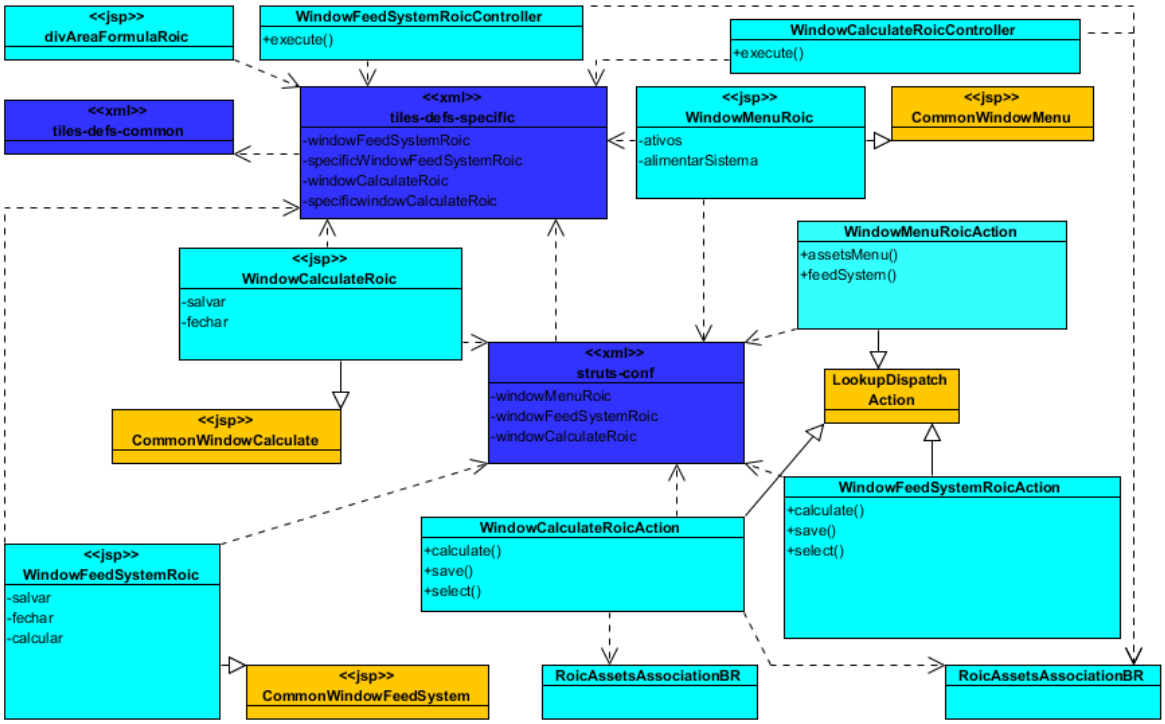

Figura 39 – Diagrama de Classe do FrameMK para *Cálculo* do Método ROIC.
Fonte: Autoria Própria.

c) Subsistema de Conexão

A partir das classes genéricas *RoicAssetsTypesDAO* e *RoicAssetsDAO* foram criadas duas especificações, uma para cada uma das classes genéricas, a saber, *RoicAssetsTypesDAOFirebird* e *RoicAssetsDAOFirebird*.

A partir das especificações é possível manipular as tabelas de Ativos e Tipos de Ativos presentes no banco de dados do Firebird, para funções como adição, deleção, edição, busca, busca por atributo e busca por código. A classe *RoicAssetsTypesDAOFirebird* controla os registros de Tipos de Ativos e a *RoicAssetsDAOFirebird* controla os registros de Ativos.

As Figura 40 e Figura 41 apresentam uma melhor descrição destes módulos.

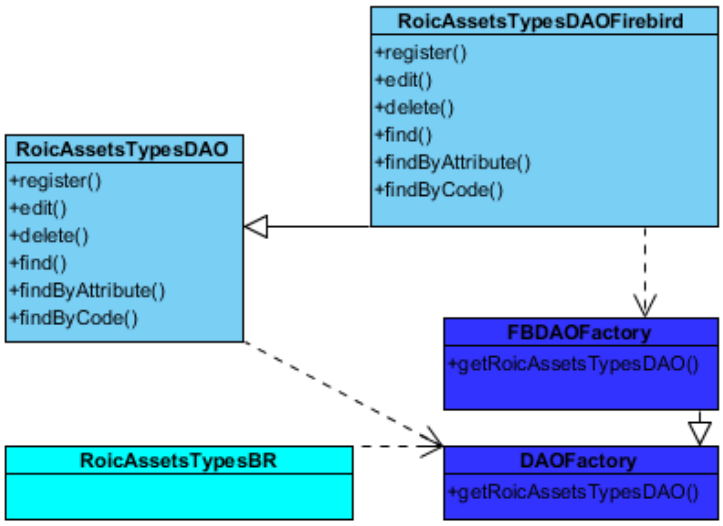

Figura 40 – Diagrama de Classe do FrameMK para *Persistência de Tipos de Ativos*.
Fonte: Autoria Própria.

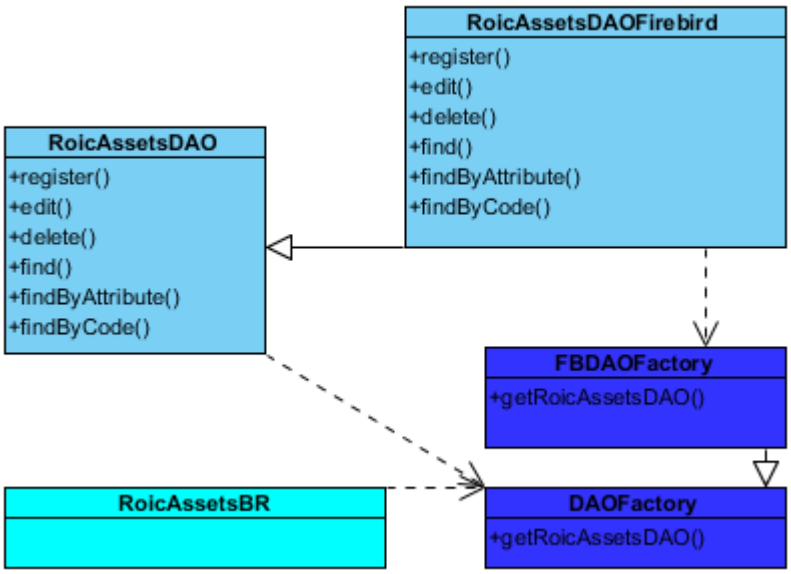

Figura 41 – Diagrama de Classe do FrameMK para *Persistência de Ativos*.
Fonte: Autoria Própria.

Posteriomente, na seção 3.3.3 relata a próxima etapa do método proposto.

3.3.3 Testar novo Requisito

Esta etapa tem a finalidade de realizar os testes para verificar se os requisitos inseridos estão realizando as suas funcionalidades esperadas e se não foram motivos de erros no FrameMK.

Os testes realizados neste trabalho se basearam no método de teste de aceitação e foram executados em tempo de execução sem o uso de classes de teste, ou seja, ao serem inseridas novas funções, variáveis, os testes eram realizados. Desta forma, foi possível analisar os resultados esperados com os resultados obtidos.

O capítulo seguinte apresenta os resultados obtidos nesta etapa.

4 RESULTADOS

Este capítulo apresenta os testes realizados no FrameMK após a inserção dos novos subsistemas, além disto, relata uma análise realizada sobre o tempo gasto com testes e as dificuldades encontradas durante o processo de extensão do *framework*. A Seção 4.1 descreve o tipo de teste e análise do tempo gasto após a inserção das novas funcionalidades. A Seção 4.2 aborda os resultados das novas adições em tempo de execução. A Seção 4.3 apresenta a avaliação do processo de extensão do FrameMK.

4.1 TESTE DE UNIDADE PARA OS NOVOS REQUISITOS

Após a identificação dos novos requisitos a serem implementados, foram realizados testes de aceitação utilizando um modelo de caso de teste, ilustrado no Quadro 7. Os elementos o modelo são: nome, resultado esperado, resultado obtido, tempo utilizado, conclusão e considerações (GOMES; SILVA, 2009).

Nome do caso	Nome do caso específico.
Resultado esperado	Como o programa deve se comportar.
Resultado obtido	Como o programa se comportou.
Tempo utilizado	Tempo transcorrido para o desenvolvimento do teste.
Conclusão	Comparação dos resultados.
Considerações no decorrer do teste	Considerações finais e observações sobre o caso.

