

ANDRÉ KENJI HORIE
GABRIEL ISEPPE PORTO
RODRIGO AUGUSTO AZEVEDO PELEGRINI PEREZ

SISTEMA DE PRECIFICAÇÃO E CONTROLE DE
RISCO DE CARTEIRA DE ATIVOS
(SPCRCA)

São Paulo
2008

ANDRÉ KENJI HORIE
GABRIEL ISEPPE PORTO
RODRIGO AUGUSTO AZEVEDO PELEGRINI PEREZ

SISTEMA DE PRECIFICAÇÃO E CONTROLE DE RISCO DE CARTEIRA DE ATIVOS (SPCRCA)

Trabalho de conclusão de curso apresentado
à Escola Politécnica da Universidade de São
Paulo para obtenção do título de Engenheiro
de Computação.

São Paulo
2008

ANDRÉ KENJI HORIE
GABRIEL ISEPPE PORTO
RODRIGO AUGUSTO AZEVEDO PELEGRINI PEREZ

SISTEMA DE PRECIFICAÇÃO E CONTROLE DE RISCO DE CARTEIRA DE ATIVOS (SPCRCA)

Trabalho de conclusão de curso apresentado
à Escola Politécnica da Universidade de São
Paulo para obtenção do título de Engenheiro
de Computação.

Área de Concentração:
Engenharia de Computação

Orientadora:
Prof^a. Dr^a. Selma Shin Shimizu Melnikoff

Co-orientador:
Sr. Arnaud Seydoux

São Paulo
2008

Resumo

Este trabalho apresenta o processo de desenvolvimento do Sistema de Precificação e Controle de Risco de Carteira de Ativos (SPCRCA). Este sistema visa ser uma ferramenta para auxiliar os gestores de fundos de investimentos na avaliação da viabilidade de um ativo ou grupo de ativos, através de análises estratégicas acerca de informações históricas de variáveis relevantes e de simulações de parâmetros de mercado. Desta forma, é possível a modelagem de ativos e avaliação quanto ao seu valor, risco ou outros indicadores. São aplicados diversos conceitos da área de finanças para as funções de negócios do sistema, assim como as disciplinas de Engenharia de Software e Gerência de Projetos no processo de desenvolvimento, de modo a garantir a entrega de um produto final de qualidade e que apresente uma arquitetura robusta, e para a qual requisitos como por exemplo configurabilidade, escalabilidade, modularidade e processamento em tempo real sejam respeitados.

Palavras-chave: Ativos, Análises Financeiras, Controle de Risco, Engenharia de Software, Arquitetura de Software Orientada a Padrões

Abstract

This work presents the development process for the System of Pricing and Risk Control of a Portfolio (SPCRCA, in Portuguese). This system aims to be a tool for helping investment fund managers in the evaluation of the feasibility of an asset or basket of assets through strategical analysis of historical information of relevant variables and simulation of market parameters. As a result, it enables modeling of financial assets and evaluation of their value, risk or other indicators. A variety of concepts of Finances is applied to the business functions of the system, as well as the disciplines of Software Engineering and Project Management to the development process, in order to guarantee a final product of quality which presents a robust architecture and whose requirements such as configurability, scalability, modularity and real-time processing are respected.

Keywords: Equities, Financial Analysis, Risk Control, Software Engineering, Pattern-Oriented Software Architecture

Dedicatória

Dedico este trabalho à minha mãe Satie, por me ensinar perseverança e força de vontade, e acima de tudo, por ter me ajudado a ser tudo o que eu atualmente sou. Aos meus irmãos Mauro e Leonardo, por todo o apoio incondicional. Ao meu pai Yuji, por me inspirar a fazer meu melhor para alcançar meus objetivos de vida. Dedico, por fim, à minha família e aos meus amigos, em especial aos meus grandes amigos Shindi, Vanessa e Simone, pelo companheirismo e momentos compartilhados.

André Kenji Horie

Dedico este trabalho à minha mãe Yara, cujos valores, amor e carinho me ajudaram a ser o que sou. Ao meu pai Carlos, por me apoiar e sempre acreditar em mim. À minha irmã Tatiana, pelas risadas, carinho e apoio. Aos meus avós, por me mostrarem a riqueza do saber e importância de ser uma pessoa correta. Dedico, por fim, à minha namorada e companheira Flávia, pelo seu amor, apoio e paciência ao longo destes sete anos.

Gabriel Iseppe Porto

Dedico este trabalho aos meus pais, Mariza e Wanderley, pela orientação, pela compreensão e pelo apoio em todas as escolhas que fiz no percurso de me tornar o que sou. Aos meus irmãos, Thiago e Renato, pelos bons momentos de nossa infância e pelos que ainda estão por vir. À minha namorada Malu, meu primeiro e eterno amor, pela inspiração nos momentos que precisei e pelos sonhos que compartilhamos. Dedico, por fim, ao Henrique, Thiago e Vander, pela amizade que sobreviverá/transcenderá a convivência na república.

Rodrigo Augusto Azevedo Pelegrini Perez

Agradecimentos

Agradecemos primeiramente a Deus, por tudo.

À professora Selma Shin Shimizu Melnikoff, pela orientação, atenção e apoio, indispensáveis para este Projeto de Formatura.

Ao Arnaud Seydoux, pelo papel fundamental no conhecimento transmitido quanto ao mercado financeiro e à arquitetura de sistemas.

À Escola Politécnica da Universidade de São Paulo, pela excelência na formação em Engenharia, provendo-nos oportunidade de aprendizado e crescimento.

Ao Departamento de Engenharia de Computação e Sistemas Digitais, por prover a seus alunos não apenas com conhecimento técnico, mas também com a capacitação esperada de um Engenheiro de Computação.

À Vector Investimentos, por apresentar a necessidade para este projeto e dispor ao grupo ambiente para o desenvolvimento do trabalho.

A nossos colegas, pela amizade e cooperatividade.

Por fim, a todos que direta ou indiretamente contribuíram para este trabalho.

Lista de Figuras

1	Camadas da Engenharia de Software	p. 36
2	Fases do Processo Unificado	p. 37
3	Matriz de exposição aos riscos	p. 54
4	Diagrama de navegação	p. 68
5	Diagrama do modelo de casos de uso	p. 69
6	Visão lógica da arquitetura: Blocos <i>Application</i> , <i>Business</i> e <i>Framework</i> . .	p. 71
7	Visão física da arquitetura: Estrutura do PAC em tempo de execução . . .	p. 72
8	Diagrama de classes das interfaces relacionadas ao uso de <i>factories</i> . . .	p. 73
9	Diagrama de sequência da criação de objetos utilizando <i>factories</i>	p. 74
10	Diagrama de classes do <i>manager</i>	p. 75
11	Diagrama de classes relacionado com o acesso a dados	p. 78
12	Diagrama de classes para a segurança de acesso	p. 79
13	Modelo do banco de dados para a segurança de acesso	p. 80
14	Diagrama de sequência da autenticação e do acesso a um recurso	p. 81
15	Diagrama de classes das interfaces e enumerações necessárias à imple- mentação de <i>securities</i>	p. 82
16	Diagrama de classes de <i>trades</i>	p. 83
17	Diagrama de classes envolvendo criação de <i>securities</i> e <i>trades</i> de <i>stocks</i> .	p. 83
18	Diagrama de sequência envolvendo criação de <i>securities</i> e <i>trades</i> de <i>stocks</i>	p. 84
19	Diagrama de sequência do cálculo do risco	p. 85
20	Exemplo de gráfico utilizando o componente <i>Drawing</i>	p. 93
21	Implementação de áreas de trabalho	p. 96

22	Tela final do Módulo Principal	p. 101
23	Tela final do módulo <i>Book View</i>	p. 102
24	Tela final do módulo <i>Trade Builder</i>	p. 103
25	Tela final do módulo <i>Security Builder</i>	p. 103
26	Tela final do módulo <i>Market Pricing and Risk</i>	p. 104
27	Tela final do módulo <i>Historic Pricing</i>	p. 104
28	Tela final do módulo <i>Market Scenario</i>	p. 105
29	Tela final do módulo <i>Cash Flows</i>	p. 105
30	Protótipo de tela do Módulo Principal	p. 116
31	Protótipo de tela do <i>Book View</i>	p. 117
32	Protótipo de tela do <i>Trade Builder</i>	p. 117
33	Protótipo de tela do Pricing	p. 118
34	Protótipo de tela do <i>Market Risk</i>	p. 118
35	Protótipo de tela do <i>Historic Pricing</i>	p. 119
36	Protótipo de tela do <i>Market Scenario</i>	p. 119
37	Protótipo de tela do <i>Cash Flows</i>	p. 119
38	Diagrama entidade-relacionamento do <i>schema</i> SPCRCA	p. 121
39	Diagrama entidade-relacionamento do <i>schema</i> Application	p. 121
40	Diagrama entidade-relacionamento do <i>schema</i> Personal	p. 122
41	Diagrama entidade-relacionamento do <i>schema</i> Daily	p. 122
42	Diagrama entidade-relacionamento do <i>schema</i> Framework	p. 123
43	Representação dos fluxos gerados por um título de renda fixa	p. 129

Lista de Tabelas

1	Catálogo de <i>Design Patterns</i>	p. 42
2	Distribuição de papéis entre os recursos humanos	p. 51
3	Matriz de resposta aos riscos	p. 54
4	Regras para nomes e <i>tags</i> em <i>commands</i>	p. 77
5	Estimativa das Atividades e Cronograma	p. 115

Lista de Abreviaturas e Siglas

ACM *Association for Computing Machinery*

BOVESPA Bolsa de Valores de São Paulo

CVM Comissão de Valores Mobiliários

IEEE *Institute of Electrical and Electronics Engineers*

Ibovespa Índice Bovespa

IHC Interface Homem-Computador

MVC *Model-View-Controller*

NDF *Non Deliverable Forward*

OHLC *Open High Low Close*

OMG *Object Management Group*

OO Orientação a Objetos

PAC *Presentation-Abstraction-Control*

PMI *Project Management Institute*

POSA Arquitetura de Software Orientada a Padrões

RUP *Rational Unified Process*

SEI *Software Engineering Institute*

SQL *Structured Query Language*

UML *Unified Modeling Language*

WBS *Work Breakdown Structure*

Sumário

1	Introdução	p. 18
1.1	Objetivo do Trabalho	p. 19
1.2	Motivação	p. 20
1.3	Justificativa	p. 21
1.4	Estrutura do Trabalho	p. 21
2	Contexto do SPCRCA	p. 23
2.1	Introdução	p. 24
2.2	Mercado Financeiro	p. 24
2.2.1	Títulos e Valores Mobiliários	p. 26
2.2.1.1	Ações	p. 26
2.2.1.2	Derivativos	p. 27
2.2.1.3	Títulos de Dívida Pública	p. 30
2.2.1.4	Títulos de Dívida Privada	p. 30
2.2.2	Fundos de Investimento	p. 30
2.2.3	Tipos de Análises	p. 31
2.2.3.1	Market Pricing	p. 31
2.2.3.2	Market Risk	p. 32
2.2.3.3	Market Scenario	p. 32
2.2.3.4	Historical Pricing	p. 32
2.2.3.5	Profit and Loss Explain	p. 32
2.2.3.6	Cash Flow	p. 32

2.3	Perspectivas do Produto	p. 33
2.4	Unidades Organizacionais Envolvidas	p. 33
2.5	Conceitos Teóricos	p. 33
2.5.1	Gerenciamento de Projeto	p. 33
2.5.1.1	Gerenciamento de Escopo	p. 34
2.5.1.2	Gerenciamento de Tempo	p. 34
2.5.1.3	Gerenciamento de Custo	p. 35
2.5.1.4	Gerenciamento de Risco	p. 35
2.5.2	Engenharia de Software	p. 36
2.5.2.1	Processo Unificado	p. 36
2.5.2.2	Unified Modeling Language	p. 38
2.5.2.3	Engenharia de Requisitos	p. 38
2.5.3	Arquitetura de Software	p. 39
2.5.3.1	Conceitos de Arquitetura de Software	p. 39
2.5.3.2	Princípios Fundamentais	p. 40
2.5.4	Design Patterns e Architectural Patterns	p. 41
2.5.4.1	Design Patterns	p. 41
2.5.4.2	Architectural Patterns	p. 43
2.6	Considerações Finais	p. 44
3	Planejamento	p. 45
3.1	Introdução	p. 46
3.2	Metodologia de Projeto	p. 46
3.3	Gerenciamento de Escopo	p. 47
3.3.1	Escopo da Solução	p. 47
3.3.2	Conjunto de Tarefas	p. 48
3.3.2.1	Work Breakdown Structure	p. 48

3.4	Gerenciamento de Tempo e de Custo	p. 50
3.4.1	Estimativas	p. 50
3.4.1.1	Técnica Top-Down	p. 50
3.4.1.2	Técnica Bottom-Up	p. 50
3.4.1.3	Resultados Consolidados	p. 50
3.4.2	Recursos do Projeto	p. 51
3.4.2.1	Recursos Humanos	p. 51
3.4.2.2	Recursos de Software	p. 51
3.4.2.3	Recursos de Hardware	p. 51
3.5	Gerenciamento de Risco	p. 52
3.5.1	Levantamento dos Riscos	p. 52
3.5.2	Análise de Criticidade e Estratégia de Resposta	p. 53
3.6	Considerações Finais	p. 55
4	Concepção	p. 56
4.1	Introdução	p. 57
4.2	Requisitos Funcionais	p. 57
4.2.1	Funções de Negócios	p. 57
4.2.1.1	Market Pricing and Risk	p. 57
4.2.1.2	Market Scenario	p. 57
4.2.1.3	Historical Pricing	p. 58
4.2.1.4	Cash Flows	p. 58
4.2.2	Funções de Infra-Estrutura	p. 58
4.2.2.1	Títulos e Valores Mobiliários	p. 58
4.2.2.2	Trade Builder e Security Builder	p. 59
4.2.2.3	Book View	p. 59
4.2.2.4	Autenticação	p. 60

4.3	Requisitos Não-Funcionais	p. 60
4.3.1	Qualidade de Processo	p. 60
4.3.2	Configurabilidade da Interface	p. 61
4.3.3	Desempenho	p. 61
4.3.4	Disponibilidade	p. 61
4.3.5	Escalabilidade	p. 62
4.3.6	Manutenabilidade	p. 62
4.3.7	Segurança de Acesso	p. 62
4.3.8	Testabilidade	p. 62
4.3.9	Usabilidade	p. 62
4.4	Características dos Usuários	p. 63
4.5	Interfaces	p. 63
4.5.1	Interface com o Usuário	p. 63
4.5.2	Interfaces de Comunicação	p. 63
4.6	Restrições	p. 63
4.7	Dependências	p. 64
4.8	Considerações Finais	p. 64
5	Elaboração	p. 65
5.1	Introdução	p. 66
5.2	Interface Homem-Computador	p. 66
5.2.1	Caracterização dos Usuários	p. 66
5.2.1.1	Structurer	p. 66
5.2.1.2	Trader	p. 66
5.2.1.3	Product Control	p. 67
5.2.2	Estilo da Interface	p. 67
5.2.3	Protótipo de Navegação	p. 67

5.2.4	Protótipo de Telas	p. 68
5.3	Modelo de Casos de Uso	p. 68
5.4	Arquitetura do Sistema	p. 70
5.5	Modelagem do Sistema	p. 72
5.5.1	Framework	p. 72
5.5.1.1	Factories	p. 73
5.5.1.2	Patterns	p. 76
5.5.1.3	PAC, Command e Command Processor	p. 76
5.5.1.4	Acesso a Dados	p. 77
5.5.1.5	Segurança de Acesso	p. 79
5.5.2	Business	p. 81
5.5.2.1	Securities e Trades	p. 82
5.5.2.2	Risk	p. 84
5.5.2.3	Servidor Tempo Real	p. 85
5.5.3	Application	p. 86
5.5.4	Considerações sobre Requisitos Não-Funcionais	p. 87
5.6	Modelo de Dados	p. 87
5.7	Critérios de Aceitação	p. 88
5.7.1	Mapeamento dos Requisitos	p. 88
5.7.2	Testes	p. 88
5.7.3	Aceitação	p. 88
5.8	Considerações Finais	p. 88
6	Construção	p. 90
6.1	Introdução	p. 91
6.2	Protótipo Mínimo	p. 91
6.3	Visão Geral da Construção	p. 92