Quadro 7 – Modelo de caso de teste
Fonte: Gomes e Silva (2009).

Seguindo o modelo, explica-se a seguir os testes para o subsistema *Production Line*. O Quadro 8, ilustra um caso de teste efetuado com sucesso.

Nome do caso	Adição de *Link ProductionLine* em JSP
Resultado esperado	Habilitar *link* no sitio do FrameMK de Linha de Produção
Resultado obtido	Habilitar *link* no sitio do FrameMK de Linha de Produção
Tempo utilizado	3 minutos.
Conclusão	O resultado obtido foi igual ao esperado, pois mostrou o *link* da Linha de Produção no sitio do FrameMK.
Considerações no decorrer do teste	Teste realizado ao executar a Classe *WindowMenuAbc.jsp* que carrega os *links* inseridos anteriormente.

Quadro 8 – Caso de Teste para habilitar *link* para Linha de Produção.
Fonte: Autoria Própria.

O Quadro 9, descreve o teste realizado no subsistema *Product*. A operação se destina a testar a *Tag Action* para a adição de produto.

Nome do caso	Adição da *Tag Action* para *WindowAddProduct* no arquivo *struts-config*.xml
Resultado esperado	Chamar *Tag Action* de adição de produto.
Resultado obtido	Chamar *Tag Action* de adição de produto.
Tempo utilizado	4 minutos.
Conclusão	O resultado obtido foi exatamente o que se esperava. Ao ser acionado através de um *click*, adicionar ou editar, o *forward windowsAddProduct* é chamado e dispara a *Tag Action* de adição de produto.
Considerações no decorrer do teste	Teste realizado ao executar as opções de adicionar ou editar produto na classe *WindowFindProduct*.jsp

Quadro 9 – Caso de Teste na adição da *Tag Action* para *WindowAddProduct* em *struts-config*.xml.
Fonte: Autoria Própria.

Além dos casos de teste exemplificados, foram testadas todas as funcionalidades dos subsistemas pertencentes aos métodos ABC. Desta forma, todas as operações dos subsistemas como: cadastrar, consultar, editar e desabilitar tiveram os resultados analisadas, além das operações de acesso ao menu.

Ao todo, foram realizados vinte e um (21) testes, sendo sete (7) para o subsistema *Production Line*, sete (7) para subsistema o *Product* e sete (7) para *Activity*. Do total de vinte e um, obtiveram sucesso 16 testes e apenas 5 revelaram erros inesperados necessitando ajustes.

Um ponto para que a quantidade de erros fosse baixa, está realacionado a própria linguagem java que tem a vantagem de tratar determinados erros sintáticos, como por exemplo: não permitir atribuições indevidas, realizar testes do código conforme à medida que o programador desenvolve e ainda sugere a inserção de exceções.

O Gráfico 1, ilustra a quantidade de testes que obtiveram sucesso e os que os resultados que não foram os esperados no primeiro momento.

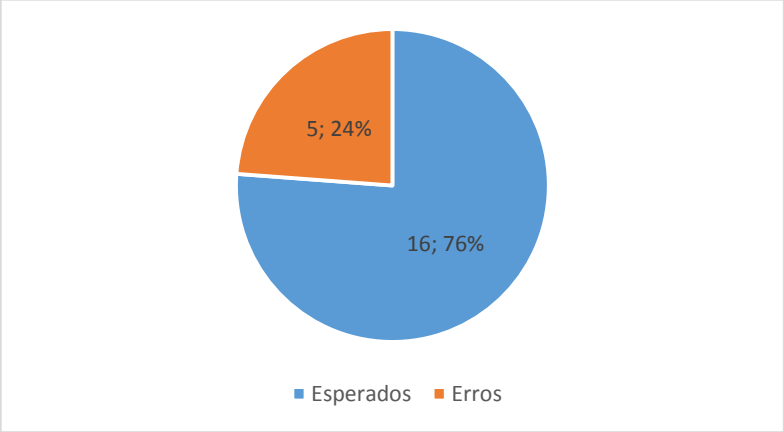

Gráfico 1 – Total de testes realizados.
Fonte Autoria Própria.

A relação entre o tempo necessário em minutos em cada operação é exibido no Gráfico 2.

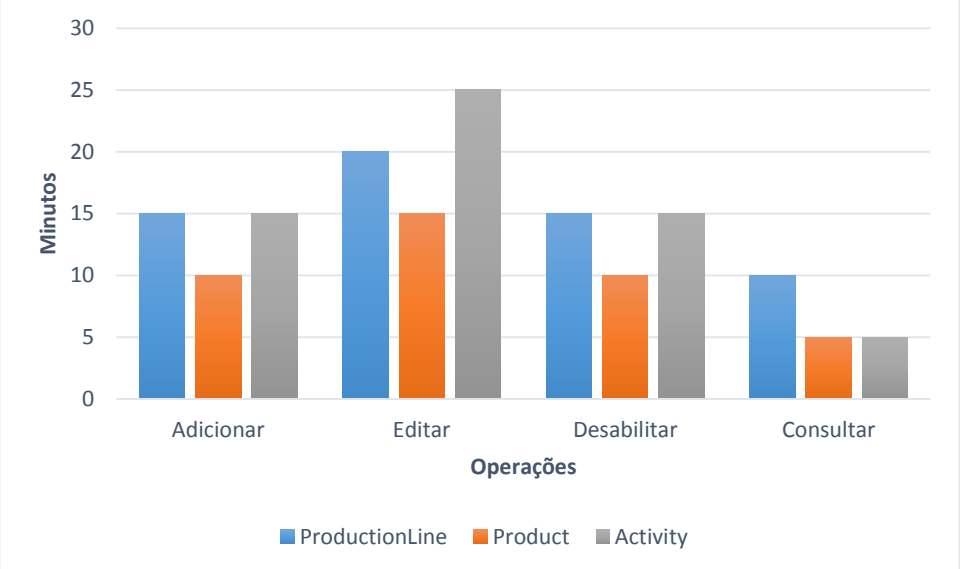

Gráfico 2 – Tempo utilizado para a realização dos testes de unidade.
Fonte Autoria Própria.

Apesar do tempo que se levou para a aplicação de testes, estes permitiram deixar os novos códigos funcionando e obteve-se maior qualidade no FrameMK. Ressalta-se que não foram realizados testes para os métodos ROIC e Marginal, pois este não foram implementados neste trabalho.

Conforme o Gráfico 2, o tempo total para cada operação foi obtido pela soma de todos os testes realizados. Por exemplo, o subsistema *Activity* teve maior tempo de teste na operação Editar, pois a sua lógica é mais complexa.

Na próxima seção é visualizado o resultado final dos requisitos inseridos no protótipo do FrameMK.

4.2 RESULTADO DA INSERÇÃO DOS NOVOS REQUISITOS NO PROTÓTIPO DO FRAMEMK

A Figura 42 ilustra o menu com todas as opções habilitadas para o método ABC. A Figura 10 mostrou que as opções *Linhas de Produção, Produto e Atividade* estavam desabilitadas. Figura 43