6.4	Detalhamento da Construção	p. 92
6.4.1	Framework	p. 92
6.4.1.1	Arquitetura Message Queuing	p. 92
6.4.1.2	Drawing	p. 93
6.4.2	Business	p. 94
6.4.2.1	Simulação de Servidor Tempo Real	p. 94
6.4.3	Application	p. 95
6.4.3.1	Módulos Funcionais	p. 95
6.4.3.2	Integração entre Módulos	p. 96
6.5	Considerações Finais	p. 97
7	Transição	p. 98
7.1	Introdução	p. 99
7.2	Testes	p. 99
7.2.1	Testes Unitários	p. 99
7.2.2	Testes Modulares	p. 99
7.2.3	Testes de Integração	p. 99
7.2.4	Testes de Requisitos Não-Funcionais	p. 100
7.3	Sistema Final	p. 101
7.3.1	Módulo Principal	p. 101
7.3.2	Book View	p. 101
7.3.3	Trade Builder e Security Builder	p. 101
7.3.4	Market Pricing and Risk	p. 102
7.3.5	Historic Pricing	p. 103
7.3.6	Market Scenario	p. 104
7.3.7	Cash Flows	p. 104
7.4	Aceitação do Sistema	p. 105

7.5	Considerações Finais	p. 106
8	Considerações Finais	p. 107
8.1	Conclusões	p. 108
8.2	Contribuições	p. 109
8.3	Trabalhos Futuros	p. 109
	Referências	p. 111
	Apêndice A – Planejamento de Atividades	p. 114
A.1	Visão Geral	p. 114
A.2	Estimativa das Atividades e Cronograma	p. 114
	Apêndice B – Protótipo de Telas	p. 116
B.1	Visão Geral	p. 116
B.2	Telas	p. 116
B.2.1	Módulo Principal	p. 116
B.2.2	Book View	p. 116
B.2.3	Trade Builder	p. 117
B.2.4	Market Pricing	p. 117
B.2.5	Market Risk	p. 118
B.2.6	Historic Pricing	p. 118
B.2.7	Market Scenario	p. 118
B.2.8	Cash Flows	p. 119
	Apêndice C – Modelo de Dados	p. 120
C.1	Visão Geral	p. 120
C.2	Estrutura	p. 120
C.2.1	Schema SPCRCA	p. 120

C.2.2	Schema Application	p. 120
C.2.3	Schema Personal	p. 121
C.2.4	Schema Daily	p. 122
C.2.5	Schema Framework	p. 122

Apêndice D – Métodos de Precificação e Risco p. 124

D.1	Visão Geral	p. 124
D.2	Métodos de Precificação	p. 124
D.2.1	Ações	p. 124
D.2.2	Contratos Futuros	p. 125
D.2.2.1	Futuro de Índice Bovespa	p. 125
D.2.2.2	Futuro de Taxa Média DI	p. 126
D.2.2.3	Futuro de Taxa de Câmbio - Reais por Dólar	p. 127
D.2.2.4	Futuro de Cupom Cambial	p. 127
D.2.3	Títulos de Renda Fixa	p. 128
D.2.4	Opções	p. 129
D.3	Método de Análise de Risco	p. 130

1 *Introdução*

“A verdadeira diferença entre a construção e a criação é esta: uma coisa construída só pode ser amada depois de construída, mas uma coisa criada ama-se mesmo antes de existir.”

Charles Dickens

1.1 Objetivo do Trabalho

Este trabalho apresenta o processo de desenvolvimento do Sistema de Precificação e Controle de Risco de Carteira de Ativos (SPCRCA). Este sistema visa ser uma ferramenta para auxiliar os gestores de fundos de investimentos na avaliação da viabilidade de um ativo (bens e direitos resultante de transações das quais futuros benefícios são esperados) ou grupo de ativos, através de análises estratégicas acerca de informações históricas de variáveis relevantes e de simulações de parâmetros de mercado. Desta forma, é possível a modelagem de ativos e avaliação quanto ao seu valor, risco ou outros indicadores.

Tanto os processos de modelagem quanto os de avaliação dos ativos mostram-se como fatores-chave para o êxito das estratégias estabelecidas por qualquer instituição financeira. Pensando nisso, o sistema visa a agilizar os processos de modelagem, negociação e manutenção em carteira dos ativos adquiridos.

Assim, apresenta-se o objetivo do trabalho como o desenvolvimento do sistema proposto através de um processo sistemático de desenvolvimento. Para tal, aplica-se a disciplina de Gestão de Projetos no processo de desenvolvimento, de modo a garantir controle acerca de variáveis de projeto como o escopo, tempo e custo, visando sempre a otimizar a qualidade do produto final, e a Engenharia de Software, em especial nas fases iniciais do projeto. Utiliza-se o Processo Unificado (BOOCH; JACOBSON; RUMBAUGH, 1999b) como *framework* para o ciclo de vida de software. Este é um processo iterativo e incremental dividido nas fases de Concepção, em que se estabelece a justificativa e o escopo do projeto, identificando-se os requisitos do sistema, de Elaboração, na qual a arquitetura é definida e o sistema modelado, de Construção, em que se codifica baseando-se nos modelos estabelecidos na fase anterior, e por fim de Transição, na qual testes são realizados e o sistema é distribuído. Para o SPCRCA foi realizado um grande esforço na fase de Concepção, principalmente no levantamento de requisitos, e de Elaboração, no que diz respeito à definição da arquitetura e à modelagem do sistema. Com isso, projetou-se um sistema com foco para que sua arquitetura final seja robusta e escalável, que requisitos como configurabilidade, modularidade e processamento em tempo real fossem respeitados e também para que toda a complexidade inerente a um sistema como este fosse facilmente gerenciável.

Como produto final do trabalho obteve-se o SPCRCA com as funcionalidades de criação e configuração de *trades*, gerenciamento de *trades* em uma carteira de ativos e análises de fluxos de caixa, preços históricos, cenários de mercado e risco sobre *trades*.

1.2 Motivação

De acordo com o relatório de diretrizes para o currículo de graduação em Engenharia de Computação, elaborado por *Institute of Electrical and Electronics Engineers* (IEEE) e *Association for Computing Machinery* (ACM), a Engenharia de Computação é definida como “a disciplina que engloba a ciência e tecnologia no design, construção, implementação e manutenção de componentes de hardware e software de sistemas de computação modernos e equipamentos controlados por computadores” (IEEE, ACM, 2004). Esta área é tradicionalmente considerada como uma combinação da Ciência da Computação com a Engenharia Elétrica e está solidamente fundamentada nas teorias dessas duas áreas para a resolução de problemas técnicos, através do desenvolvimento de hardware, software, redes, processos, entre outros assuntos.

Desta forma, viu-se no Projeto de Formatura a oportunidade de se fazer Engenharia, buscando para isso, um projeto desafiador, que aplicasse os conceitos estudados ao longo do curso de graduação da Escola Politécnica da Universidade de São Paulo, complementando assim a formação de Engenheiro através da transformação de conhecimento explícito em tácito.

Observa-se que a análise de risco é imprescindível para instituições financeiras. O próprio cenário econômico mundial atual exemplifica a necessidade. As soluções existentes são ou baseadas em planilhas, apresentando-se em geral desorganizadas, de baixo desempenho, baixa configurabilidade e baixa escalabilidade, ou são específicos para uma dada funcionalidade ou tipo de produto. Além disso, a matemática financeira exigida para as soluções, que requer cálculos complexos com diversas variáveis e modelos, indicam a necessidade por um sistema robusto.

Com isso, a necessidade de um sistema de avaliação de carteira de ativos pela empresa co-orientadora Vector Investimentos veio ao encontro dos objetivos dos alunos no que diz respeito à sua perspectiva em relação ao Projeto de Formatura, uma vez que os requisitos apresentados exigiam uma solução de arquitetura bem estruturada e um processo de desenvolvimento orientado à qualidade do produto final, além de possibilitar a simulação de uma relação empresa/cliente.

1.3 Justificativa

As soluções existentes de análise de risco, como anteriormente mencionado, mostram-se ineficientes no que diz respeito à incorporação de novas funcionalidades e análises, desempenho, entre outros fatores. De fato, em razão da não existência de uma ferramenta que se adeque ao negócio, a necessidade do desenvolvimento de um sistema com tal intuito foi observada pela Vector Investimentos. Diante disto, este projeto representa não só um desafio quanto à sua complexidade de desenvolvimento (dada, entre outros fatores, a abrangência do sistema), mas principalmente uma quebra de paradigma quanto às práticas do mercado financeiro.

Do ponto de vista técnico, justifica-se este sistema com a necessidade de um desenvolvimento sistemático e controlado. O IEEE, através do seu Glossário Padrão de Terminologias de Engenharia de Software, define que a Engenharia de Software é “a aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para o desenvolvimento, operação e manutenção de software, e os estudos destas abordagens” (IEEE, 1990). Desta forma, utilizando ferramentas como *design patterns* e os *architectural patterns*, soluções reutilizáveis de problemas recorrentes e bem conhecidos, este trabalho visa aplicar os conceitos de Engenharia de Software no processo de desenvolvimento conforme expresso na definição, através de uma arquitetura orientada para o uso extensivo destas *patterns*, prática conhecida como Arquitetura de Software Orientada a Padrões (POSA), em um sistema de gerenciamento de risco de uma carteira de ativos.

Aceitou-se então o desafio. Nesta monografia está descrito o processo de desenvolvimento do sistema, desde o planejamento até a entrega da primeira versão do produto final, com enfoque nas diversas partes do ciclo de vida do software, destacando para cada fase os principais aspectos deste trabalho.

Existem ainda diversos outros aspectos do projeto que ilustram pontos de interesse da Engenharia de Computação. Estes problemas surgiram no decorrer do processo de desenvolvimento do sistema, e serão abordados ao longo deste trabalho.

1.4 Estrutura do Trabalho

O capítulo 2 apresenta o contexto no qual se insere este trabalho, identificando-se unidades organizacionais envolvidas e as perspectivas do produto final. Descrevem-se os conceitos de finanças necessárias para o entendimento sistema, assim como o levan-

tamento das necessidades do cliente. Além disso, conceitos teóricos de Engenharia e Ciência da Computação utilizados no processo de desenvolvimento também são abordados.

O capítulo 3 trata do planejamento do desenvolvimento do SPCRCA. São considerados os planejamentos do gerenciamento de escopo, de tempo, de custo e de risco. Apresenta-se a metodologia de trabalho, o levantamento de escopo, divisão deste em conjuntos de tarefas e em tarefas, estimativas de tempo, análise de risco, entre outros.

O capítulo 4 descreve as atividades da fase de Concepção do SPCRCA e os respectivos produtos gerados. Como atividades desta fase, foram levantados os principais requisitos funcionais e não-funcionais do sistema, os quais serviram de base para o entendimento do sistema, gerando ao final um documento sintetizando os aspectos relevantes.

O capítulo 5 descreve a fase de Elaboração, que consistiu da modelagem da maioria dos requisitos do sistema e da definição da sua arquitetura. As atividades desta fase incluem a modelagem do núcleo do sistema e da maioria dos componentes arquiteturais significativos.

O capítulo 6 descreve a fase de Construção, apresentando os aspectos relativos à implementação do sistema. Com os produtos desta fase, procura-se obter uma capacidade operacional inicial do sistema, sendo assim correspondem a um primeiro incremento, no contexto de Processo Unificado, levando em consideração as partes mais relevantes.

O capítulo 7 apresenta a fase de Transição do sistema, incluindo descrições dos testes e homologação, considerações sobre a distribuição do sistema e o treinamento dos usuários.

Por fim, o capítulo 8 apresenta as conclusões, as contribuições e os trabalhos futuros deste trabalho.

Em diversos momentos desta monografia, são utilizados termos em inglês. Isso ocorre quando estes termos ou forem mais largamente utilizados que os termos em português ou apresentarem maior clareza em relação ao significado desejado pelos autores.

2 Contexto do SPCRCA

*“As razões próprias nascem do entendimento, as alheias
vão pegadas à memória, e os homens não se convencem pela
memória, senão pelo entendimento.”*

Padre Antônio Vieira

2.1 Introdução

Neste capítulo é apresentada a contextualização do sistema no mercado financeiro, expondo as necessidades levantadas junto ao cliente. Por fim, identificam-se os conceitos teóricos necessários para o entendimento do trabalho.

Vale lembrar que objetivo deste trabalho não é um estudo sobre o mercado financeiro, e sim uma análise de como a tecnologia da informação pode auxiliar na tomada de decisões e controle de processos inserido no contexto do mercado financeiro.

2.2 Mercado Financeiro

Entende-se por mercado financeiro o meio pelo qual recursos financeiros excedentes são disponibilizados para empresas e pessoas físicas com necessidade de capital. Ou seja, assim como em qualquer outro mercado, no mercado financeiro os recursos vão de uma mão para outra, cobrando-se uma taxa por isto (NETO, 2005).

O mercado financeiro pode ser segmentado em quatro subdivisões:

- Mercado Monetário
- Mercado de Crédito
- Mercado de Capitais
- Mercado Cambial

O mercado monetário diz respeito a operações de curto e curtíssimo prazo, visando o controle eficaz da liquidez monetária na economia. Títulos de dívida pública são negociados neste tipo de mercado, destacando-se os papéis emitidos pelo Tesouro Nacional com o intuito de cobrir o orçamento público.

Neste tipo de mercado, a mais importante moeda de transação é a taxa de juros conhecida por taxa SELIC. O termo SELIC significa Sistema Especial de Liquidação e Custódia, que é um sistema utilizado para controlar e custodiar as negociações de títulos. À taxa definida no âmbito desse sistema dá-se o nome de taxa SELIC, que é aceita como a taxa livre de risco no mercado nacional.

O mercado de crédito engloba as operações de curto e médio prazo, voltado para a composição dos ativos permanentes e de capital de giro das empresas, ou seja, os ins-

trumentos negociados nesse tipo de mercado têm por objetivo principal suprir as necessidades de caixa a curto e médio prazo. As operações de crédito podem ocorrer por meio de concessão de créditos às pessoas físicas ou ainda empréstimos e financiamentos às empresas.

Entre os quatro tipos de mercado, o mercado de capitais aparece como fonte principal de recursos permanentes para a economia por efetuar a aproximação entre os que têm capacidade de poupança, os investidores, e aqueles com necessidade de recurso de longo prazo. Neste contexto, aparecem as tão faladas ações de empresas de capital aberto, que representam a menor parte do capital social da empresa.

Por último, aparece o mercado cambial, segmento onde ocorrem as operações de compra e venda de moedas internacionais.

Embora sirvam como base para o entendimento de como funciona o mercado financeiro, os quatro segmentos citados acima muitas vezes se sobrepõem, trazendo cada vez mais a necessidade de controles eficazes de acompanhamento e risco.

Diversos agentes participam do mercado financeiro, sendo estes dos seguintes tipos:

- Instituições Financeiras Bancárias
- Instituições Financeiras Não Bancárias
- Instituições Auxiliares
- Instituições Não Financeiras

A grande diferença entre instituições financeira bancária e não bancária é que à primeira “é permitida a criação de moeda por meio do recebimento de depósitos à vista (moeda escritural)” (NETO, 2005). Encaixam-se neste tipo de denominação os bancos comerciais e múltiplos. Já as instituições não-financeiras trabalham por meio de ações, letras de câmbio, títulos de dívida, entre outros, e são constituídas por Corretoras, Fundos de Investimentos, bancos de investimento, etc.

Instituições auxiliares são aqueles que contribuem para um melhor funcionamento do mercado por meio de intermediação, como por exemplo, sociedades corretoras e as bolsas de valores. Pelo fato do SPCRCA tratar somente de operações de compra e venda de ativos, e não da intermediação de operações, o escopo deste projeto trata somente das operações realizadas por instituições financeiras não bancárias por meio dos principais instrumentos financeiros existentes no mercado e listados a seguir.

- Ações
- Contratos Futuros
- *Swaps*
- *Non-Deliverable Forward*
- Títulos de Renda Fixa

2.2.1 Títulos e Valores Mobiliários

De uma maneira geral, entende-se por títulos e valores mobiliários tudo aquilo que pode ser negociado no mercado financeiro. Ao longo do projeto, utilizou-se o termo em inglês *securities* para títulos e valores mobiliários.

Foram consideradas, as seguintes *securities* no escopo do projeto: ações, derivativos, títulos de dívida pública e títulos de dívida privada. Suas descrições são fornecidas a seguir.

2.2.1.1 Ações

As ações representam a menor fração do capital social de uma empresa. Ao comprar uma ação o investidor não se torna um credor da empresa, mas um co-proprietário e, sendo assim, este possui o direito a participação dos resultados da empresa através de dividendos.

Neste trabalho é abordada somente a classe de ações pertencentes a empresas de capital aberto, ou seja, com ações negociadas em bolsas de valores e regulamentadas através da Comissão de Valores Mobiliários Comissão de Valores Mobiliários (CVM). Contudo, empresas de capital aberto devem fornecer regularmente informações de caráter econômico, social e financeiro, possibilitando assim a análise fundamentada do valor justo de suas ações. No contexto nacional, tem-se como principal bolsa de valores a BM&F BOVESPA S.A., criada em 2008 com a integração entre Bolsa de Mercadorias & Futuros (BM&F) e Bolsa de Valores de São Paulo (BOVESPA).

Para medir o desempenho médio dos papéis na bolsa de valores, são criados índices que funcionam como uma carteira teórica. O valor absoluto do índice expressa o valor de mercado da carteira.

O índice de bolsa mais utilizado no Brasil é o Índice Bovespa (Ibovespa), criado em 1968 com o objetivo de refletir o desempenho médio dos papéis mais negociados nos pregões da Bolsa de Valores de São Paulo. Regularmente a composição do Ibovespa é revista, tendo como critério o volume de negociação dos papéis nos últimos meses.

Alguns eventos podem alterar o valor de uma ação, porém não o valor da empresa. É importante notar que o valor de mercado de uma empresa de capital aberto nada mais é do que o valor de sua ação multiplicado pela quantidade de ações subscritas, ou seja, disponíveis em mercado.

Tais eventos podem ser de três tipos:

- Bonificação: Direito do acionista em receber novas ações.
- Desdobramento: Emissão de novas ações sem alterar o capital social da empresa.
- Grupamento: Diminuição do lote de negociação sem alterar o capital social da empresa.

No apêndice D é explicado como o sistema apura o valor de uma ação e de que forma esses eventos afetam o mesmo.

2.2.1.2 Derivativos

No mercado financeiro, alguns ativos financeiros dependem diretamente do valor de outros ativos utilizados como referência, chamados *underlying*. Por exemplo, um contrato futuro de petróleo dependerá do valor à vista do barril de petróleo.

Os derivativos são negociados por meio de contratos entre uma parte comprada e outra vendida, onde ambas concordam com um preço para o *underlying* em uma determinada data futura. A partir do fechamento do contrato, o valor deste será estabelecido pelo mercado de acordo com as leis de oferta e demanda.

O conceito de derivativos não é trivial e talvez de difícil compreensão à primeira vista. Para facilitar o entendimento pode-se pensar no caso no qual uma empresa A possui uma dívida em dólar para uma data futura enquanto outra empresa B tem um recebível em dólar para a mesma data. O receio da empresa A é de que a cotação do dólar suba até a data do pagamento, ao passo que para a empresa B o problema seria se a cotação do dólar estivesse muito abaixo do nível atual. As duas empresas decidem então entrar em contrato futuro de dólar, onde a empresa A ficaria comprada em dólar e a empresa B

vendida. Desse modo, no cenário de alta do dólar, a empresa A pagaria uma dívida maior, porém, teria o direito de comprar dólar ao nível acordado no contrato futuro.

O mesmo ocorre em um cenário de baixa para a empresa B, onde ela terá uma receita menor mas compensará tal perda na venda dólar pelo preço acertado no contrato. É importante dizer que, ao entrar em um contrato deste tipo, as duas partes devem previamente depositar uma margem de garantia, que será ajustada diariamente de acordo com a variação do valor futuro do *underlying* do contrato, ou seja, se o valor do *underlying* subir, parte da margem da posição vendida será depositada na margem da posição comprada e vice-versa. Tal mecanismo evita o não pagamento por uma das partes envolvidas no contrato.

Entre as vantagens e razões para se recorrer ao uso de derivativos pode-se citar:

- Proteger contra variações de preço que tragam prejuízo.
- Melhor gerenciamento do risco.
- Realizar negócios de maior porte com volume relativamente menor de capital e nível de risco conhecido.
- Trazer liquidez ao mercado físico.

Apesar de a função básica do derivativo ser a de possibilitar o melhor gerenciamento de risco, muitos enxergam os derivativos como uma oportunidade de especulação, exatamente por se tratar de uma expectativa de preço futuro, e não o preço de algo que possua valor real no presente. Neste projeto serão abordadas as transações com derivativos realizadas em mercados futuros, NDF e *swaps*, explicados a seguir.

Mercados Futuros

As operações de mercado futuro envolvem a negociação de um ativo financeiro em data futura por um preço fixado. Neste tipo de operação, há um compromisso de compra e venda formalizada em contrato e, diferentemente do mercado à vista, no mercado futuro não há a entrega física do ativo negociado, mas sim a liquidação em dinheiro dada pela diferença do preço estabelecido em contrato com o preço à vista do ativo na data de vencimento.

Os instrumentos de futuros são negociados na BM&FBOVESPA, permitindo a livre formação de preço e o controle das garantias às operações, utilizando o sistema de ajuste diário descrito nesta seção.

Os contratos futuros podem ser divididos de acordo com o seu ativo de referência *underlying*.

- Índice Bovespa Futuro: Como explicado na seção Ações, os índices funcionam como uma carteira teórica, sendo o Ibovespa o mais importante deles. No contrato futuro do Ibovespa, negocia-se qual será o valor do índice em uma determinada data, sendo este o contrato utilizado por Gestores como forma de proteger suas carteiras de Ações.
- DI Futuro: O contrato futuro de DI é baseado na taxa CDI (depósitos interfinanceiros) publicada pela CETIP (Central de Custódia e Liquidação Financeira) que mede a taxa média dos empréstimos de um dia para os títulos de dívida privada.
- Câmbio Futuro: O contrato de câmbio futuro é referenciado na taxa de câmbio de reais por dólar na data de vencimento do contrato, e constituem uma importante ferramenta de proteção para empresas exportadoras.
- Cupom Cambial Futuro: O cupom cambial relaciona a taxa de juros interna com a taxa de câmbio da moeda local com a moeda estrangeira e tem por objetivo medir o rendimento em dólar para os investimentos feitos em moeda estrangeira no mercado local, por exemplo, um investidor americano que compra títulos brasileiros.

Swap

O *swap* é um acordo entre duas partes com o intuito de se trocar fluxos no futuro (NETO, 2005). Neste acordo são definidas as datas de troca de fluxos e a forma como esses fluxos serão calculados. Geralmente, os cálculos dos fluxos envolvem taxas futuras de juros ou de câmbio.

Para ilustrar o mecanismo de um *swap* pode-se pensar no caso mais simples, onde apenas um fluxo será trocado no prazo de um ano. Uma parte do contrato de *swap* concorda em pagar 14% sobre um principal de R\$1.000.000,00, enquanto que a contraparte concorda em pagar taxa CDI sobre o mesmo principal na data combinada. Diz-se que um fluxo do contrato está pré-fixado em 14%, enquanto a outro fluxo do contrato está pós-fixado, pois variará de acordo com o CDI. Efetivamente, na data combinada, só será paga a diferença entre os dois fluxos, por exemplo, se a taxa CDI for de 15%, somam-se os fluxos e a diferença é paga pelo devedor.

Non Deliverable Forward

O *Non Deliverable Forward* (NDF) representa um acordo que tem por objeto uma taxa de câmbio combinada entre duas partes para uma determinada data (HULL, 2006). É importante ressaltar que não há a troca física de moeda estrangeira entre as partes, havendo somente o pagamento da diferença entre a taxa acordada em contrato e a taxa no dia do vencimento.

À primeira vista o NDF pode soar como sendo igual a um *swap*; este, porém, pode conter mais de um fluxo e, ainda por cima, atrelado a outras variáveis que não a taxa de câmbio. O NDF por sua vez sempre diz respeito a um único fluxo atrelado a uma taxa cambial acordada.

2.2.1.3 Títulos de Dívida Pública

Essencialmente, no Brasil, os títulos de dívida pública são emitidos tanto pelo Tesouro Nacional como pelo Banco Central, sendo o primeiro voltado para a execução de políticas fiscais e o segundo para o controle de políticas monetárias. De maneira simplificada, pode-se dizer que os títulos públicos representam uma forma de o governo federal captar recursos ou controlar a liquidez da economia, injetando ou tirando dinheiro de circulação.

Antigamente, os investidores considerados como pessoas físicas, compravam títulos públicos somente por aquisição de cotas de fundos de investimento. Hoje, porém, os títulos do tesouro nacional se tornam cada vez mais populares, representando uma forma atraente e segura de investimento. Dentre os títulos mais negociados encontram-se as Letras do Tesouro Nacional (LTN) e Letras Financeiras do Tesouro (LFT).

2.2.1.4 Títulos de Dívida Privada

Os títulos de dívida privada representam uma forma de as empresas captarem recursos com prazo e rendimentos determinados na emissão dos títulos. No contexto deste projeto de formatura, são de especial relevância as Cédulas de Produto Rural (CPR) e os Certificados de Direitos Creditórios do Agronegócio (CDCA). Ambos tratam de operações de crédito para produtores rurais, por isso, envolvem garantias físicas e demandam atenção especial por parte do detentor do título.

2.2.2 Fundos de Investimento

Entende-se por Fundo de Investimento um conjunto de recursos monetários, provenientes de depósitos de um ou mais de investidores, tendo por objetivo a aplicação em carteiras de títulos e valores mobiliários. Os fundos, por representarem uma forma coletiva de aplicação de recursos, trazem vantagens principalmente ao pequeno investidor.

O total de recursos aplicados em um Fundo denomina-se patrimônio líquido, que é dividido por participações conhecidas por cotas; por isso, o investidor de um Fundo de Investimento é conhecido como cotista.

O Responsável por selecionar o destino dos recursos captados por um Fundo é o gestor, sendo dele, também, a responsabilidade por assumir a exposição ao risco que melhor se adeque aos propósitos do fundo.

2.2.3 Tipos de Análises

Apresenta-se a seguir os tipos de análises sobre operações financeiras a serem realizadas pelo sistema proposto. São elas:

- *Market Pricing*
- *Market Risk*
- *Market Scenario*
- *Market Historical Pricing*
- *Market Profit and Loss Explain*
- *Cash Flow*

2.2.3.1 Market Pricing

Cada título ou valor mobiliário negociado no mercado financeiro possui seu valor em determinada data. Este valor pode ser passado, ou seja, um registro de quanto o ativo valia em uma data passada, ou ainda uma projeção sobre uma data futura, utilizando os parâmetros atuais do mercado financeiro.

Através da análise de *Market Pricing*, procura-se, acima de tudo, saber o valor total de uma carteira de investimentos, através da soma dos valores de todos os ativos que a compõem.

2.2.3.2 Market Risk

Tão importante quanto saber o valor de um ativo é saber o risco que este corre de acordo com as variações do que ocorrem no mercado financeiro. A análise de *Market Risk* tem por objetivo explicitar de que forma uma pequena variação em uma determinada variável de mercado, como por exemplo, a taxa de câmbio, afetaria o valor de um ativo em carteira.

Conhecer o risco assumido por uma carteira de investimentos possibilita ao Gestor tomar medidas de proteção ao patrimônio mais eficientes, evitando assim surpresas desagradáveis com possíveis variações no mercado financeiro.

2.2.3.3 Market Scenario

A análise de *Market Scenario* possibilita, ao analista financeiro, a criação de diferentes cenários econômicos, com o intuito de auxiliá-lo na tomada de decisões.

2.2.3.4 Historical Pricing

O histórico de preço de um ativo financeiro diz muito sobre ele, ainda mais quando comparado com histórico de preço de outros ativos correlacionados. A maneira mais eficaz de realizar tal comparação é através de gráficos de preço, sendo este o intuito da análise de *Historical Pricing*.

2.2.3.5 Profit and Loss Explain

Especialmente no caso dos Fundos de Investimento, é interessante saber no presente quais as variações ocorridas no dia anterior justificam um possível lucro ou perda no patrimônio. Por exemplo, sabendo-se que uma cota, de um determinado Fundo, sofreu apreciação de 5% no dia anterior, deseja-se saber quais os ativos foram responsáveis por tal valorização. Para tanto, utiliza-se a análise de *Profit and Loss Explain*.

2.2.3.6 Cash Flow

Por último, tem-se a análise de *Cash Flow*, que nada mais é do que uma representação gráfica dos fluxos ocorridos ao longo da operação. Tal representação dá ao analista a noção exata do que ocorrerá com o caixa de suas operações.

2.3 Perspectivas do Produto

Espera-se, com o sistema proposto por este trabalho, que os processos de negócio, antes tratados por *spreadsheets*, possam ser gerenciados de modo que a inclusão de novas operações financeiras ou a execução de operações mais complexas não implique um grande retrabalho para os analistas. Deve-se garantir, assim, que haja escalabilidade e flexibilidade de reconfiguração.

Com a automatização destes processos, espera-se também a adição de funcionalidades anteriormente inexistentes, além de garantir integrabilidade com módulos já existentes, alto desempenho, modularidade, manutenibilidade, robustez, processamento em tempo real, confiabilidade, disponibilidade e segurança. A arquitetura também deve prever, devido ao foco em Orientação a Objetos (OO) e *design patterns*, a reutilização de objetos, a flexibilidade e a extensibilidade do sistema.

2.4 Unidades Organizacionais Envolvidas

Neste trabalho, foi visada a simulação de um relacionamento empresa-cliente com o requisitante do sistema em questão. Desta forma, estão envolvidos neste trabalho: a empresa fornecedora, constituída pelo grupo de alunos de graduação do curso de Engenharia de Computação da Escola Politécnica da Universidade de São Paulo; o cliente, constituído pela Vector Investimentos, empresa administradora de recursos financeiros, especializada em investimentos alternativos baseados em modelos quantitativos e operações estruturadas; e por fim, a Universidade de São Paulo, na supervisão e orientação deste trabalho. Os termos empresa fornecedora e cliente são doravante utilizados ao longo desta monografia.

2.5 Conceitos Teóricos

2.5.1 Gerenciamento de Projeto

A disciplina de gerenciamento de projetos prevê a aplicação de conhecimento, ferramentas e técnicas para alcançar os objetivos de um projeto (PROJECT MANAGEMENT INSTITUTE, 2004). Para entendimento completo de tal afirmação, define-se projeto como um empreendimento finito (com datas de início e fim especificados), com intuito de criar um produto ou serviço único que trará um benefício de algum valor agregado através de uma série de atividades que utilizam os recursos disponíveis. Desta forma, os objetivos de um projeto consequentemente indicam a situação final desejada pelo requisitante. Para tal, é imperativo que estes objetivos sejam específicos, mensuráveis, alcançáveis ou aceitáveis, relevantes e determinados no tempo.

A gestão de um projeto, segundo Project Management Institute (2004), ocorre através dos processos de iniciação, planejamento, execução, monitoração e controle, e fechamento. Estes processos incluem atividades tais como identificação de requisitos, estabelecimento de objetivos claros e alcançáveis e balanceamento das demandas por qualidade, tempo e custo, adaptando-se as especificações, planejamentos e abordagens para as diversas expectativas dos *stakeholders*, ou seja, indivíduos e organizações envolvidas. Inclusive, o gerenciamento de projeto tem como base um tripé formado pelo escopo, custo e tempo do projeto, que são interdependentes e geralmente apresentam competição entre si (um tempo menor, por exemplo, pode indicar aumento no custo e redução de escopo).

Nas seções a seguir, serão apresentados alguns tipos de gerenciamento previstos em Project Management Institute (2004).

2.5.1.1 Gerenciamento de Escopo

O gerenciamento de escopo tem por objetivo definir o trabalho que está incluso e o que não está no projeto, de modo a assegurar uma conclusão bem sucedida. É a documentação primária para o entendimento do projeto e sua natureza. Inclui processos para o planejamento e a definição do escopo, está servindo de base para decisões de projeto, subdivisão do trabalho e dos entregáveis em componentes mais gerenciáveis através do *Work Breakdown Structure* (WBS), verificação do escopo e, por fim, controle de modificações (PROJECT MANAGEMENT INSTITUTE, 2004). Observa-se que o WBS é uma ferramenta para identificar todos os itens de trabalho imprescindíveis para o sucesso do

projeto, concentrando-se em mapear os pacotes de trabalho gerados na quebra do projeto em componentes de maior nível de detalhe (NEWELL; GRASHINA, 2004). Estes pacotes são orientados pelos entregáveis previstos para cada fase.

2.5.1.2 Gerenciamento de Tempo

O gerenciamento de tempo tem por objetivo assegurar o término do projeto dentro do prazo estipulado. Este processo é composto primordialmente da definição das atividades (interdependentes entre si) que produzirão os entregáveis do projeto. Existem diversas técnicas para se definir as atividades, como a decomposição dos pacotes de trabalho definidos no escopo em atividades, o uso da lista de atividades de um outro projeto como *template* e a evolução e elaboração progressiva do WBS (PROJECT MANAGEMENT INSTITUTE, 2004). As atividades resultantes devem ser relacionadas entre si através de um diagrama de rede que expressa as dependências de cada atividade (NEWELL; GRASHINA, 2004).