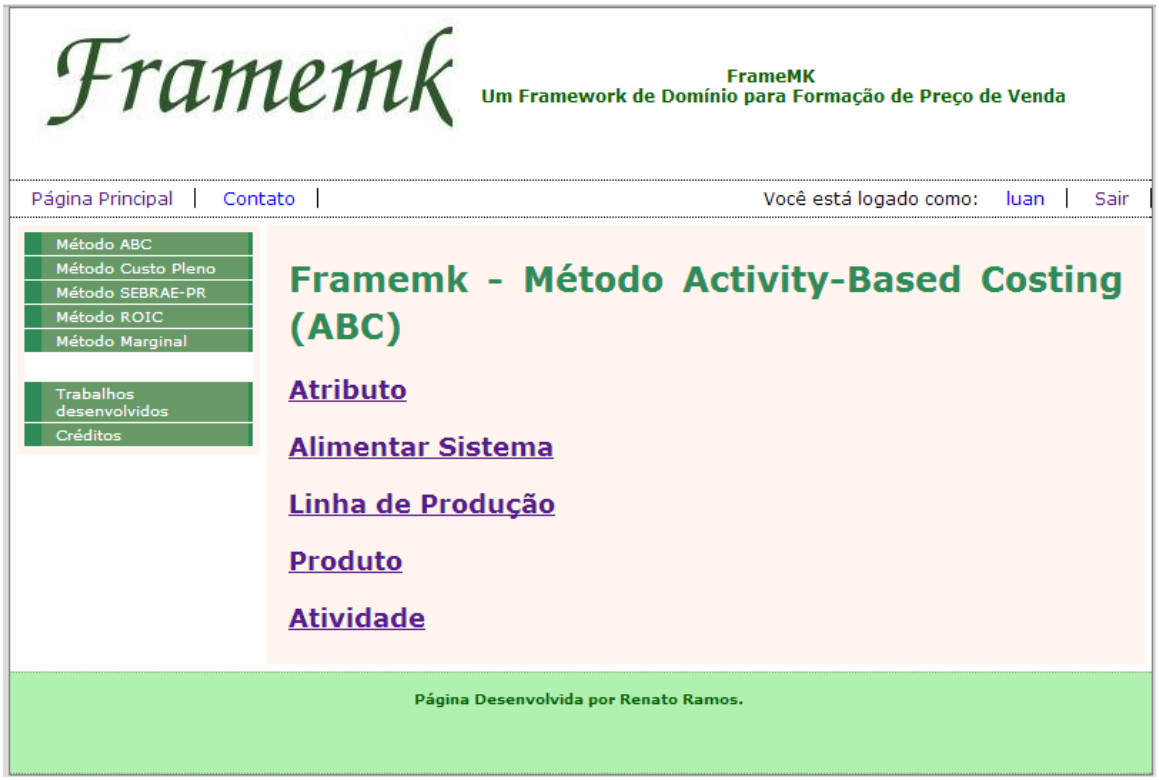

Figura 42 – Menu de opções do método ABC com todas as opções habilitadas.
Fonte: Autoria Própria.

A Figura 43 exibe as opções: editar, desativar, adicionar e consultar do subsistema *Production Line*.

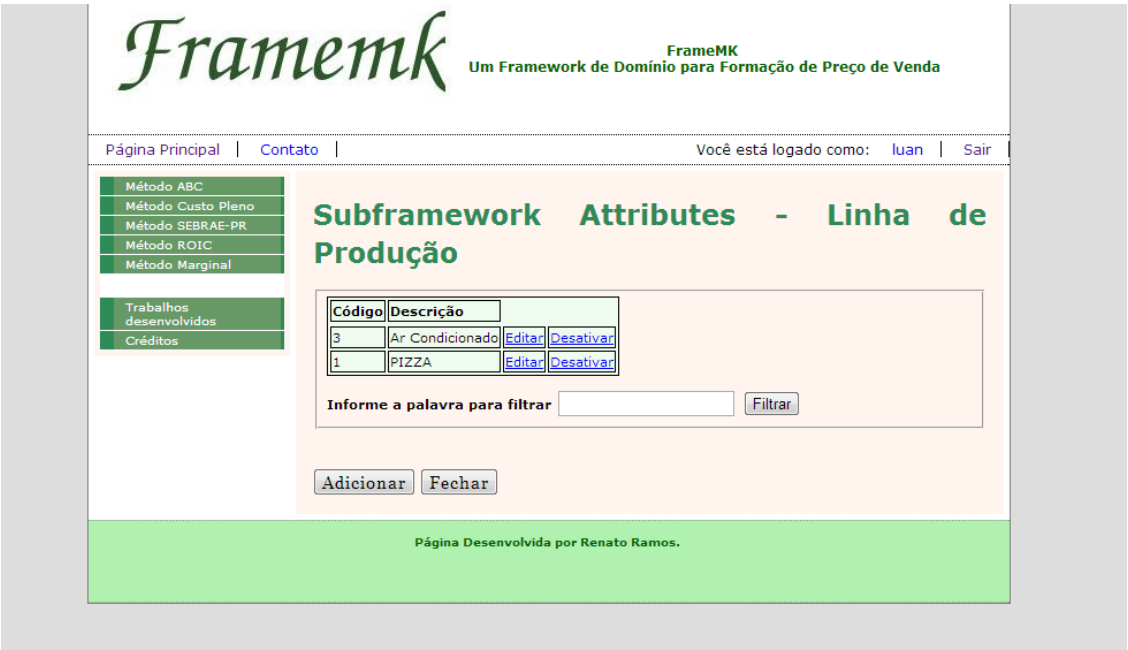

Figura 43 – Resultado da Implementação dos Requisitos para o Subsistema *ProductionLine.*
Fonte: Autoria Própria.

A Figura 44, ilustra as opções habilitadas para adicionar, editar, consultar e desativar produtos do subsistema *Product*.

Figura 44 – Resultado da Implementação dos Requisitos para o Subsistema *Product* do método ABC.
Fonte: Autoria Própria.

A Figura 45 exemplifica a opção de adição ou edição de um produto.

Figura 45 – Resultado de adição ou edição de Produtos do método ABC do FrameMK.
Fonte: Autoria Própria.

A Figura 46 exibe as opções de adicionar, editar, consultar e desativar habilitadas para o subsistema *Activity*.

Figura 46 - Resultado da Implementação dos Requisitos para o Subsistema *Activity* do método ABC
Fonte: Autoria Própria.

A Figura 47, exemplifica os resultados para adição ou edição de Atividade. Ao adicionar uma Atividade é necessário realizar seu vinculo a uma Linha de Produção.

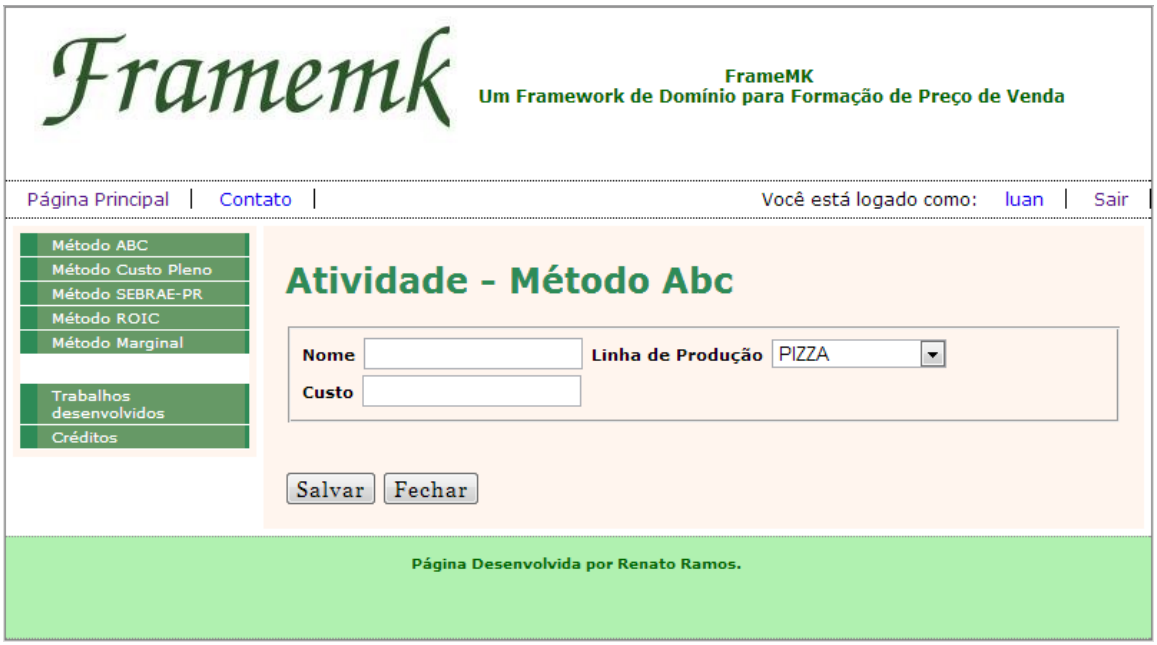

Figura 47 - Resultado de adição ou edição de Atividade.
Fonte: Autoria Própria.

As implementações para os métodos Marginal e ROIC não foram colocadas neste trabalho, pois o objetivo era modelá-las conforme apresentado no capítulo 3. A implementação destes métodos seguindo o modelo proposto será realizada em um trabalho posterior.

4.3 AVALIAÇÃO DO PROCESSO DE EXTENSÃO DO FRAMEMK

Durante a extensão do FrameMK foram notadas algumas dificuldades e facilidades no uso do processo proposto.