Deve-se, em seguida, realizar a estimativa de recursos disponíveis e a estimativa da duração das atividades através de técnicas tais como o *bottom-up*, que estima o tempo total do projeto como uma função da estimativa do tempo de cada atividade e a *top-down*, que calcula o tempo baseado em projetos anteriores similares. Com estas informações, é possível desenvolver o cronograma, que provê um “*roadmap* que representa como e quando um projeto entregará os produtos definidos no escopo” (PROJECT MANAGEMENT INSTITUTE, 2007).

2.5.1.3 Gerenciamento de Custo

O processo de gerenciamento de custos está relacionado ao custo dos recursos utilizados para completar as atividades previstas, devendo incluir também os custos de uso, manutenção e suporte do produto, serviço ou resultado do projeto (PROJECT MANAGEMENT INSTITUTE, 2004). As técnicas para estimativa do custo total de projeto são análogas às técnicas para determinar o tempo, diferindo nas entradas utilizadas, como por exemplo, condições de mercado e bases de dados comerciais. O produto deste processo é o *cost baseline* (construção que indica a utilização do orçamento aprovado distribuído sobre o período de tempo do projeto), incluindo também requisitos de investimento no projeto e planos para o controle de mudanças.

2.5.1.4 Gerenciamento de Risco

O gerenciamento de riscos inclui atividades relacionadas à condução do planejamento, identificação, análise quantitativa e qualitativa, controle, monitoramento e mitigação dos riscos do projeto. Deve haver um processo sistemático de identificação de riscos, de forma que estes sejam descritos em um nível de detalhe consistente, havendo também definições para a probabilidade de cada risco e impactos que eles possam vir a gerar através de uma matriz com escalas de impacto. Devem-se listar inclusive as causas e possíveis ações a serem tomadas, de modo que, na eventual ocorrência de um risco, exista um plano de mitigação bem estruturado (PROJECT MANAGEMENT INSTITUTE, 2004) (SODHI; SODHI, 2001).

2.5.2 Engenharia de Software

A Engenharia de Software é uma tecnologia composta por camadas, que correspondem às camadas de ferramentas, método, processo e qualidade, conforme indicado na figura 1 (PRESSMAN, 2004).

A camada de processos é o alicerce da Engenharia de Software, formando a base da gestão de projetos de software e cujas principais áreas estabelecem o contexto no qual o produto será desenvolvido. Desta forma, o processo que provê um *framework* no qual métodos técnicos são aplicados, produtos são desenvolvidos, marcos são estabelecidos, qualidade é assegurada e configuração é gerenciada. Existem alguns modelos existentes para processos de desenvolvimento de software, entre eles o Processo Unificado, que será apresentado a seguir. A camada de métodos provê o conhecimento técnico para a construção do sistema, utilizando-se de princípios básicos que governam cada área da tecnologia e incluindo atividades de modelagem e outras técnicas descritivas. Por fim, a camada de ferramentas provê apoio automatizado ou semi-automatizado para esses processos e métodos.

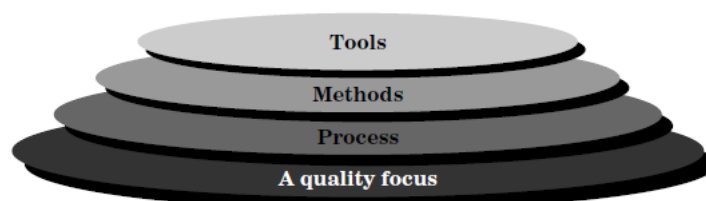


Figura 1: Camadas da Engenharia de Software

2.5.2.1 Processo Unificado

Todos os processos de software podem ser caracterizados por um *framework* comum (PRESSMAN, 2004), através de um pequeno número de atividades aplicáveis para todos os projetos de software, independente do tamanho ou complexidade. Estas atividades devem ser detalhadas através de conjuntos de tarefas de Engenharia, marcos de projeto, produtos e aspectos de garantia de qualidade de forma a serem adaptadas para as características de cada projeto.

O Processo Unificado é um *framework* iterativo e incremental para desenvolvimento de software, baseado em casos de uso para capturar requisitos funcionais, centrado na arquitetura e focado nos riscos de projeto. Ele não é, ao contrário do que seu próprio nome diz, apenas um processo, mas um *framework* extensível e customizável para necessidades específicas de projetos e organizações. Para este projeto, optou-se por utilizar o Processo Unificado devido ao foco do sistema na arquitetura e à pequena maturidade inicial em relação ao sistema final. Dessa forma, é essencial que os requisitos, arquitetura e modelagem fossem revistos constantemente, para que houvesse consistência no sistema inteiro.

O Processo Unificado apresenta quatro fases: Concepção, Elaboração, Construção e Transição (BOOCH; JACOBSON; RUMBAUGH, 1999b). Cada fase possui metas para serem cumpridas e as suas atividades são compostas por atividades que pertencem a fluxos de trabalho (requisitos, análise, projeto, implementação, teste). Um exemplo do uso do Processo Unificado se encontra na figura 2.

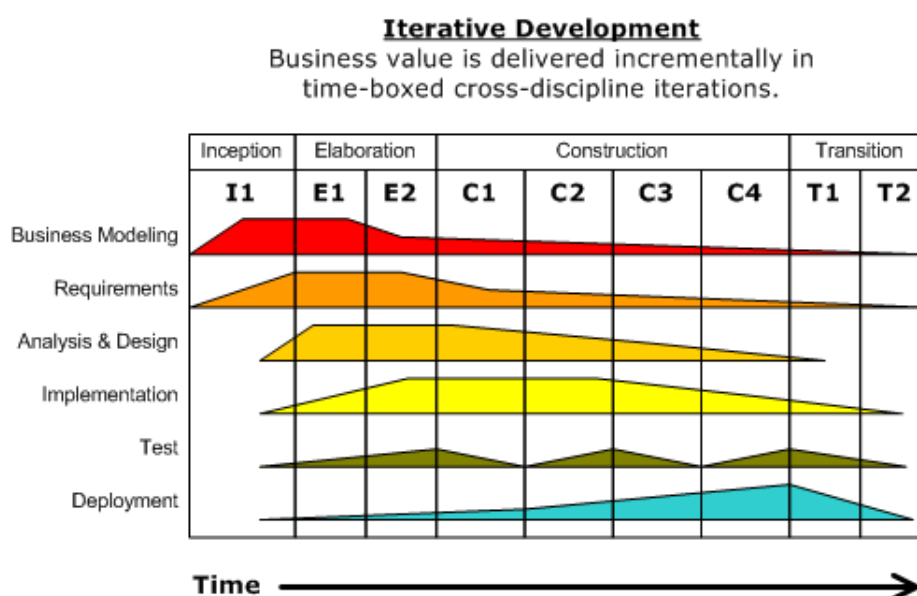


Figura 2: Fases do Processo Unificado

A fase de Concepção tem como principais objetivos estabelecer a justificativa (ou *business case*) e o escopo do projeto, identificando requisitos, os casos de uso indispensáveis e os candidatos para a arquitetura. Devem-se identificar os principais riscos e preparar um cronograma e uma estimativa de custo iniciais.

A fase de Elaboração visa identificar a grande maioria dos requisitos do sistema, levando em conta os fatores de risco e criar uma arquitetura de sistema. Inclui, como tarefas, a modelagem através de diagramas de casos de uso, classes, entre outros, utilizando a *Unified Modeling Language* (UML) como representação destes modelos. A arquitetura é validada através da criação de um protótipo mínimo, que consta do núcleo do sistema e dos componentes mais significativos.

A fase de Construção é a que envolve maior volume de trabalho; nela o sistema é codificado de acordo com os modelos estabelecidos na Elaboração e os testes são realizados.

Por fim, a fase de Transição tem como objetivo distribuir o sistema para os usuários finais, sendo os *feedbacks* dos usuários utilizados para gerar refinamentos a serem incorporados nas iterações desta fase.

No Processo Unificado, as diversas funcionalidades do sistema são implementadas em uma série de iterações, cada uma delas resultando em um novo *release* do sistema.

2.5.2.2 Unified Modeling Language

Oficialmente definido pelo *Object Management Group* (OMG), a UML é uma linguagem utilizada para visualização, especificação, construção e documentação de sistemas de software (BOOCH; JACOBSON; RUMBAUGH, 1999a). Nota-se que a UML não é um método por si só, mas foi especificada de forma a ser compatível com métodos de desenvolvimento de software. Ele é inicialmente independente de processo, mas apresenta maiores benefícios quando utilizado junto a um processo focado em casos de uso, centrado na arquitetura e que seja iterativo e incremental (BOOCH; JACOBSON; RUMBAUGH, 1999a), sendo assim amplamente utilizado com o Processo Unificado.

A UML 2.0 inclui os diagramas de estrutura, como por exemplo o diagrama de classes, de componentes, de distribuição e de pacotes, e os diagramas de comportamento, como o diagrama de atividade, de casos de uso, de sequência e de comunicação. A representação destes diagramas é feita através de blocos, que podem ser abstrações dos entes a serem modelados, e relacionamentos entre estas abstrações.

2.5.2.3 Engenharia de Requisitos

“Um requisito é um atributo necessário de um sistema, que identifique a capacidade, características ou fator de qualidade de um sistema de modo que este tenha valor e utilidade para o consumidor ou usuário” (YOUNG, 2004). A Engenharia de Requisitos é uma disciplina que engloba todas as atividades do processo de produção de um documento de requisitos, e manutenção do mesmo. Suas atividades incluem a identificação dos *stakeholders*, entendimento das necessidades e expectativas do sistema planejado, compreensão do domínio na qual o sistema se insere, identificação de requisitos, clarificação, especificação, priorização, alocação, rastreamento, gerenciamento, teste e validação (YOUNG, 2004). Deve-se ressaltar que os requisitos nem sempre são identificáveis desde o início do projeto e, portanto, um ciclo de vida incremental deve gerenciar esta situação.

Para o processo de desenvolvimento de software, os requisitos podem ser funcionais, que indicam as capacidades operacionais do sistema, e os não-funcionais, que são propriedades como confiabilidade e segurança. Uma especificação criteriosa deve garantir que todos os requisitos sejam necessários, viáveis, corretos, concisos, não ambíguos, completos, consistentes, verificáveis, rastreáveis, alocáveis, independentes de *design* e não redundantes (YOUNG, 2004).

A validação da implementação de um requisito pode ser feita através de técnicas como a prototipação e geração de casos de testes. De todo modo, observa-se a importância de ter requisitos verificáveis e rastreáveis através de todo o sistema (seja no *design*, na codificação, em testes e inclusive na documentação) para que a validação do produto final seja concisa e completa, garantindo assim que os objetivos iniciais do projeto sejam atingidos.

2.5.3 Arquitetura de Software

A Arquitetura de Software de um sistema computacional pode ser definida como “a(s) estrutura(s) deste sistema, formada a partir de elementos de software, as propriedades de visibilidade externa destes e o relacionamento entre eles” (BASS; CLEMENTS; KAZMAN, 1997). Ela manifesta um conjunto inicial de decisões de *design*, definindo restrições globais para a implementação e ditando a organização geral do sistema, tais como paradigmas de programação e estilos de arquitetura. Através da arquitetura, é possível exibir os atributos do sistema, como por exemplo alto desempenho, prever as qualidades do sistema e facilitar a gerência de alterações. A arquitetura também é descrita como *design* estratégico. Difere

do *design* detalhado (ou tático), uma vez que este está relacionado a restrições locais, tais quais *design patterns* e *architectural patterns*.

2.5.3.1 Conceitos de Arquitetura de Software

Existem alguns conceitos relacionados à Arquitetura de Software que são interessantes citar neste momento. A primeira delas, estilo de arquitetura, é definida como o agrupamento de sistemas em função de sua estrutura organizacional (BUSCHMANN et al., 1996). Deve expressar tanto a estrutura fundamental quanto o método utilizado para construí-lo. Pode-se mencionar também o conceito de *framework*, que nada mais é que um sistema ou subsistema parcialmente completado, provisionando blocos para a construção deste (BUSCHMANN et al., 1996).

No entanto, para se discutir as representações de arquitetura, apresentam-se as seguintes definições: estrutura e visão (BASS; CLEMENTS; KAZMAN, 1997). Por visão, entende-se a representação de um conjunto coerente de elementos da arquitetura e seus relacionamentos, como por exemplo a visão funcional/lógica. Já estrutura é o conjunto de elementos em si, do modo como eles existem em hardware e software. As estruturas podem ser divididas em três grupos. As estruturas modulares indicam “unidades de implementação”, apresentando uma representação baseada no código. A estrutura de componentes e conectores sinaliza o comportamento em tempo de execução, ilustrando como as unidades de computação comunicam-se entre si. Por último, a estrutura de alocação divide os elementos de software de acordo com o ambiente no qual eles são criados e executados, como por exemplo em qual processadores serão executados, em qual arquivos serão armazenados, entre outros.

2.5.3.2 Princípios Fundamentais

Conforme Buschmann et al. (1996), são citados alguns princípios fundamentais da disciplina de Arquitetura de Software, que são técnicas significativas para o sucesso do desenvolvimento de um sistema de software.

Abstração é um princípio largamente utilizado para lidar com complexidade, conceito não provindo da computação, mas sim da própria natureza humana. A abstração é a característica essencial de um objeto ou um componente que o difere dos outros, definindo barreiras conceituais. Também relacionado a abstrações, o encapsulamento trata do agrupamento dos elementos que constituem a estrutura e o comportamento de uma abstração.

Modularização é a decomposição de um sistema de software em subsistemas e componentes, apresentando fronteiras bem definidas entre eles para lidar com a complexidade. Módulos servem como *containers* físicos para funcionalidades ou responsabilidades de uma aplicação. Inclusive, a separação de responsabilidades indica que papéis não-relacionados entre si devem ser separados dentro de um componente. Todo o componente de software deve ser suficiente, completo e primitivo, ou seja, deve possuir as características da abstração que permitem uma interação eficiente, deve ter todas as características relevantes da abstração e todas as operações devem ser simples de implementar.

O ato de ocultar informações também caracteriza um princípio, fazendo com que os detalhes da implementação de um componente sejam transparentes para seu cliente, minimizando assim o acoplamento entre os módulos. Análogo ao acoplamento, tem-se a coesão. Ela, ao contrário do acoplamento, trata de aspectos intramodulares, sendo assim uma medida para a conectividade entre as funções e elementos dentro de um único módulo.

Deve haver separação entre política e implementação e entre interface e implementação. Por política, entende-se componentes que lidam com decisões sensíveis ao contexto, com conhecimento da semântica e interpretação da informação, diferentemente da implementação, que executa algoritmos independente do contexto. Uma interface, por sua vez, consiste apenas das assinaturas das funções, definindo assim a funcionalidade do componente. A interface é acessível aos clientes, enquanto que sua implementação não deve ser.

A técnica de ponto de referência única indica que os itens dentro de um sistema de software devem ser em geral declarados e definidos apenas uma vez, de modo a não gerar inconsistências. Por fim, o princípio de dividir e conquistar sugere a divisão de tarefas ou componentes em partes menores que podem ser desenhadas independentemente.

2.5.4 Design Patterns e Architectural Patterns

2.5.4.1 Design Patterns

Design pattern são soluções reutilizáveis para problemas gerais de *design* em um contexto particular. Eles expressam técnicas comprovadas como bem-sucedidas para sistemas OO, apresentando sempre: o contexto do problema ao qual devem ser aplicados, a solução empregada através de uma descrição abstrata do arranjo geral dos elementos, e as consequências da utilização dos *patterns* (GAMMA et al., 1995) (BUSCHMANN; HENNEY;

SCHMIDT, 2007b).

Um formato consistente para descrever os *design patterns* vê-se necessário, de forma a documentar o contexto no qual determinado *pattern* é inserido. Este *template*, frequentemente adotado na literatura e introduzido em (GAMMA et al., 1995), caracteriza-se por dispor informações acerca do nome e da classificação do *pattern* em questão, de seu intuito, outros nomes pelos quais é conhecida, motivação, aplicabilidade, estrutura através de uma representação gráfica das classes, classes e objetos participantes e seus papéis, consequências, considerações sobre implementação, exemplo de código, usos conhecidos e *patterns* relacionados.

A tabela 1 apresenta uma breve descrição de alguns *design patterns*. Esses *patterns* são apenas algumas de uso geral e estão presentes em (GAMMA et al., 1995). Existem diversas outras voltadas para assuntos específicos, tais como objetos concorrentes em rede (BUSCHMANN et al., 2000), para gerenciamento de recursos (KIRCHER; JAIN, 2007), para computação distribuída (BUSCHMANN; HENNEY; SCHMIDT, 2007a), para segurança (SCHUMACHER et al., 2005), entre outros.