Na etapa de *Análise e Identificação de Novos Requisitos* a serem Implementados, foi possível observar que Capeller e Andrade (2010) já haviam compreendido e modelado os requisitos do método ABC. Oliveira e Crema (2009) da mesma forma fizeram modelagens referentes aos métodos ROIC e Marginal, porém, algumas modificações nas modelagens foram feitas nestes dois métodos, pois foi observado que o modelo não abrangia os métodos como um todo.

Para que fossem implementados os métodos, foi necessário descobrir os pontos corretos para sua inserção, por isto foi realizado o entendimento da arquitetura do *framework* em questão e logo após, iniciou-se o desenvolvimento. Para a etapa de *Implementação de um novo requisito* e *Levantamento de pontos de junção no framework*, a primeira dificuldade foi o entendimento do *framework Struts* e o funcionamento da arquitetura do FrameMK.

Na etapa de *Teste dos novos requisitos*, foi selecionado um modelo de como os testes seriam efetuados, a partir dele foram realizados vários casos de testes pensando nos mais que relevantes. Como resultado de cada um deles (teste bem sucedido ou teste mal sucedido) foram montados gráficos de pizza e barra para comparação.

Finalmente, constatou-se que o processo sugerido foi importante porque permitiu observar e validar novos requisitos a serem adicionados em um *framework* de domínio, implementá-los e verificar a acurácia da inserção de cada um dentro do modelo.

5 CONCLUSÃO

Dentre os processos de extensão de *framework* apresentados neste trabalho, escolheu-se propor um processo adaptado de Camargo e Masiero (2005) para que se conseguisse estender o FrameMK.

Foram analisados possíveis pontos de junção do FrameMK para adição dos novos requisitos dos métodos ABC, ROIC e Marginal. O método ABC teve seus subsistemas de *Production in Line*, *Product* e *Activity* codificados, usando como base a modelagem de Capeller e Andrade (2010). Em relação aos métodos ROIC e Marginal, apesar de terem sido estudados previamente por Oliveira e Crema (2009) , tiveram suas modelagens refeitas para melhor atender ao cálculo de seus respectivos métodos. Foram modelados subsistemas de Controle de Produtos, Controle de Custos Fixos, Cálculo do Preço de Venda e Conexão para o método Marginal e os subsistemas de Controle de Ativos, Cálculo do Preço de Venda e Conexão para o método ROIC.

Os subsistemas propostos para os métodos ROIC e Marginal foram somente modelados e incorporados a estrutura do FrameMK.

Foram realizados testes para verificar se o modelo proposto de inclusão das funcionalidades no FrameMK estavam corretos.

5.1 TRABALHOS FUTUROS

Algumas das propostas para continuação deste trabalho são:

- Implementação dos métodos ROIC e Marginal dentro do Framemk a partir do modelo criado.
- Estudar e desenvolver novos métodos de precificação de produto para incorporá-los ao FrameMK, tornando-o um *framework* de caixa-preta.
- Estudar outros *frameworks* de aplicação que implementam o padrão de *front-controller* para atualização e refatoração do FrameMK, pois o *framework Struts* já se encontra antiquado em comparação a outros apresentados com a linguagem Java.

REFERÊNCIAS

ANDRADE, V. C; CAPELLER, P. E. B. **Uso do processo dirigido a responsabilidades no desenvolvimento da arquitetura e modelagem do framework de preço de venda**. 2010. 164f. Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) - UTFPR, Pronta Grossa - PR, 2010.

ALPI, S. C. **Procedimentos para a Construção de um Sistema de Informação para Rateio de contas Telefônicas em uma Indústria de Bebidas**. 2007. 35f. Trabalho de Conclusão de Curso (Administração em Sistema de Informação Gerencial) – FAQ, Socorro, 2007.

ASSIS, F. B.. **Desenvolvimento de Software Dirigido por Teste de Aceitação**. 2012, 36f. Trabalho de especialização em Informática (Engenharia de Software) – UFMG, Belo Horizonte, 2012.

BARATA, P. V. A. et al. **Retorno Sobre o Investimento do Ponto de Vista da Empresa e do Empresário**. 2003, 15f. Artigo (Ciências Contábeis) - Universidade Federal do Pará, Belém, 2003.

BRAGA, R. T. V.; MASIERO, P. C. Identification of framework hot spots using pattern languages. **In:** SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), 15, 2001, Rio de Janeiro.

BORNIA, A. C. **Análise gerencial de custos – aplicação em empresas modernas.** Porto Alegre: Bookman, 2002.

BOSCH, J. et al.. Object-Oriented Frameworks - Problems & Experiences. Ronneby, Suécia. University of Karlskrona, 1997, 20 p., **Relatório de técnico**.

CAFEO, B. B. P. **Teste estrutural de integração contextual (Nível N) de programas orientados a objetos e a aspectos: especificação do grafo de controle de programa integrado e estudo de critérios e restrições**. 2009. 58f.

Trabalho de diplomação (Bacharel em Ciência da Computação) – USP, São Paulo, 2009.

CAMARGO, V. V.; MASIERO, P. C. Uma Abordagem de Evolução de Sistemas Orientados a Objetos Apoiada por Frameworks Transversais, 2005, Uberlândia. **Anais** do SBES 05, 2005, Minas Gerais, p. 18-33.

CRAZUSKI, A.; FEITOSA L. B.; CORDEIRO, T. L.. **Identificação dos pontos de estabilidade e de flexibilidade dos métodos para o estabelecimento de preço de venda**. 2008. 157f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa, 2008.

CUSTOS COMÉRCIO. *Custos e formação de preços - Comércio*. Em: <http://www.baixaki.com.br/download/custos-e-formacao-dos-precos-comercio.htm>. Acesso em: 05 maio 2013.

FAYAD, M. E.; SCHMID D. C. **Implementing application frameworks:** object-oriented frameworks at work.New York: Wiley Computer, 1999. 729p.

GOMES, E. W. C.; SILVA, R. A. **Verificação do *subframework* de análise semântica de fórmulas utilizando testes de software na fase de unidade.** 2009. 159f. Trabalho de diplomação (Tecnologia m análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa – PR, 2009.

GPES. Grupo de Pesquisa de Engenharia de Software - GPES. Disponível em <http://www.pg.utfpr.edu.br/gpes/> acesso em 20 junho 2013.

JUNIOR, C. R.. **Um Web Service para busca de Preço de Venda**. 2010, 72f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa, 2010.

LACERCA, V. S.. **Refatoração do aplicativo gerenciador de menus dinâmicos do sítio Arcabomk**. 2012, 101f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa, 2012.

LEÃO, N. S. **Formação de Preços de Serviços e Produtos**. 1ª ed., São Paulo: Nobel, 2008.

MARTINS, E. **Contabilidade de Custos**. 8ª ed., São Paulo: Atlas, 2001.

MATOS, S. N. Definição Um Panorama dos Processos de Desenvolvimento de Framework de Domínio. 2007, 42f. **Relatório Técnico-Científico** (Engenharia Eletrônica e Computação) – ITA, São José dos Campos, 2007.

MATTSSON, M. **Object-Oriented Frameworks - A survey of methodological issues**. 1996, 128f. Tese (Ciência da Computação e Administração de Negócios) - University College of Karlskrona/Ronneby, Suécia, 1996.

MOFFSOFT: Moffsoft Calculator*. Version 2.* Moffsoft.com, 2013.

MYERS, G. J. ***The Art of Software Testing***. 2ª ed., New Jersey: John Wiley & Sons, Inc., 2004.

OLIVEIRA, R. R. de.; CREMA, R. J. C. **Definição dos pontos de estabilidade e de flexibilidade, em nível de requisitos, no domínio de preço de venda**. 2009. 193f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa, 2009.

PADOVEZE, C. L. **Contabilidade Gerencial: um enfoque em sistema de informação contábil**. 5ª ed., São Paulo: Atlas, 1997.

RAJLICH, V. MSE: A Methodology for Software Evolution. ***Software Maintance: Research and Practice***, *Journal of Software Maintenance*, Detroid, vol.9, p.103-125, 1997.

RAMOS, R. **Refatoração da camada de apresentação do framework de preço de venda (Framemk)**. 2011, 64f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas) – UTFPR, Ponta Grossa, 2011.

SAITO, S., YAKAMOTO, S. The Incremental Goal Evolution Process Methodology. ***Business/IT Aligment and Interoperability***, BUSITAL, Valencia, vol.237. 2006.

SANTOS, J. J. **Formação do Preço e do Lucro**. 4ª ed., São Paulo: Atlas, 1994.