2.5.4.2 Architectural Patterns

Architectural patterns oferecem soluções já estabelecidas para problemas de arquitetura em Engenharia de Software através da descrição de elementos e tipos de relacionamentos, junto de um conjunto de restrições de como utilizá-los (BASS; CLEMENTS; KAZMAN, 1997). O termo estilo de arquitetura, mencionado anteriormente, também é amplamente utilizado neste contexto. Classificam-se estes *patterns* em quatro grupos, não sendo esta divisão exaustiva (BUSCHMANN et al., 1996). Neste trabalho, são apresentados apenas alguns exemplos de *architectural patterns*.

A primeira categoria diz respeito a *patterns* com o intuito de evitar um conjunto desorganizado com muitos componentes. Ela fornece suporte para a decomposição de uma tarefa de um sistema em subtarefas de forma controlada. Pertence a esta categoria por exemplo a arquitetura em camadas, na qual cada uma daquelas subtarefas pertence a um nível de abstração particular.

A segunda categoria trata sistemas distribuídos. Um exemplo desta categoria é o *microkernel*, que separa as funções mínimas do núcleo da funcionalidade específica dos consumidores, servindo como soquete para adicionar extensões e coordená-las. Os sistemas distribuídos são interessantes por garantirem desempenho, escalabilidade e confiabilidade.

Tabela 1: Catálogo de *Design Patterns* (adaptado de Gamma et al. (1995))

Nome	Descrição
Abstract Factory	Interface para criar famílias de objetos relacionados ou dependentes sem especificar sua classe concreta.
Adapter	Permite que classes com interfaces incompatíveis trabalhem juntas através da conversão de interfaces.
Bridge	Desacopla uma abstração de sua implementação de modo que ambos possam variar independentemente.
Builder	Separa a construção de objetos da sua representação, de modo que um único processo de construção gere diversas representações.
Chain of Responsibility	Evita o acoplamento entre quem envia e quem recebe uma requisição, dando a mais de um objeto a oportunidade de lidar com ela. As requisições são repassadas pela corrente até que um objeto a consuma.
Command	Encapsula uma requisição como um objeto, permitindo parametrizar clientes com diferentes requisições.
Composite	Compõe os objetos em uma estrutura de árvore, permitindo que clientes tratem objetos individuais e composições de objetos da mesma forma.
Facade	Provê a interface para um conjunto de interfaces em um subsistema.
Factory Method	Define uma interface para criar um objeto, permitindo que as subclasses decidam qual classe instanciar.
Interpreter	Define o interpretador para uma certa representação de gramática de uma dada linguagem.
Iterator	Provê um método para acesso de elementos de um objeto agregado sequencialmente.
Lazy Instantiation	Atrasa a criação de um objeto até a primeira vez que for necessário.
Mediator	Define um objeto que encapsula como um conjunto de objetos interação entre si.
Memento	Captura e externaliza o estado interno de um objeto para que este seja recuperado posteriormente.
Observer	Define uma dependência um-para-muitos entre objetos, de modo que se o estado de um destes objetos muda, todos os dependentes são notificados.
Prototype	Cria objetos copiando um protótipo.
Proxy	Controla o acesso a um objeto.
Singleton	Assegura que uma classe tenha apenas uma instância, e provê um ponto global de acesso.
State	Permite um objeto alterar seu comportamento quando o estado interno mudar.
Strategy	Encapsula uma família de algoritmos, permitindo variar entre eles independentemente.
Template Method	Define o esqueleto de um algoritmo em uma operação, permitindo que subclasses redefinam alguns passos do algoritmo.
Visitor	Representa uma operação a ser executada nos elementos da estrutura de um objeto.

A terceira categoria engloba sistemas interativos. Possui dois *patterns*: o *Model-View-Controller* (MVC) e o *Presentation-Abstraction-Control* (PAC). O MVC consiste de três componentes, que são o *model*, responsável por encapsular o núcleo das funcionalidades e os dados, o *view*, responsável por disponibilizar informação para o usuário, e o *control*, que gerencia a entrada do usuário. Já o PAC define uma estrutura para interação entre sistemas de software na forma de uma hierarquia de agentes que cooperam entre si, na qual cada agente é responsável por um aspecto da funcionalidade do sistema. O componente de *presentation* é relacionado à interação homem-computador, o *abstraction* ao núcleo e o *control* à comunicação com outros agentes.

Por fim, a última categoria visa os sistemas adaptáveis. O *microkernel*, por exemplo, consegue se adaptar facilmente à mudança nos requisitos de sistema, enquanto o *reflection* provê um mecanismo para mudança na estrutura e no comportamento do sistema dinamicamente.

2.6 Considerações Finais

Este capítulo abordou os principais aspectos teóricos utilizados no trabalho. Os conceitos relacionados ao mercado financeiro são utilizados primordialmente para entendimento das necessidades do usuário, sendo possível assim o levantamento de requisitos das funções de negócio. Além disso, para que o sistema apresente o nível de qualidade desejado, são aplicados os conceitos de Gerência de Projetos e Engenharia de Software, de forma que haja controle sobre o processo de desenvolvimentos em diversos níveis. Por fim, os *design patterns* e *architectural patterns* são largamente utilizados na arquitetura e modelagem do sistema, prática conhecida como POSA. Esta tem por objetivo aplicar técnicas de construção de sistemas bem-definidos seguindo os princípios fundamentais de arquitetura e enfatizando a importância de propriedades não-funcionais. Desta forma, o uso destes padrões auxilia na análise e descrição de propriedades de alto-nível de sistemas complexos, dando suporte para sua mudança e evolução (BUSCHMANN et al., 1996).

As perspectivas quanto ao produto estabelecem diversos requisitos que, na sua maioria, podem ser facilmente observados qualitativamente no sistema final devido ao foco que este deve ter na arquitetura. De fato, o próprio protótipo já mostra a perspectiva de atendimento de modularidade e escalabilidade, por exemplo. Análises quantitativas das métricas, no entanto, estão fora do escopo do projeto.

3 *Planejamento*

“A estratégia é uma economia de forças.”

Karl von Clausewitz

3.1 Introdução

Neste capítulo são apresentados a metodologia de trabalho e o gerenciamento dos processos utilizados para o desenvolvimento do sistema. O produto das atividades mapeadas por este capítulo é o Planejamento do Projeto. Portanto, as próximas seções descrevem os processos de gerenciamento do escopo, através da definição do escopo e quebra do trabalho em tarefas, gerenciamento de tempo e de custo, através das estimativas de projeto, e gerenciamento de risco, através do levantamento dos riscos e estratégias para diminuição do impacto dos mesmos.

3.2 Metodologia de Projeto

O processo de desenvolvimento utilizado para o SPCRCA é uma customização do Processo Unificado. Como sua própria definição expressa, ele é um *framework* a ser adaptado para as diferentes necessidades, sendo focado para o controle de risco e centrado na arquitetura. Estas duas características são não só desejáveis, mas imperativas para o sistema proposto neste trabalho, uma vez que a necessidade de controle de risco é determinada pela inicial falta de maturidade para estimativas de projeto que condigam com o prazo determinado para a conclusão, e que o foco na arquitetura é uma característica essencial do processo de desenvolvimento para os requisitos apresentados no capítulo 4. Além das fases e atividades propostas por este processo, utilizou-se também da disciplina de Gerenciamento de Projetos prevista pelo *Rational Unified Process* (RUP), uma extensão Processo Unificado (IBM, 2008), de modo a garantir maior controle sobre aspectos como risco e qualidade.

O desenvolvimento segue em sua maior parte as premissas do Processo Unificado, conforme descrito ao longo desta monografia. Uma ressalva porém acontece com o foco em casos de uso previsto no Processo Unificado. Houve neste trabalho utilização de protótipos da interface com o usuário como ferramenta de auxílio na construção do modelo de casos de uso para a captura dos requisitos funcionais, prática justificada pela aquisição de maior entendimento das necessidades do usuário final antes do início da modelagem. Como motivação para tal, cita-se que o investimento médio na indústria com o processo de levantamento de requisitos equivale de 2% a 3% do custo total do projeto e, nessa premissa, o custo real ultrapassou em 80% a 200% o custo planejado; por outro lado, projetos que concentraram 8% a 14% de esforço no levantamento de requisitos apresentaram custos de até apenas 50% maiores (YOUNG, 2004), embora obviamente o ideal seja que

não haja aumentos. A escolha de dar maior importância ao levantamento de requisitos mostrou-se muito importante para o projeto pelo tempo economizado nas fases seguintes.

Quanto à organização desta monografia, apresentam-se algumas considerações. Deve-se ressaltar que há um descompasso no que diz respeito às atividades das fases no processo real e no processo como aqui descrito. No processo real utilizado de fato no desenvolvimento, as atividades são iterativas e incrementais. Os esforços para o levantamento de requisitos, por exemplo, acontecem em sua maior parte na fase de Concepção, muito embora também sejam recorrentes nas outras fases. Devido à dificuldade de se relatar nesta monografia essas iterações conforme ocorreram, optou-se por mencionar o resultado final apenas na fase em que há maior concentração de esforços para realização de uma atividade. Assim, seguindo o exemplo dado, os requisitos são descritos sob a fase de Concepção, e inclui-se qualquer reengenharia ocorrida posteriormente. Da mesma forma, argumenta-se que tanto os objetivos como o escopo deveriam pertencer à fase de Concepção, mas eles foram deslocados de seu capítulo original motivados pela necessidade de sua introdução em momentos anteriores nesta monografia.

3.3 Gerenciamento de Escopo

3.3.1 Escopo da Solução

Fundos de investimento são formas de aplicação financeira nas quais grupos de investidores realizam investimentos financeiros visando um determinado retorno esperado, havendo a divisão da receita gerada e das despesas necessárias no empreendimento. Esses fundos possuem carteiras de ativos montadas pelos gestores, que constituem o conjunto dos bens e direitos resultante das transações das quais futuros benefícios são esperados. Uma carteira de ativos é formada por ativos ou *securities* de diversos tipos, cada qual com um risco diferente. O risco total da carteira é repassado ao aplicador ou cotista do fundo, sendo a sua análise de grande relevância para os clientes do fundo.

Neste cenário, o SPCRCA tem como finalidade primordial ser uma ferramenta de auxílio para o estudo da viabilidade de uma *security*, auxiliando os gestores do fundo a montar estratégias que adicionam ativos no mesmo, através da análise de informações do histórico das variáveis relevantes e das simulações de parâmetros de mercado. Um usuário, através de um sistema de modelagem das *securities*, poderá avaliá-las quanto ao seu valor ou ao risco. Nesta avaliação, as *securities* da carteira poderão ser filtradas e agrupadas de maneira customizável, e os diversos cálculos serão aplicados sobre um grupo de *securities*

escolhido, dependendo do agrupamento relevante ao negócio.

3.3.2 Conjunto de Tarefas

3.3.2.1 Work Breakdown Structure

A estrutura da divisão do trabalho proposta pelo WBS prevê o sistema, a partir do resultado final preterido, como uma estrutura em árvore relativa à quebra do sistema em subsistemas menores. A regra dos 100% afirma que a soma do trabalho dos filhos deve equivaler a exatamente 100% do trabalho do pai (PROJECT MANAGEMENT INSTITUTE, 2006), de modo que ao final da quebra, não haja trabalho além do escopo e nem parte do escopo não contemplado por algum item. Além disso, todos os itens em um mesmo nível da hierarquia devem ser mutuamente exclusivos para que não haja trabalho duplicado ou indefinição quanto a responsabilidades.

O WBS é orientado para os entregáveis do projeto, com a finalidade de apresentar a evolução do projeto, mitigando assim riscos relacionados à maturidade do mesmo. Para estruturas que subdividem o trabalho total por fases de projeto, como é o caso do Processo Unificado, deve-se assegurar que cada fase seja separada das outras por um entregável utilizado como critério de transição.

A seguir, apresenta-se um planejamento das atividades definidas utilizando-se o WBS a partir dos entregáveis relativos às fases do Processo Unificado e ao gerenciamento de projeto. A cada entregável correspondem atividades para a sua geração. Este planejamento ainda inclui estimativas de tempo e custo dos pacotes de trabalho (elementos de maior nível de detalhe do WBS) que são utilizadas posteriormente neste trabalho:

- Planejamento:

1. Planejamento de Projeto: Apresenta a análise de riscos, atividades do projeto, equipe e estimativas de custo e prazo, entre outros.

- Conceção:

1. Resumo do Projeto: Apresenta objetivos e escopo preliminar.
2. Especificação de Requisitos: Apresenta requisitos funcionais e não-funcionais, restrições, características dos usuários, interfaces e critérios de aceitação.

- Elaboração:

1. Projeto de IHC: Apresenta o projeto das telas do usuário final e a navegação entre elas.
 2. Modelo de Casos de Uso: Apresenta o relacionamento entre o sistema e tudo que atua sobre ele.
 3. Modelo de Classes: Apresenta a modelagem das classes do sistema.
 4. Arquitetura do Sistema: Apresenta os componentes do sistema e sua interdependência.
 5. Modelo de Banco de Dados: Apresenta o modelo de dados a ser desenvolvido.
 6. Modelo Dinâmico: Apresenta o comportamento dinâmico do sistema através de máquinas de estados e diagramas de sequência.
 7. Plano de Testes: Apresenta a estratégia de testes do sistema.
- Construção:
 1. Protótipo mínimo do sistema: Protótipo para apresentar o funcionamento dos componentes críticos do sistema, incluindo a integração deles.
 2. Camada de dados: Camada de dados e sua interface com a camada de negócios.
 3. Interfaces com sistemas externos: Integração do sistema com fontes de dados externas.
 4. Camada de negócios: Camada de negócios com as regras levantadas junto ao cliente.
 5. Interface com o usuário: Proposta final da interface com o usuário a partir do protótipo criado anteriormente.
 6. Módulo funcional unitário: Componentes críticos da camada de negócios funcionais.
 7. Módulos funcionais: Componentes da camada de negócios funcionais.
 8. SPCRCA v1.0 Alpha 1: Sistema funcional integrado antes dos testes finais.
 - Transição:
 1. SPCRCA v1.0 Alpha 2: Sistema funcional depois dos testes modulares finais.
 2. SPCRCA v1.0 Alpha 3: Sistema funcional depois dos testes de integração finais.
 3. SPCRCA v1.0 Release Candidate 1: Sistema funcional depois dos testes automáticos e testes alfa.

4. SPCRCA v1.0 Build 001: Sistema funcional homologado.

3.4 Gerenciamento de Tempo e de Custo

3.4.1 Estimativas

3.4.1.1 Técnica Top-Down

A estimativa *top-down* (também conhecida como Estimativa por Analogia) utiliza como base uma atividade de algum projeto já realizado análoga à atividade a ser realizada no projeto atual, comparando-se as proporções de ambos em relação a fatores como tamanho e complexidade. Desta forma, esta técnica é utilizada na estimativa de um parâmetro quando a informação necessária para o cálculo é inexistente ou indisponível no momento. Para o SPCRCA, ela foi utilizada no início do trabalho para verificar o tempo disponível para a realização do mesmo como forma de verificar a viabilidade do escopo proposto, dado o prazo definido. Levantando o número de horas, tem-se 8 meses com dedicação de 4 horas semanais e 4 meses com dedicação de 12 horas semanais para cada um dos três integrantes da equipe. Com isso, o total de horas do projeto foi estimado em 960 homens-hora.

3.4.1.2 Técnica Bottom-Up

A técnica de estimativa *bottom-up* permite uma estimativa mais refinada de um certo componente de trabalho ou do trabalho como um todo. Ela, assim como as outras técnicas, pode ser utilizada tanto para a análise de tempo quanto de custo do projeto. Nela, cada tarefa é dividida, e cada uma dessas divisões são estimadas individualmente. Os resultados em seguida são agregados para se obter o total do trabalho. Neste caso, o custo total foi estimado em 1003,8 homens-hora. No apêndice A, apresenta-se a estimativa de tempo de cada uma das atividades do WBS.

3.4.1.3 Resultados Consolidados

O total de horas utilizando-se a técnica *bottom-up* foi prevista para 1003,8 horas, enquanto para a estimativa por analogia foi de 960 horas. Tem-se, portanto, uma média de 993,9 horas, com desvio padrão de 14,0 horas. Verificou-se, portanto, que as estimativas

apresentaram resultados semelhantes e também que o tempo efetivo gasto com o projeto mostrou-se condizente com o estimado.

3.4.2 Recursos do Projeto

3.4.2.1 Recursos Humanos

A empresa fornecedora do sistema organizou a equipe com os papéis de gerente de projetos, arquiteto de sistema, analista de negócio e desenvolvedores. Desta forma, a atribuição dos papéis fica mapeada conforme a tabela 2.

Tabela 2: Distribuição de papéis entre os recursos humanos

Função	Membro da Equipe		
	AKH	GP	RP
Gerente de Projeto	x		x
Arquiteto de Sistema	x		x
Analista de Negócio		x	
Desenvolvedor	x	x	x

Além disso, a empresa cliente deve dispor de um gerente para validação do plano de negócios, um arquiteto para validação da especificação e dos modelos gerados, e uma equipe de testes para homologação tanto das funcionalidades quanto dos dados gerados.

3.4.2.2 Recursos de Software

Os ambientes de desenvolvimento devem possuir sistema operacional Windows XP ou Vista, com o Visual Studio 2005 utilizando o *framework* Microsoft.NET versão 2.x e banco de dados MySQL versão 5.x. Além disso, para o controle de versões, foi utilizado o SVN. Já para homologação e produção, os clientes devem possuir sistema operacional Windows XP ou Vista, e os servidores de banco de dados devem possuir o gerenciado MySQL versão 5.x. Além disso, também serão necessárias as bibliotecas NUnit 2.4.7 para testes unitários do sistema, e o Zedgraph 5.1.4 para a construção de gráficos.

3.4.2.3 Recursos de Hardware

Os recursos de hardware necessários para o desenvolvimento são *desktops* para desenvolvimento e servidor para o SVN, além de estrutura de rede. Para homologação e

produção, é necessário um servidor de banco de dados, com os usuários acessando-o a partir de máquinas clientes nas quais o software estará distribuído.

3.5 Gerenciamento de Risco

3.5.1 Levantamento dos Riscos

Para uma análise mais estruturada, é necessário dividir o sistema de acordo com as fontes de risco, de modo a categorizá-los sistematicamente. Um mapeamento abrangente consiste na divisão em:

- Riscos técnicos: Engloba os requisitos, a tecnologia, a complexidade e outros aspectos de vertente técnica;
- Riscos externos: Engloba fornecedores, regulamentos, mercados e clientes;
- Riscos organizacionais: Engloba recursos humanos e materiais, e priorização dentro da organização, entre outros;
- Riscos gerenciais: Engloba estimativas, controle, envolvimento dos *stakeholders* e comunicação.

Quanto aos riscos técnicos, deve-se levar em conta completeza, clareza, validade e viabilidade dos requisitos, funcionalidade, desempenho e outras restrições na análise e design do produto e possíveis problemas e dificuldades no desenvolvimento e nos testes do sistema, observando sempre a qualidade do produto implementado. Foram identificados os seguintes riscos:

- Mudança de escopo;
- Falta de clareza e completeza dos requisitos;
- Falta de clareza e completeza da modelagem;
- Não identificação dos requisitos não-funcionais;
- Falta de entendimento das regras de negócio;
- Erros não-previstos de implementação;

- Ambiente de desenvolvimento não adequado.

Em relação aos riscos externos, foram consideradas como fontes de riscos os seguintes itens:

- Falhas nos sistemas externos;
- Falta de agilidade no processo de homologação.

Os riscos gerenciais remetem a erros que comprometam a disponibilidade dos recursos definidos anteriormente. São eles:

- Subestimativa da implementação;
- Processos internos falhos;
- Falta de comunicação inter e intra-equipes.

Deve-se notar que os riscos organizacionais não foram analisados pelo fato de a empresa fornecedora do sistema não se inserir dentro do contexto de uma organização.

3.5.2 Análise de Criticidade e Estratégia de Resposta

A partir dos riscos identificados no item anterior, estima-se a probabilidade e o impacto que pode vir a causar, para cada um dos riscos. O impacto é estimado como a porcentagem do tempo total de projeto que será afetado, caso o risco identificado venha de fato a ocorrer (tabela 3). Correlacionando-se esses dois indicadores, é possível obter a matriz de exposição dos riscos (figura 3), uma representação gráfica obtida a partir da plotagem de cada risco como um ponto em um gráfico de probabilidade em função do impacto, permitindo assim analisar os riscos que ameaçam o projeto.

Além disso, definem-se ações a serem tomadas na eventual ocorrência de cada risco também na tabela 3. As ações são classificadas em: mitigar, quando a consequência da ocorrência de um risco é diretamente trabalhada; evitar, quando é possível que ações anteriores possam ser tomadas para diminuir a probabilidade; aceitar, quando não há ações anteriores ou posteriores que possam diminuir o impacto; e transferir, quando o risco for pertinente a terceiros.

Tabela 3: Matriz de resposta aos riscos

	Risco	Probab. Impacto		Ação Requerida
1	Mudança de escopo	80%	90%	Mitigar: Analisar impacto e aceitar ou recusar mudança
2	Falta de clareza e completeza dos requisitos	70%	80%	Mitigar: Analisar constantemente a qualidade dos requisitos, modificando quando necessário
3	Falta de clareza e completeza da modelagem	50%	70%	Mitigar: Analisar constantemente a qualidade da modelagem, modificando quando necessário
4	Não aplicação dos requisitos não-funcionais	30%	50%	Evitar: Projetar a arquitetura de modo a minimizar futuros problemas com requisitos não funcionais
5	Falta de entendimento das regras de negócio	50%	80%	Evitar: Documentar as regras de negócio junto dos requisitos e validar
6	Erros não-previstos de implementação	50%	50%	Mitigar: Consertar os erros
7	Ambiente de desenvolvimento não adequado	30%	20%	Evitar: Adequar o ambiente antes do início do projeto
8	Falhas nos sistemas externos	10%	50%	Transferir: Analisar junto à parte responsável a causa da falha
9	Falta de agilidade no processo de homologação	40%	30%	Evitar: Garantir alocação de recursos
10	Subestimativa dos recursos para implementação	70%	50%	Mitigar: Analisar constantemente a conformidade com o cronograma, alocando recursos para minimizar impactos
11	Processos internos falhos	10%	10%	Evitar: Treinar os envolvidos quanto aos processos internos
12	Falta de comunicação inter e intra-equipes	60%	60%	Evitar: Planejar canais de comunicação bem definidos

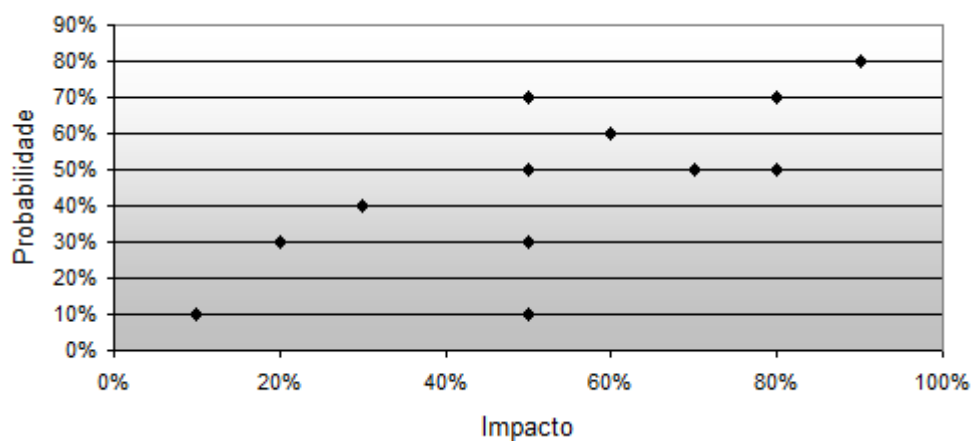


Figura 3: Matriz de exposição aos riscos

3.6 Considerações Finais

Este capítulo apresentou as técnicas utilizadas no planejamento do sistema. Dado o escopo definido junto ao cliente, o sistema foi dividido em tarefas pelo WBS, identificando-se os entregáveis relevantes para cada fase do Processo Unificado e, em seguida, dividido em atividades. Com esta divisão do trabalho, os recursos necessários para a realização do projeto foram estimados com maior precisão, o que facilitou a identificação, análise e mitigação de possíveis riscos. Desta forma, minimizou-se a competição entre os componentes do tripé da gestão de projetos, formado pelo escopo, custo e tempo (ver seção 2.5.1), através de uma abordagem que procurou prever suas possíveis causas.

4 *Concepção*

*“Quem conduz e arrasta o mundo não são
as máquinas, mas as idéias.”*

Victor Hugo

4.1 Introdução

Neste capítulo são levantados os principais requisitos funcionais e não-funcionais do sistema, gerando a Especificação de Requisitos ao fim da fase do Processo Unificado correspondente.

4.2 Requisitos Funcionais

Devido aos requisitos não-funcionais, como escalabilidade e flexibilidade, citados anteriormente, vê-se a importância de considerar a arquitetura do sistema logo nas fases iniciais do ciclo de vida do software, inclusive havendo interações entre as fases de Concepção e de Elaboração (MEI, 2000). As funções a seguir apresentam alguma maturidade em relação à arquitetura provida por essas interações, de forma a garantir maior consistência.

4.2.1 Funções de Negócios

As funções de negócio compreendem a análise de precificação e risco da carteira de ativos. Podem ser separadas em cinco frentes, sendo elas a própria precificação (ou *Market Pricing*), a análise de risco de mercado (ou *Market Risk*), a precificação histórica (ou *Historical Pricing*), o *Market Scenario* e os fluxos de caixa (ou *Cash Flows*).

4.2.1.1 Market Pricing and Risk

O *Market Pricing* deve definir o valor de uma *security* ou grupo de *securities* escolhido pelo usuário do sistema para uma dada data, podendo este valor ser expresso em diferentes moedas. O *Market Risk*, por sua vez, deve calcular a medida da variação do preço devido às flutuações das variáveis de mercado, como câmbio USD/BRL, curva de juros, curva de volatilidade e preços de ações também para *securities* individuais ou grupo de *securities*. Estas duas análises podem ser realizadas conjuntamente, de modo a facilitar a visualização por parte do usuário final do sistema.

4.2.1.2 Market Scenario

O *Market Scenario* deve permitir a simulação de uma situação do mercado através da manipulação das variáveis relevantes. Para isso, ao escolher uma ou mais *securities*, é dada a opção de se aplicar modificadores aos valores como o preço, por exemplo, e em seguida plotar gráficos que mostram a situação do mercado sob estas novas condições.

4.2.1.3 Historical Pricing

O *Historical Pricing*, como o próprio nome diz, deve calcular o preço de uma *security* em um determinado intervalo de tempo, exibindo-o em um gráfico. O gráfico pode ou não ser atualizado em tempo real com dados provenientes do mercado, cabendo ao usuário esta decisão.

4.2.1.4 Cash Flows

O fluxo de caixa (ou *Cash Flow*) deve mostrar, para as *securities* escolhidas, o montante recebido e pago em uma forma gráfica, ressaltando os valores em uma linha de tempo. Deve ser possível identificar quais fluxos pertencem a qual *security*, no caso de haver mais de uma plotada.

4.2.2 Funções de Infra-Estrutura

As funções de infra-estrutura englobam requisitos não ligados diretamente ao negócio, mas que são imprescindíveis para o funcionamento do sistema.

4.2.2.1 Títulos e Valores Mobiliários

Ações

O valor das ações, para qualquer data, deve estar de acordo com os preços praticados pelo mercado. Além do preço da ação, o sistema deve ser capaz de identificar o histórico de pagamento de dividendos da empresa, sendo estes ajustados de acordo com os eventos de Bonificação, Desdobramento e Agrupamento que ocorrerem após o pagamento de dividendos.

Contratos Futuro

O sistema deve utilizar os preços de fechamento registrados na BM&F, mas efetua o cálculo de ajuste diário por conta própria. Para cada tipo de contrato há uma determinada forma de se calcular o ajuste diário.

Título de Dívida

Os títulos de dívida, pública ou privada, são criados pelo próprio usuário através de inserção de fluxos, sendo estes dos seguintes tipos:

- *Simple Cash Flow*: Fluxo monetário simples, não correlacionado com taxa de juros, ocorrendo em uma determinada data. Por exemplo, pagamento de R\$1.000,00 em 23 de janeiro de 2008.
- *Fixed Cash Flow*: Fluxo atrelado a uma taxa de juros fixa incidindo sobre um principal. Por exemplo, 15% de R\$10.000,00 a receber em 04/07/2009 contando a partir de 04/08/2008.
- *Float Cash Flow*: Fluxo atrelado a uma taxa de juros, praticada no mercado, que varia dia-a-dia incidindo sobre um principal. Por exemplo, pagamento de 110% da taxa CDI sobre R\$10.000,00 em 04/07/2009 contando a partir de 04/08/2008.

Com esta abordagem, o sistema deve ser capaz de representar qualquer título de dívida, incluindo ativos que não se encaixam nesta categoria, como NDF e *swap*.

4.2.2.2 Trade Builder e Security Builder

Com os módulos *Trade Builder* e o *Security Builder*, deve ser possível a construção de *trades* através da escolha do tipo de *trade* e de atribuição de variáveis pertinentes, ou de *securities* através da união de diversos fluxos de caixa. As *trades* e *securities* construídas devem poder ser salvas e carregadas para posterior edição, e serão utilizadas nos outros módulos do sistema.

4.2.2.3 Book View

Deve haver também um módulo *Book View*, que permite a manipulação de carteiras de ativos. A carteira deve ter um formato de árvore, podendo as *securities* adicionadas ser classificadas em tipos definidos pelo usuário. Estes tipos correspondem a nós da árvore, e podem ser reordenados em novas hierarquias. Deve ser possível salvar e carregar as carteiras, respeitando nestas operações o modo como a hierarquia foi montada.

4.2.2.4 Autenticação

A autenticação deve permitir que apenas pessoas autorizadas tenham acesso a informações e módulos restritos. Além disso, deve ser possível também bloquear as funcionalidades do aplicativo tanto por escolha do usuário quanto por *timeout*, de modo que só seja possível retornar à utilização do sistema quando a senha for inserida novamente.

4.3 Requisitos Não-Funcionais

A seguir, apresentam-se os requisitos não-funcionais definidos junto ao cliente. A seguir, apresentam-se os requisitos não-funcionais definidos junto ao cliente. Cuidados especiais quanto a estes requisitos serão abordados na modelagem (capítulo 5) e seus testes serão melhor descritos no capítulo 7.

4.3.1 Qualidade de Processo

Define-se confiabilidade como a probabilidade de um sistema funcionar conforme especificado durante um período de tempo (BHATTI, 2005). O padrão 982.2 da IEEE (IEEE, 1988) afirma que “um programa de gerenciamento de confiabilidade de software requer que um conjunto balanceado de objetivos de qualidade do usuário seja estabelecido, e objetivos de qualidade intermediários que ajudarão a alcançar os objetivos de qualidade do usuário sejam identificados”.

Dado que este requisito depende fortemente da qualidade do sistema, identificam-se então fatores de qualidade que impactam nele, para as diversas fases do ciclo de vida do software (ROSENBERG; SHAW, 1998):

- Concepção:
 - Necessidades e objetivos dos usuários: Funcionalidade, desempenho, completude e consistência;
 - Padronização da documentação;
 - Requisitos: Aderência às necessidades, arquitetura, ambiente operacional, completude e facilidade de uso.
- Elaboração: Complexidade, modularidade, interfaces, expansibilidade, tempo, tamanho e completude.

- Construção: Complexidade, interfaces, padrões para desenvolvimento, completeza e manutenibilidade.
- Transição:
 - Testes: Cobertura funcional, interfaces de componentes, medidas de desempenho;
 - Instalação: Realismo operacional, cobertura da configuração e interfaces hardware/software;
 - Operação e manutenção: Integridade das mudanças, facilidade de uso e de aprendizado.

Espera-se que, com esforços em todas as fases, ou seja, com o processo de desenvolvimento voltado para a qualidade, o resultado final apresente confiabilidade satisfatória. Além disso, a confiabilidade pode ser medida quantitativamente através, por exemplo, de relatórios que são criados na ocorrência de erros.

4.3.2 Configurabilidade da Interface

A configurabilidade indica que o sistema deve ser customizável pelo usuário. Isso é atingido através de funcionalidades incorporadas ao sistema de acordo com os requisitos de interface levantados.

4.3.3 Desempenho

O sistema deve possuir um bom desempenho para processar cálculos complexos, com tempo de resposta respeitando um máximo tolerável. Ao invés de se utilizar abordagens como *benchmarks*, o desempenho refere-se ao nível de satisfação do usuário quanto ao tempo de resposta. Obviamente, aspectos objetivos e mensuráveis que venham a afetar diretamente o desempenho, como utilização da memória, carregamento da rede, comportamento das filas e do banco de dados, são analisados.