SILVA, L. S. **Um método para identificação de aspectos em nível de análise baseados em atributos de requisitos não-funcionais**. 2012. 92f. Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) - UTFPR, Ponta Grossa - PR, 2012.

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000, 262f. Tese (Doutorado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

SOLVEIT: SolveIt Software*. Version 6.* pine-grove.com/SolveIT/, 2013.

SOUZA JUNIOR, S. J. **Framework para sistemas de controle - Definição a partir do domínio de defesa civil**. 2002, 138f. Dissertação (Mestrado em Sistemas e Computação) – Instituto Militar de Engenharia, Rio de Janeiro, 2002.

TALIGENT. **Building object-oriented frameworks**, Taligent Inc. white paper, 1994.

YASSIN, A., FAYAD, M. E. Application frameworks: A survey. In: FAYAD, M. E., JOHNSON, R. E. **Domain-Specific Application Frameworks:** Frameworks Experience by Industry. New York: John Wiley& Sons, 2000. Cap. 29 p. 615-632.

WERNKE, R. **Análise de Custos e Preços de Venda: Ênfase em aplicações e casos nacionais**. 1ª ed., São Paulo: Editora Saraiva, 2005.

Cálculo do preço de venda. Disponível em <http://www.sebraepr.com.br>. Acessado em: 20 jul.de 2013.

APÊNDICE A - MODELAGEM DO SUBSISTEMA *PRODUCT*

Este subsistema do método ABC tem relacionamento com a linha de produção, pois no momento de inserir um novo produto é necessário conhecer a sua linha de produção.

Assim, como mencionado na inserção do subsistema *ProductionLine,* a opção de Produto no sitio do FrameMK também se encontrava desabilitada conforme a Figura 10. Logo, a opção estará disponível após a adição deste novo subsistema.

A identificação de um ponto de junção foi realizada após análise do código fonte do FrameMK, sua documentação e análise feita no BD para entender melhor a estrutura do subsistema *Product*, com isso, identificou-se a existência da tabela Produto já criada.

O subsistema *Product* é responsável por exibir todos os produtos cadastrados, podendo o usuário efetuar as funções de cadastrar, editar ou desabilitar um novo produto.

A Figura 48 ilustra o modelo criado por Crazuski et al. (2008) que exemplifica este subsistema com suas funcionalidades.

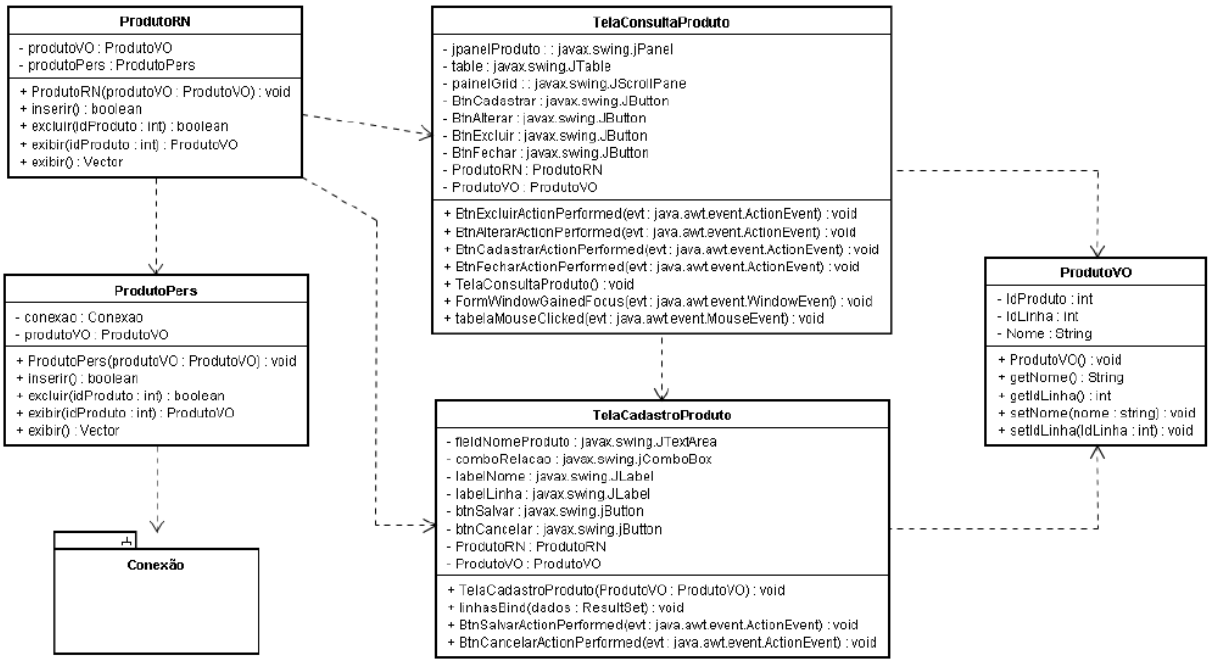

Figura 48 - Diagrama de Classes do subsistema "Gerenciar Produto" do método ABC. Fonte: Crazuski et al. (2008) p. 95.

Em seguida, foi analisado os códigos do FrameMK e visto o que seria necessário para inserir o novo subsistema. Primeiramente, foi adicionado um *link* para *Product* no JSP *windowsMenuAbc.*jsp, a Figura 49 ilustra esta adição.

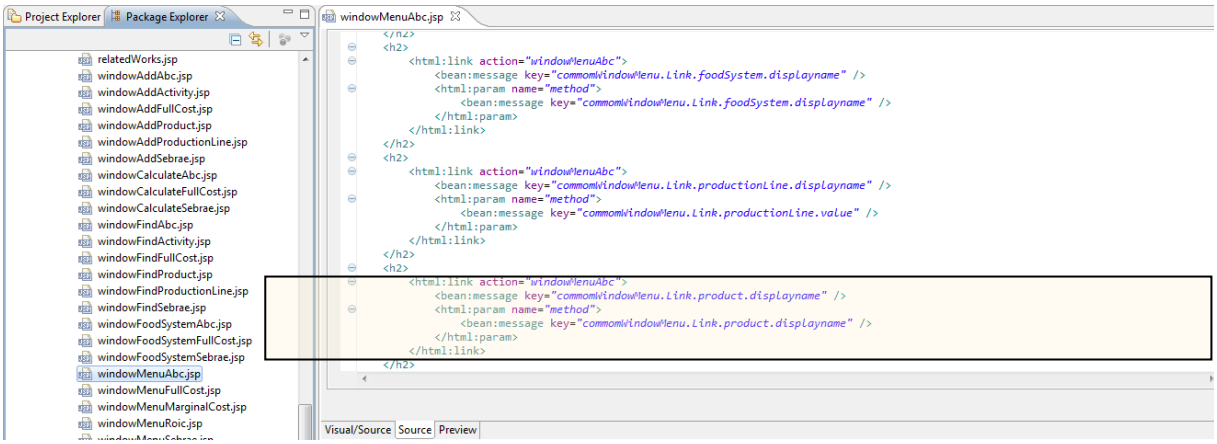

Figura 49 - Adição de *Link* para *Product* no JSP *windowMenuABC.jsp.*
Fonte: Autoria Própria.

Outra tarefa realizada, ilustrada na Figura 50, foi a inserção de *forward* para *action windowFindProduct* no arquivo *Struts-Config*. Este arquivo recebe o evento de *click* e chama a classe *Action* vinculada ao *Path*.