4.3.4 Disponibilidade

Definindo-se disponibilidade como a probabilidade de um sistema funcionar em um determinado instante, tem-se que o SPCRCA requer alta disponibilidade. Isto significa que

ele deve ter alta qualidade e a recuperação de falhas deve acontecer rapidamente, caso esta seja interna ao sistema. No caso de falhas com as fontes de dados externas, o usuário deve ser notificado da indisponibilidade.

4.3.5 Escalabilidade

O sistema deve ser escalável quanto ao tamanho da carteira e a quantidade de dados em tempo real. O aumento não deve implicar em queda brusca de desempenho nem deixar de funcionar segundo a especificação. Este requisito deve ser previsto pela arquitetura do sistema, e pode ser testado através de testes de aumento de carga.

4.3.6 Manutenibilidade

O sistema deve ser projetado para que sua manutenção seja realizada de forma rápida e eficiente. Para isso, os modelos e documentos devem ser atualizados ao longo do desenvolvimento e ser consistente em relação ao código, a arquitetura deve ser feita com componentes de baixo acoplamento e alta coesão e deve haver documentos de teste.

4.3.7 Segurança de Acesso

Como o sistema apresenta informações que devem ser protegidas de acesso indevido, os dados devem ser criptografados e o acesso deve ser restrito, necessitando de autenticação. Desta forma, certas funções podem ser utilizadas apenas por certos perfis de usuário.

4.3.8 Testabilidade

O sistema deve conter testes automáticos que verificam o bom funcionamento do sistema e a cada disfunção encontrada cria-se um relatório. No caso de uma falha insegura, deve avisar o usuário sobre tal.

4.3.9 Usabilidade

O sistema deve ser de fácil utilização pelos usuários, com uma interface agradável e intuitiva, e com eficiência na utilização. A usabilidade deve ser mensurável através de

níveis de satisfação.

4.4 Características dos Usuários

Os usuários previstos para o sistema podem ser classificados em três tipos. Os *structurers* são usuários comerciais que trabalham diretamente com o cliente para estruturas as *securities*. É permitido abrir, criar e salvar *securities*, mas sem adicioná-las na carteira. Os *traders* gerenciam o risco das carteiras, em específico as *securities* que pertencem ao mesmo tipo do *trader*. Por fim, o *product controller* solicita relatórios de risco da carteira, acessando-a por inteira, mas com permissão de leitura apenas. Observa-se que todos os usuários possuem alta escolaridade, tendo no mínimo curso de graduação, e experiência tanto nas suas áreas de atuação quanto em informática.

4.5 Interfaces

4.5.1 Interface com o Usuário

O sistema deve ser *desktop*, composto por múltiplas janelas, sendo elas pertencentes a diferentes áreas de trabalho. Ao mudar de área de trabalho, o aplicativo deve mostrar apenas as janelas referentes à área de trabalho em questão.

4.5.2 Interfaces de Comunicação

O sistema faz uso de protocolos de rede local TCP/IP e a comunicação com o servidor de dados em tempo real será feita através de *message queuing*.

4.6 Restrições

Primeiramente, o sistema deve ser desenvolvido sobre o *framework* Microsoft.NET (em específico, a linguagem C#) e utilizando o gerenciador de banco de dados MySQL, de acordo com o padrão já adotado pelo cliente. Segundo, devido à necessidade de o sistema ser em tempo real (atualizando-se a cada dado recebido da bolsa de valores), a aplicação não deve ser web, uma vez que a conexão cliente/servidor não poderá ser mantida, além do fato do custo de desempenho envolvido. Por fim, a comunicação entre sistemas deve

ser feita por *message queuing*, uma vez que esta solução já está implantada entre outros sistemas do cliente. Esse método permite um sistema continuar consumindo suas filas caso venha a falhar, aumentando a disponibilidade.

4.7 Dependências

O sistema depende da disponibilidade da fonte de dados e do sistema *publisher-subscriber*. O mau funcionamento de um destes componentes implica o não cumprimento das funções do software, devendo-se neste caso haver alerta para o usuário da falha ocorrida.

4.8 Considerações Finais

Na fase de Concepção, entre 70% a 80% dos requisitos funcionais e 100% dos não-funcionais foram definidos, já prevendo um melhor detalhamento dos requisitos na próxima fase do Processo Unificado à medida que a maturidade quanto ao sistema final aumente tanto por parte do cliente quanto da equipe de projeto.

Além disso, devido aos requisitos não-funcionais identificados, é imperativo que se houvesse uma visão voltada para a arquitetura no início do ciclo de vida do sistema, sendo esta também uma motivação maior para que a revisão dos requisitos durante a Elaboração seja constante, uma vez que a discussão sobre a arquitetura é iniciada nesta fase. Estes requisitos necessitam também de outros cuidados na modelagem (ainda na fase de Elaboração) e de testes especiais (na fase de Transição).

5 *Elaboração*

*“A arquitetura é uma ciência, surgindo de muitas
outras, e adornada com muitos e variados
ensinamentos: pela ajuda dos quais um julgamento é
formado daqueles trabalhos que são o resultado
das outras artes.”*

Vitrúvio

5.1 Introdução

Obtêm-se, com os tópicos descritos nas seções seguintes, o Projeto de Interface Homem-Computador (IHC), o Modelo de Casos de Uso, o Modelo de Classes, o Modelo Dinâmico, a Arquitetura de Sistema, o Modelo de Dados e os critérios de aceitação.

5.2 Interface Homem-Computador

A fase de Elaboração foi iniciada a partir da interface com o usuário, devido à pouca maturidade em relação ao produto final por parte tanto do cliente quanto do fornecedor. A abordagem de gerar um protótipo inicial da interface antes dos modelos de casos de uso e de classes tem por objetivo consolidar a idéia em relação ao sistema final por parte de todos os interessados do projeto, auxiliando a visualização do que é previsto para o produto, e formalizar com maior detalhe os requisitos do sistema.

5.2.1 Caracterização dos Usuários

5.2.1.1 Structurer

É o usuário comercial que estrutura os *trades*. É capaz de criar *trades* complexos e analisá-los utilizando os módulos do SPCRCA.

- Funções permitidas:
 - Criar, editar, salvar, abrir e excluir *trades* e *securities*.
 - Analisar *trades*.
 - Pode criar e gerenciar uma carteira falsa.
- Funções não permitidas:
 - Abrir, adicionar ou gerenciar *trades* na carteira real.

5.2.1.2 Trader

Gerencia o risco da carteira. Usuários do tipo *Trader* podem ser especializados em certos tipos de *trades*.

- Funções permitidas:
 - Adicionar *trade* criado pelo usuário do tipo *Structurer* na carteira.
 - Gerenciar carteira.
 - Analisar *trades*.
- Funções não permitidas:
 - Criar ou editar *trades* e *securities*.

5.2.1.3 Product Control

Gera relatório de risco da carteira, acessando-a por inteira. Tem por função verificar a boa gestão da carteira.

- Funções permitidas:
 - Acessar carteira (apenas leitura).
 - Analisar *trades*.
- Funções não permitidas:
 - Modificar carteira.
 - Criar, editar, salvar, abrir e excluir *trades* e *securities*.

5.2.2 Estilo da Interface

Devido aos requisitos de interfaces homem-computador descritos em 4.5.1, vê-se a necessidade de se utilizar *Windows Forms* com múltiplas janelas, de modo a suportar diversas áreas de trabalho.

5.2.3 Protótipo de Navegação

A figura 4 mostra o diagrama de navegação do sistema. Pelo módulo principal, é possível abrir todas as outras telas. Selecionando *trades* ou pastas na tela *Book View*, pode-se enviá-los para as telas *Historic Pricing*, *Market Scenario*, *Cash Flows* e *Market Pricing and Risk* para que as devidas análises sejam realizadas.

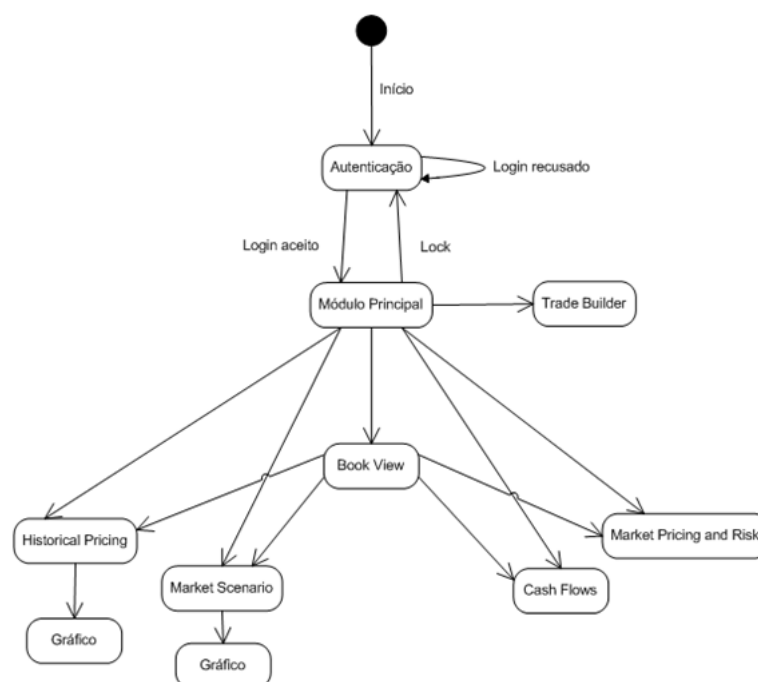


Figura 4: Diagrama de navegação

5.2.4 Protótipo de Telas

O protótipo de telas determina a interface de cada tela e suas funcionalidades, e permite testar a navegabilidade proposta. Mais detalhes do protótipo de telas pode ser encontrado no protótipo de telas, presente no apêndice B.

5.3 Modelo de Casos de Uso

Levando em consideração os tipos de usuários, suas características e o protótipo da interface levantados na seção anterior, é possível iniciar com maior maturidade a modelagem dos casos de uso. Na figura 5 está a representação do relacionamento entre os atores e os principais casos de uso.

Os casos de uso estão resumidos a seguir:

- *CRUD Trade Builder*: Este caso de uso prevê a criação, edição, visualização e exclusão de um *trade* para usuários já autenticados no sistema que selecionaram a opção *Trade Builder* no menu principal. É prevista também a customização das configurações durante operações de criação e edição.
- *CRUD Secutiry Builder*: Este caso de uso prevê a criação, edição, visualização e

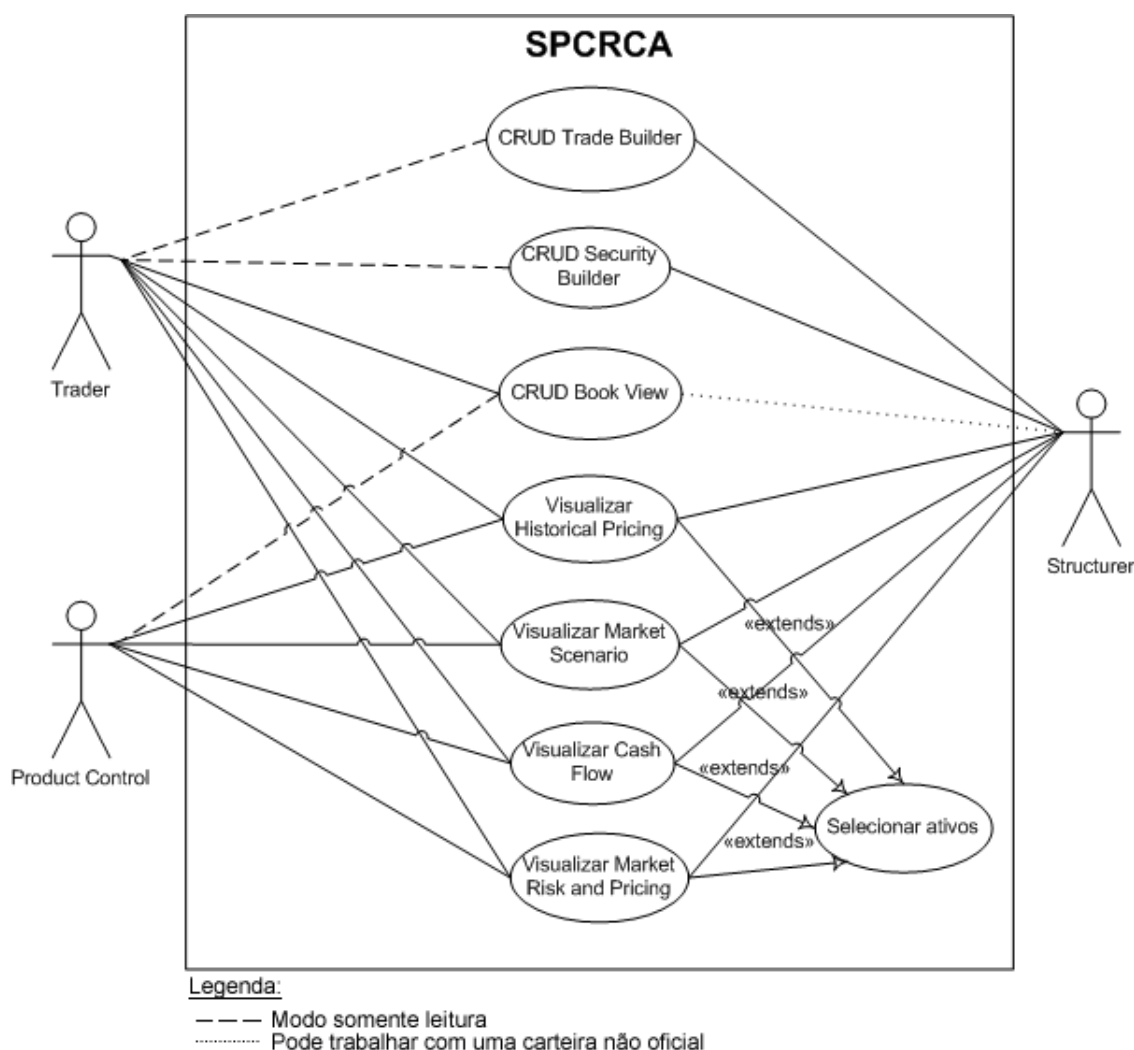


Figura 5: Diagrama do modelo de casos de uso

exclusão de uma *security* para usuários já autenticados no sistema que selecionaram a opção *Security Builder* no menu principal. É prevista a customização completa da *security*, indicando-se os fluxos de caixa que a compõem, para as operações de criação e edição.

- *CRUD Book View*: Este caso de uso prevê a criação, edição, visualização e exclusão de uma carteira de ativos para usuários já autenticados no sistema que selecionaram a opção *Book View* no menu principal. É prevista a customização dos ativos nesta carteira através de sua adição, exclusão e agrupamento.
- *Visualizar Historical Pricing*: Este caso de uso prevê a visualização da precificação histórica de ativos selecionados, para usuários já autenticados no sistema que selecionaram a opção *Historical Pricing* no menu principal.
- *Visualizar Market Scenario*: Este caso de uso prevê a visualização de um cenário

de mercado para ativos selecionados, para usuários já autenticados no sistema que selecionaram a opção *Market Scenario* no menu principal. É prevista a configuração das variáveis de mercado de modo a conseguir o cenário desejado.

- Visualizar *Cash Flow*: Este caso de uso prevê a visualização dos fluxos de caixa para ativos selecionados, para usuários já autenticados no sistema que selecionaram a opção *Cash Flow* no menu principal.
- Visualizar *Market Pricing and Risk*: Este caso de uso prevê a visualização do preço e de indicadores de risco relevantes para ativos selecionados, para usuários já autenticados no sistema que selecionaram a opção *Market Pricing and Risk* no menu principal. É prevista a possibilidade de reagrupamento dos ativos e de mudança da moeda na qual os valores são exibidos.
- Selecionar ativos: Este caso de uso prevê a seleção de um ativo ou grupo de ativos para sua posterior utilização nos demais módulos funcionais do sistema.

5.4 Arquitetura do Sistema

O sistema SPCRCA tem sua organização lógica composta de três grandes blocos: *Application*, *Business* e *Framework*. Os componentes de *Application* referem-se às funcionalidades diretamente ligadas à aplicação. Devem mapear os requisitos funcionais relacionados às análises financeiras, provendo desde a interface com o usuário até comunicação entre módulos funcionais da aplicação. Os componentes de *Business*, por sua vez, captam as funções de negócio no que diz respeito ao suporte às análises, provendo os dados necessários. Por fim, os componentes de *Framework* fornecem suporte a outros dois blocos por meio de interfaces e classes de uso geral. Esta divisão em blocos tem por objetivo garantir o desacoplamento e a reusabilidade das classes construídas. O bloco *Framework* é completamente independente dos demais blocos, não possuindo referências aos demais blocos. O *Business* pode utilizar o *Framework*, mas não pode utilizar o *Application*, e o bloco *Application* pode acessar os demais blocos. A figura 6 mostra os blocos e como eles interagem. Esses blocos serão abordados nas próximas seções.