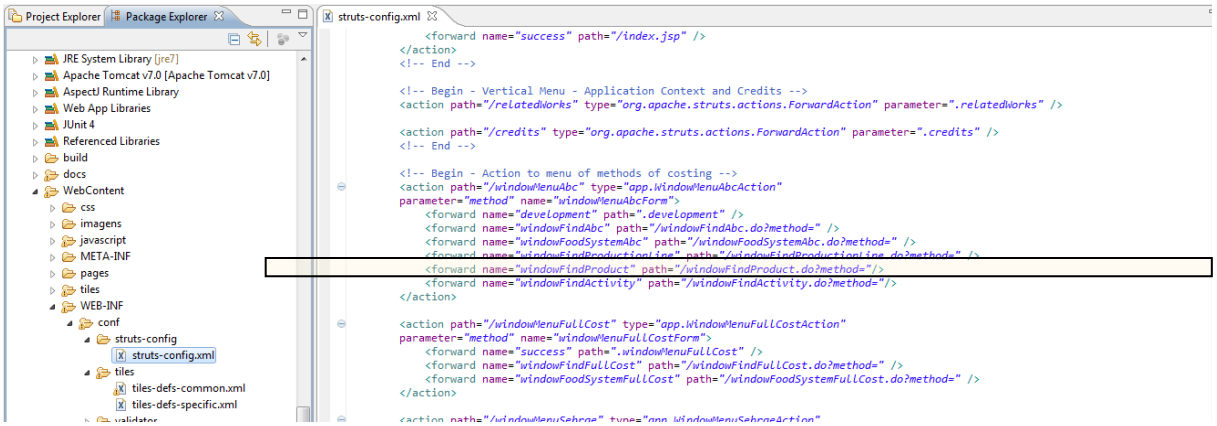

Figura 50 – Adição do *forward* para *windowFindProduct* sem método especificado.
Fonte: Autoria Própria.

Logo após, foi criado um método para o Menu Produto na classe *WindowMenuAbcAction*, que chama o método correspondente ao *click* no *jsp* e faz uma chamada conforme a Figura 51, ao *Forward* definido na Figura 50**Erro! Fonte de referência não encontrada.**.

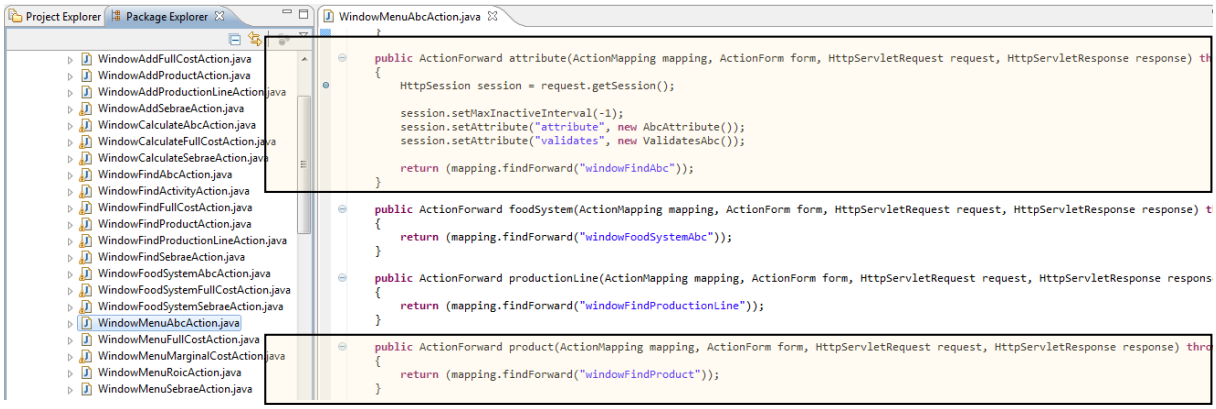

Figura 51 - Criação do método para receber requisições sem especificação de método.
Fonte: Autoria Própria.

Outra tarefa executada, ilustrada na Figura 52, foi a criação do *forward* que redireciona para classe *WindowFindProductAction*, não especificando o método a ser chamado, sendo assim, existe um método *unspecified* que é chamado para esta situação. E posteriormente, este método é sobrescrito.

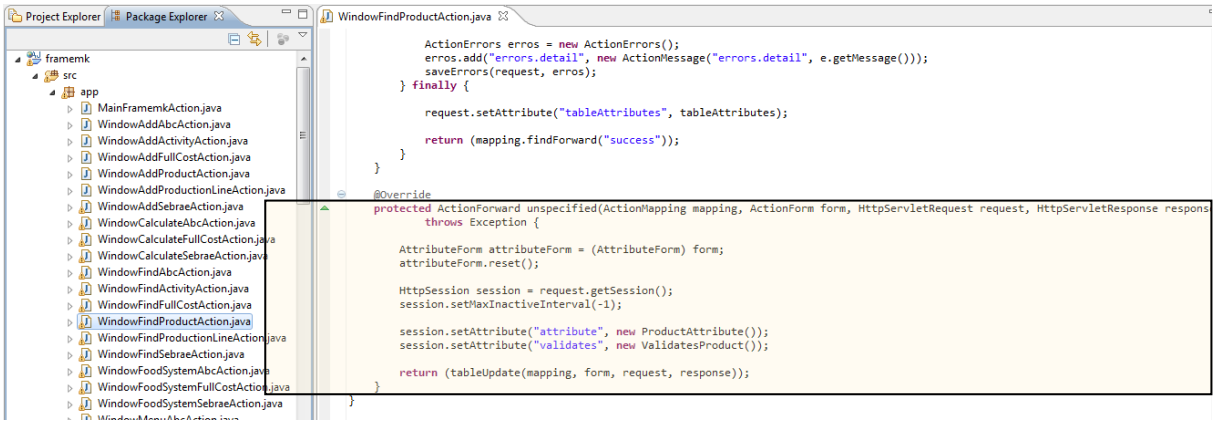

Figura 52 – Redirecionamento do *Forward* para a classe *WindowFindProductAction.*
Fonte: Autoria Própria.

A Figura 53 ilustra a adição de um *Tag Action* para *WindowFindProduct* no arquivo *struts-config.xml*, que na chamada do *foward "success" é requisitado para windowFindProduct.*

Figura 53 - Adição de uma *Tag Action* para *WindowFindProduct.*
Fonte: Autoria Própria.

Em seguida, na Figura 54 é ilustrada a criação da classe *windowFindProduct.jsp* para criar o menu de produto.

Figura 54 - Criação da classe *windowFindProduct.jsp.*
Fonte: Autoria Própria.

A Figura 55 apresenta a adição das *tags definition* para .*windowFindProduct* e .*specificWindowProduct* no arquivo *tiles-defs-specific*.xml que aponta para *windowFindProduct*.jsp mostrando a herança com *commonWindowFind*.jsp

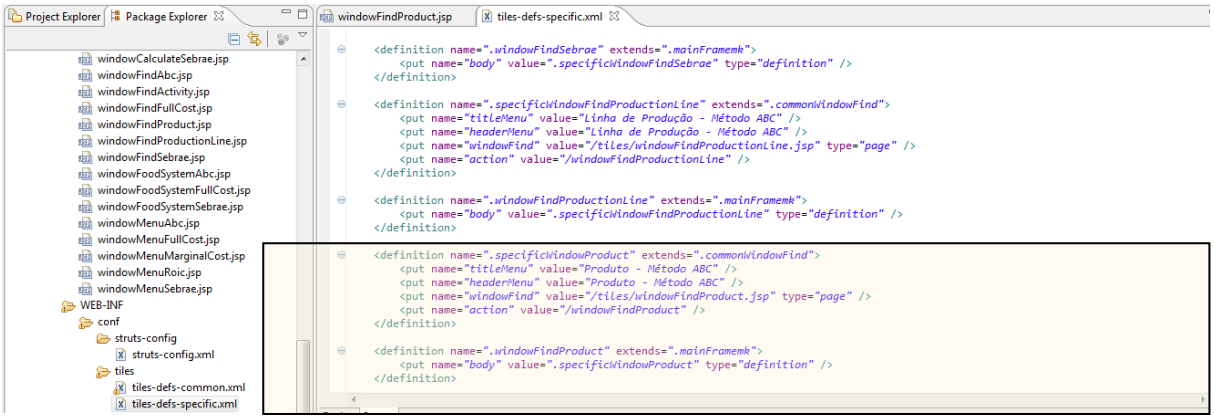

Figura 55 - Definição da classe *.windowFindProduct* no arquivo *tiles-defs-specific*.xml.
Fonte: Autoria Própria.

A tarefa seguinte, ilustrada na Figura 56, foi a adição da *Tag Action* para *WindowAddProduct* para possível *click* em opção de adicionar e/ou editar Produtos em *WindowFindProduct*.jsp. Quando alguma das opções é acionada por meio de um *click* o *forward windowAddProduct* da *Tag*, ilustrada na Figura 53, é disparado e chama a *Action* de adição de produto.

Figura 56 - Adição da *Tag Action* para *WindowAddProduct*.
Fonte: Autoria Própria.

A Figura 57,Figura 58 mostra que a classe *WindowAddProductAction* também contém um método *unspecified* que simplesmente chama *forward success* para abrir a tela de adição e/ou edição.

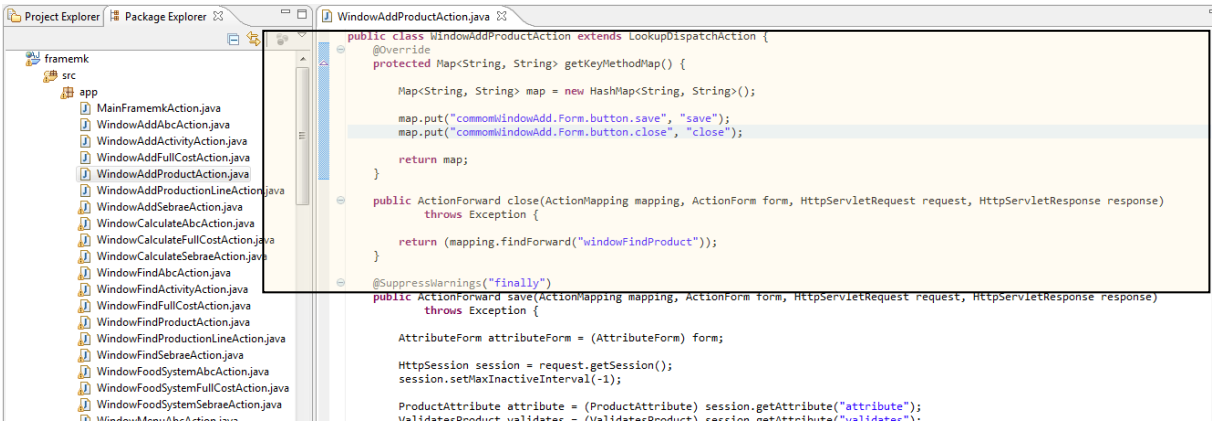

Figura 57 - Método para chamar *forward success* e abrir tela adição e/ou edição.
Fonte: Autoria Própria.

Em seguida, foi criado a classe *WindowAddProduct*.jsp para as opções de adição e/ou edição de produtos. A Figura 58 mostra essa tarefa. Este *jsp* se apresenta quase vazio, pois as únicas *tags* necessárias para a adição de produto, são as mesmas do *jsp* base, a saber, *WindowCommonAdd.jsp.*

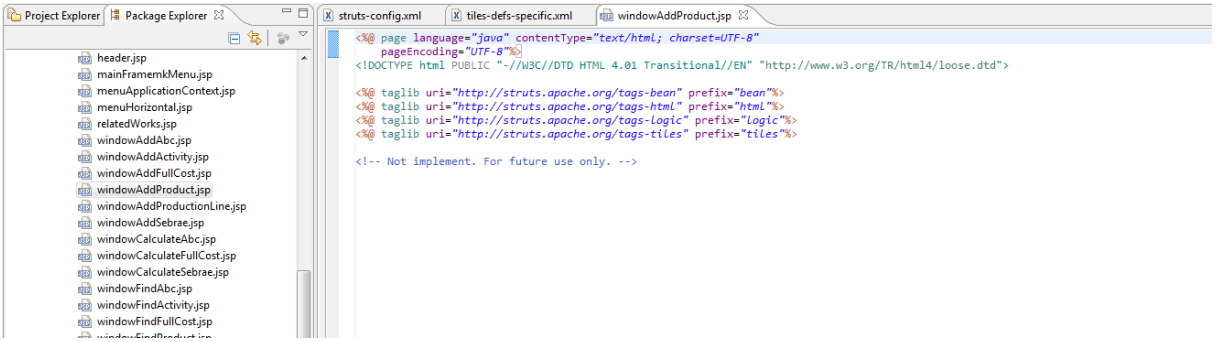

Figura 58 - Criação da classe *WindowAddProduct*.jsp para adição e/ou edição de produtos.
Fonte: Autoria Própria.

A Figura 59 ilustra a definição das *tags definition para windowAddProduct e .specificWindowAddProduct*, criadas no arquivo *tiles-defs-specific.*xml que apontam para *windowAddProduct*.jsp que herda partes de *commonWindowAdd.jsp*.

Figura 59 – Definição criada no arquivo *tiles-defs-specific.*xml da classe *WindowFindProduct.*
Fonte: Autoria Própria.

APÊNDICE B - MODELAGEM DO SUBSISTEMA *ACTIVITY*

O subsistema *Activity* equivale ao Gerenciar Atividade do Método ABC. Ao analisar o *Activity* identificou-se que o mesmo é composto de vários atributos tais como: Nome da Atividade, Linha de Produção e Custo Total. Tais atributos são necessários para calcular o preço de venda.

Posteriormente a análise sobre o subsistema, verificou-se que há dependências por parte do subsistema *Activity* como, por exemplo, a necessidade de ter um novo nome para a atividade, também se relaciona com atributo e com linha de produção.

Assim, como mencionado na inserção do subsistema *ProductionLine e Product,* a opção de Atividade no sitio do FrameMK também se encontra desabilitada conforme a Figura 10. A Figura 60 ilustra o subsistema *Activity* através do diagrama de classes.

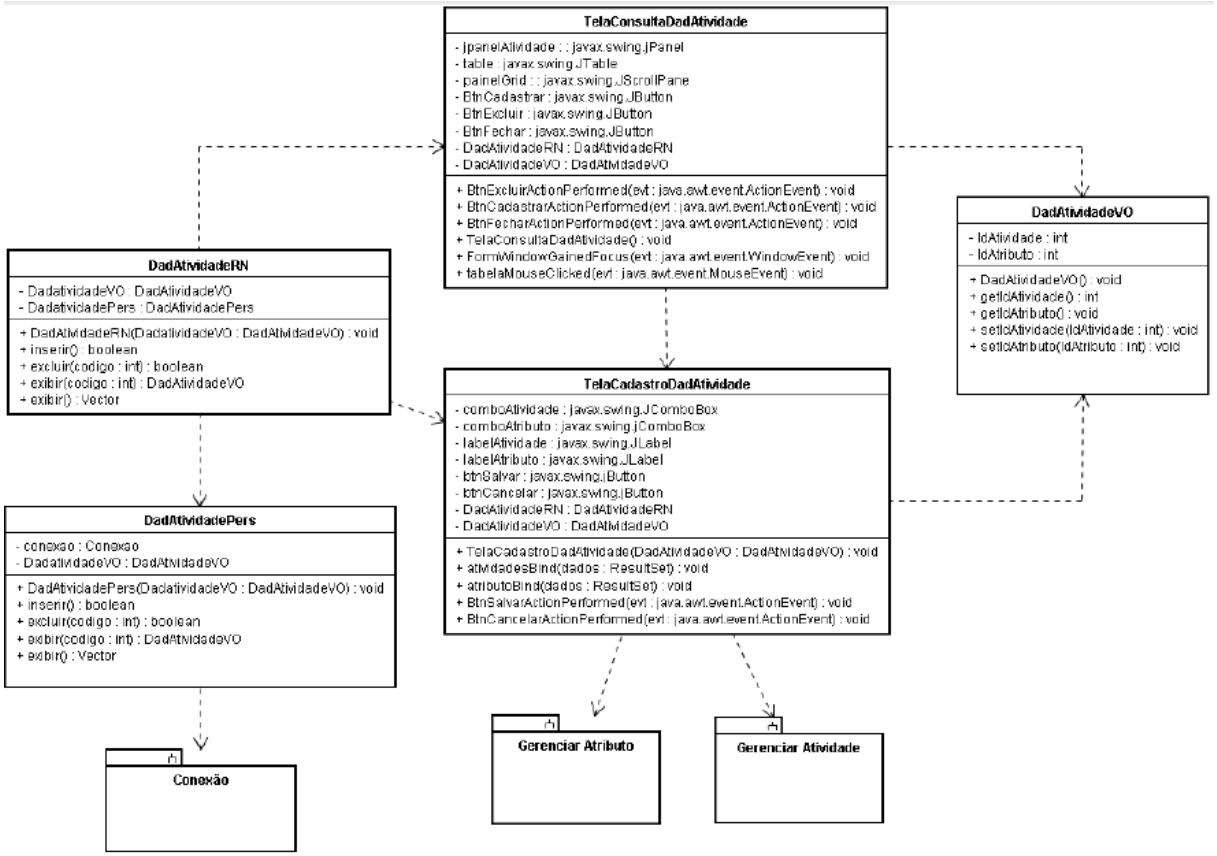

Figura 60 - Diagrama de Classes do subsistema "Gerenciar Atividade".
Fonte: Crazuski et al. (2008, p. 116).

A tarefa seguinte foi analisar os códigos e visualizar o que seria necessário para inserir os novos requisitos deste subsistema. Primeiramente, foi adicionado um *link* para *Activity* no JSP *windowsMenuAbc.*jsp. A Figura 61 ilustra esta alteração.

Figura 61 - Adição de *Link* para *Activity* no JSP *WindowMenuABC.jsp.*
Fonte: Autoria Própria.

A tarefa seguinte foi inserir o *forward* para *action windowFindActivity* no arquivo *Struts-Config.xml*. Este arquivo recebe o evento *click* e chama a classe *Action* vinculada ao *Path*. A Figura 62, exemplifica a tarefa realizada.

Figura 62 - Inserção do *forward* para *windowFindActivity* sem método especificado.
Fonte: Autoria Própria.

Foi criado também um método para o Menu Produto na classe *WindowMenuAbcAction*, que chama método correspondente ao *click* no *jsp* e faz uma chamada conforme mostra a Figura 63.

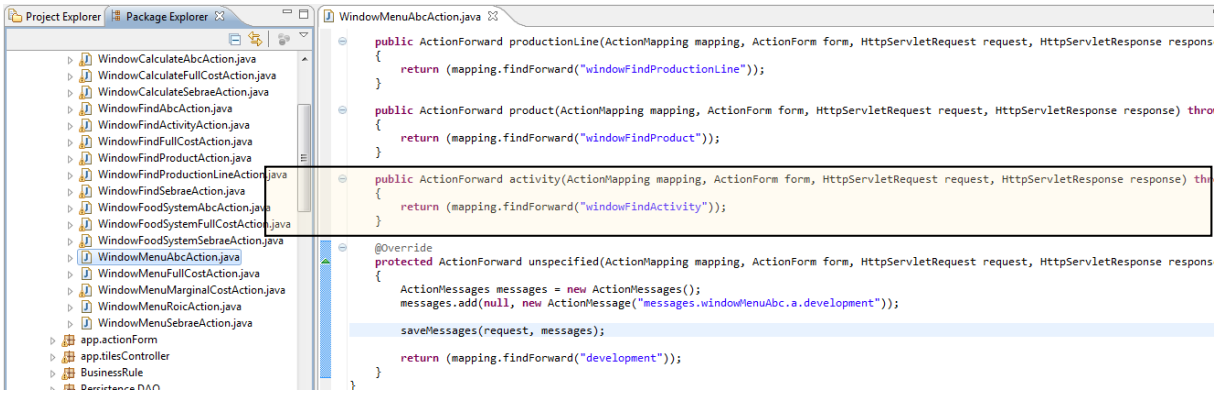

Figura 63 - Adicionando *forward* para *windowFindProduct* sem método especificado.
Fonte: Autoria Própria.

Posteriormente, realizou-se a chamada do *forward* que redireciona para classe *WindowFindProductAction*, não especificando o método a ser chamado, sendo assim, existe um método *unspecified* que é chamado para esta situação. E posteriormente, este método é sobrescrito. Este método é exemplificado na Figura 64**Erro! Fonte de referência não encontrada.**.

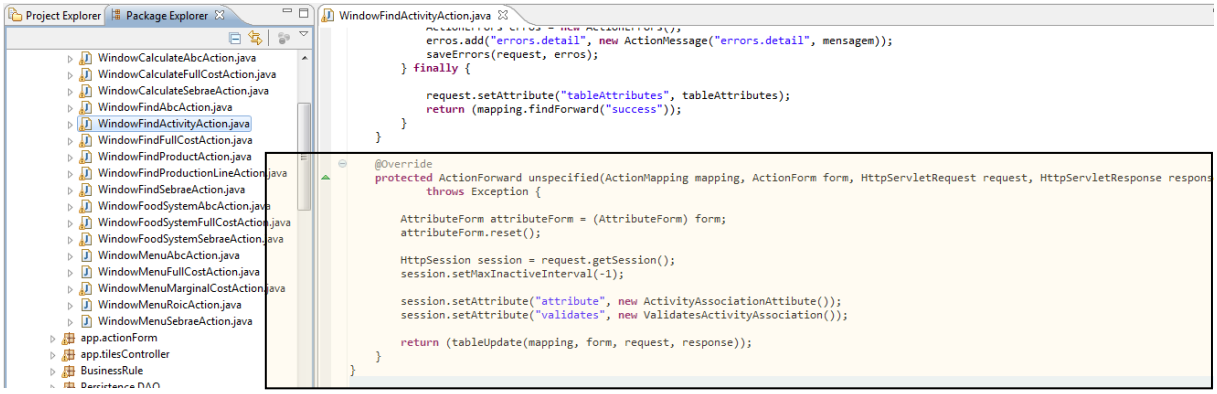

Figura 64 - Redirecionamento do *Forward* para a classe *WindowsFindActivityAction.*
Fonte: Autoria Própria.

A Figura 65 mostra a adição de um *Tag Action* para *WindowFindActivity* no arquivo *struts-config.xml*, que na chamada do *foward success* é chamado por *.windowFindActivity.*

Figura 65 - Adição de uma *Tag Action* para *WindowFindActivity.*
Fonte: Autoria Própria.

A Figura 66, apresenta a adição das *tags definition* para .*windowFindActivity* no arquivo *tiles-defs-specific*.*xml* que aponta para *windowFindActivity*.jsp mostrando a herança com *commonWindowFind*.jsp.

Figura 66 - Definição da classe *.windowFindActivity* no arquivo *tiles-defs-specific.xml.*
Fonte: Autoria Própria.

A tarefa seguinte, mostrada na Figura 67, foi a adição da *Tag Action* para *WindowAddActivity* para possível *click* em opção de adicionar ou editar Atividade em *WindowFindActivity*.jsp. Quando alguma das opções é acionada por meio de um *click*, o *forward windowAddActivity* da *Tag* ilustrada na Figura 65 é disparado e chama a *Action* de adição de produto.

Figura 67 – Adição da *Tag Action* para *WindowAddActivity.*
Fonte: Autoria Própria.

A Figura 68 exibe a classe *WindowAddActivitytAction* que também contém um método *unspecified* o qual chama *forward success* para abrir a tela de adição e/ou edição.

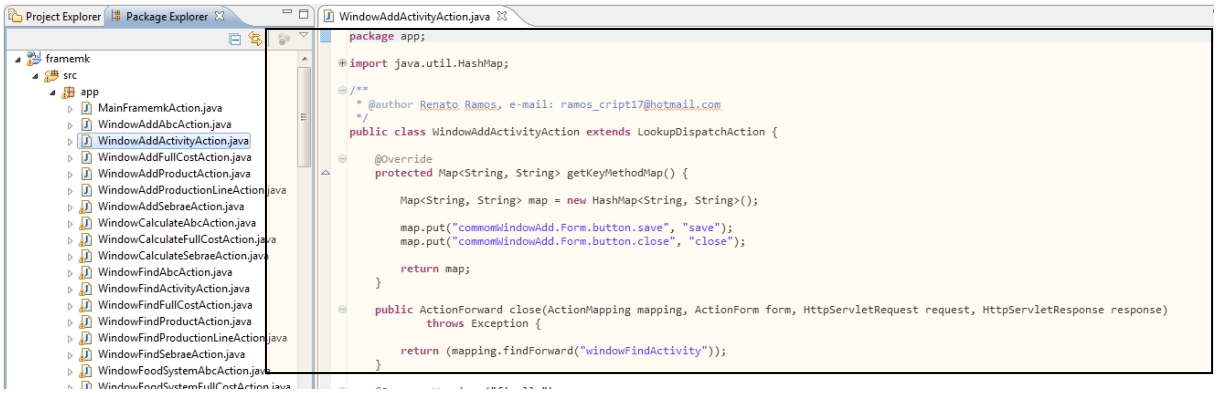

Figura 68 - Método para chamar *forward success* e abrir tela adição e/ou edição.
Fonte: Autoria Própria.

A Figura 69 ilustra a definição da *tag definition para windowAddActivity*, criadas no arquivo *tiles-defs-specific.*xml que aponta para *windowAddActivity*.jsp e herda partes de *commonWindowAdd*.jsp

Figura 69 – Definição criada no arquivo *tiles-defs-specific.*xml da classe *WindowFindActivity.* Fonte: Autoria Própria.