O SPCRCA possui vários módulos, que devem trocar mensagens entre si. Por exemplo, o módulo *Book View*, ao selecionar um *trade*, pode-se enviá-lo para o módulo *Historic Pricing* para que o gráfico do seu valor seja plotado em função do tempo. Desta forma, e também para garantir a escalabilidade, modularidade, padronização e consistência do

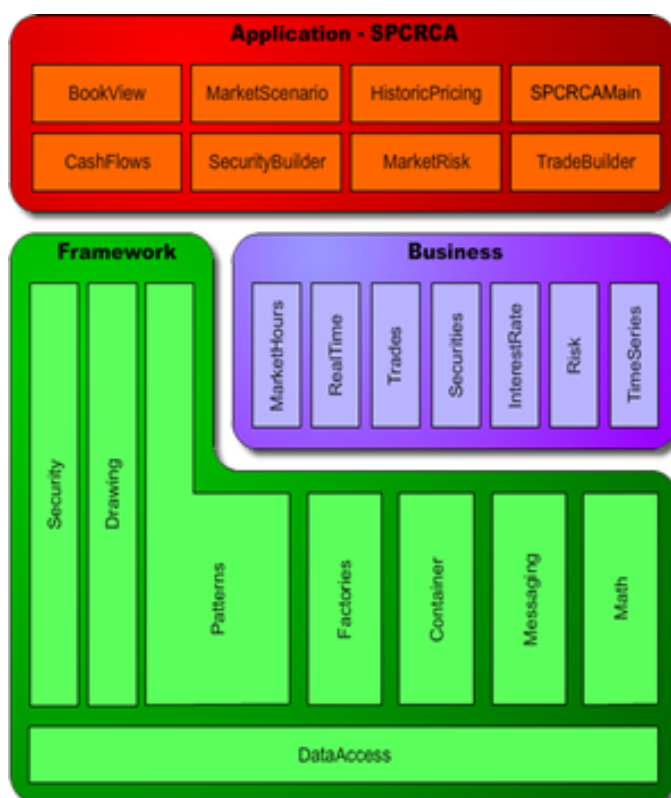


Figura 6: Visão lógica da arquitetura: Blocos *Application*, *Business* e *Framework*

sistema, viu-se a necessidade de se seguir um padrão para a arquitetura.

A primeira *architectural pattern* pesquisada para ser utilizada foi o MVC (BUSCHMANN et al., 1996), sendo esta a escolha óbvia devido à sua ampla utilização para sistemas de janelas. Contudo, esta se mostrou muito limitada para o que era necessário, já que o fato de possuir uma única classe *Model* impede que um módulo do sistema seja separado e utilizado em outro sistema devido às referências feitas. O MVC se adequa muito bem quanto à separação das camadas de interface, de tratamento de eventos e de modelo, mas ela não esclarece como a comunicação e a organização entre módulos deve ser feita. Pesquisou-se então a *architectural pattern* PAC (BUSCHMANN et al., 1996). Esta possui vantagens em relação ao MVC, embora ainda assim apresente-se limitada a ponto de não atender exatamente o que era necessário (mudanças feitas na *design pattern* PAC para total adequação quanto às necessidades são descritas na seção 5.5.1.3). Por exemplo, embora o MVC em geral apresente maior eficiência, o PAC tem melhor suporte para multitarefa e interfaces com o usuário que executam tarefas específicas (BUSCHMANN et al., 1996).

O PAC descreve como separar as mesmas três camadas do MVC, só que por módulos denominados agentes. Cada agente possui três componentes: *presentation*, *abstraction* e *control*, sendo que a comunicação entre diferentes agentes é feita entre as interfaces de

controle. Estas três camadas são suficientes para o funcionamento do agente, havendo ainda a possibilidade de se montar uma hierarquia de agentes em árvore, como mostra a figura 7.

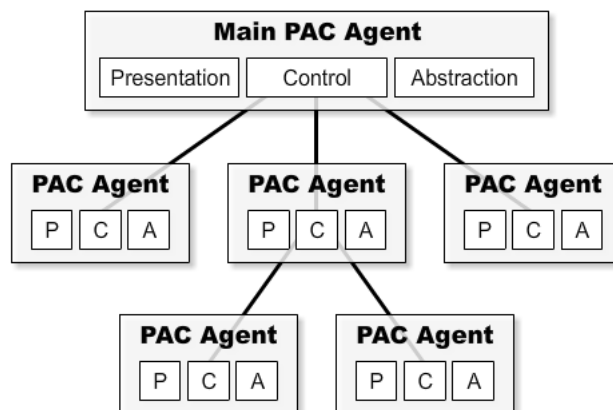


Figura 7: Visão física da arquitetura: Estrutura do PAC em tempo de execução

A organização física do SPCRCA é feita implementando-se o PAC como uma hierarquia de agentes de dois níveis. A raiz da árvore é o módulo principal da aplicação, e todos os outros módulos criados posteriormente em tempo de execução são seus filhos.

5.5 Modelagem do Sistema

Procurou-se utilizar diversos *design patterns* na modelagem do sistema, entretanto a especificação das *design patterns* existente atualmente proporciona problemas quanto à ambiguidade de suas definições (MAK; CHOY; LUN, 2004) (TAIBI, 2006). Para isso, viu-se a necessidade uma modelagem precisa, de modo que o formalismo elimine preocupações a respeito da utilização desses padrões.

Esta seção descreve a modelagem específica dos principais *packages* do sistema, que podem ser encontrados na figura 6, além de mostrar os diversos problemas técnicos enfrentados durante o processo de desenvolvimento, e as respectivas soluções adotadas.

5.5.1 Framework

O bloco do *Framework* (ver arquitetura na figura 6) possui *packages* que servem de apoio aos demais blocos, podendo ser reutilizado em qualquer outro projeto por possuir uma natureza genérica. Os *packages* mais importantes para o funcionamento do sistema

que são *Factories*, *Patterns*, *DataAccess* e *Security* (segurança de acesso) serão detalhados a seguir. O PAC pertence ao *package Patterns*, mas por sua elevada importância no sistema, possui uma seção própria.

5.5.1.1 Factories

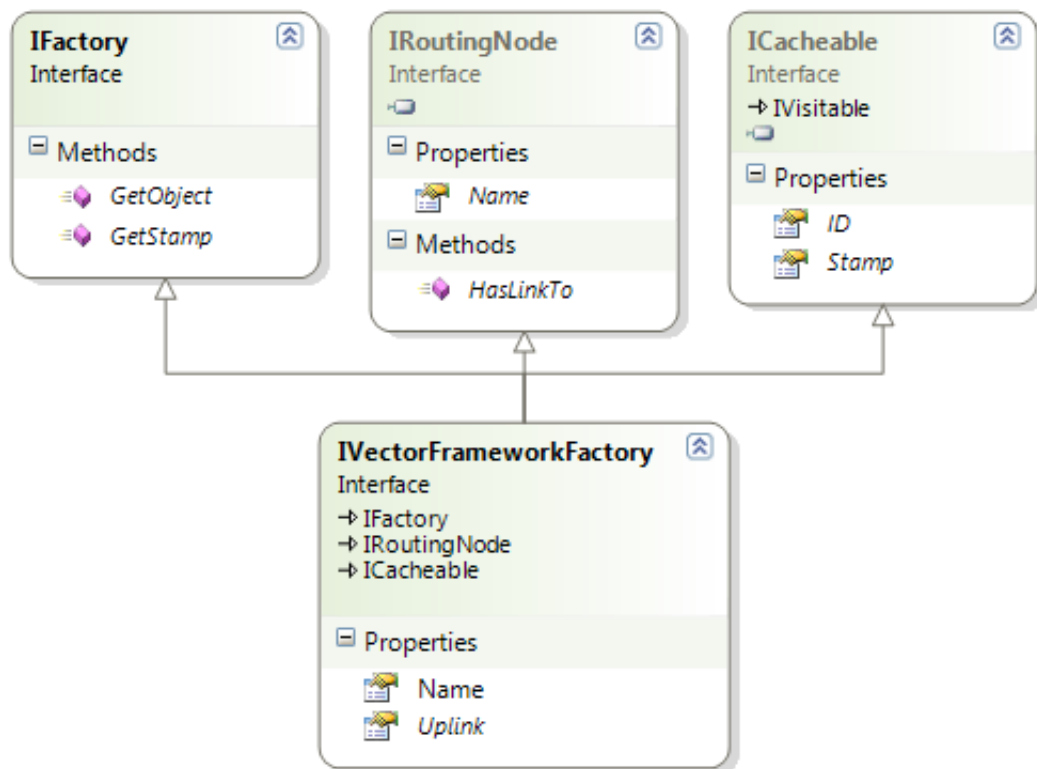


Figura 8: Diagrama de classes das interfaces relacionadas ao uso de *factories*

As *factories* são baseadas nos *design patterns* *Abstract Factory* e *Builder* (GAMMA et al., 1995). Elas são objetos que instanciam e configuram instâncias de determinadas classes, criando-as a partir de *requests*. Cada *request* deve possuir todas as informações necessárias para criação e configuração dos objetos a serem criados. O processo de criação de objetos utilizando *factories* é detalhado através dos diagramas de classes da figura 8 e do diagrama de sequência da figura 9.

Há a possibilidade de se utilizar um cache. Na primeira vez que a *factory* receber uma dada *request*, ela guarda seu identificador e o objeto construído no cache. Nas vezes seguintes, se for recebida uma *request* com um identificador já armazenado, o objeto será resgatado do cache, economizando no tempo de criação e na memória ocupada por uma nova instância.

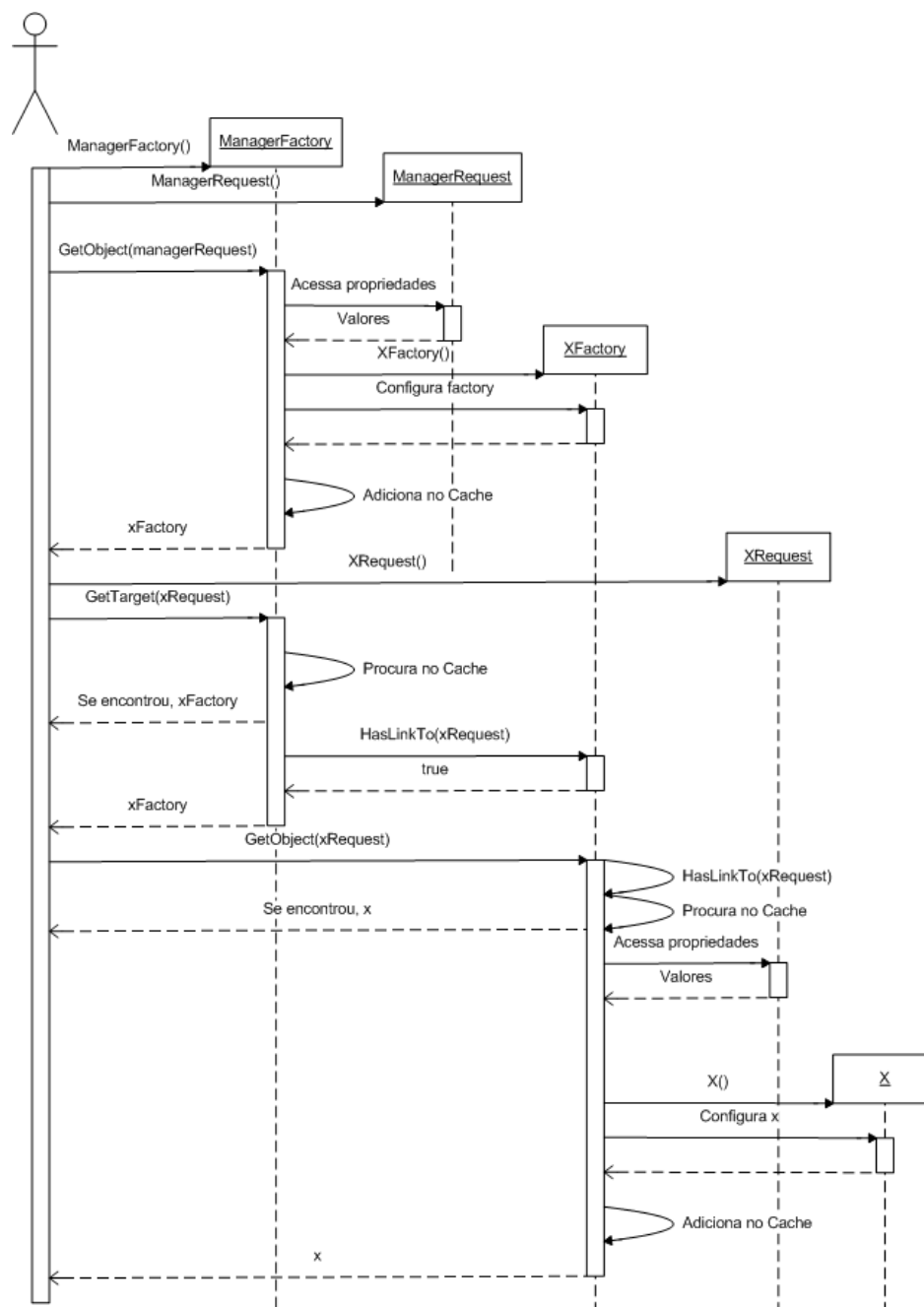
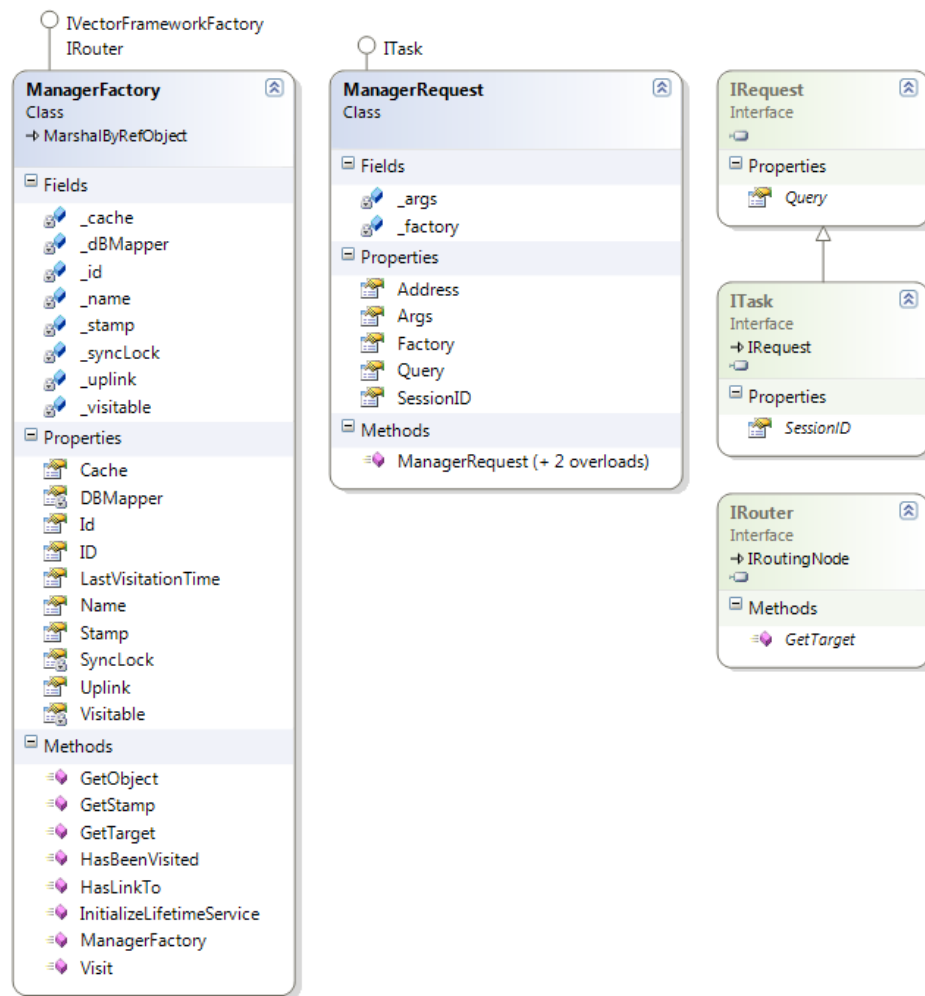


Figura 9: Diagrama de sequência da criação de objetos utilizando *factories*

Existe também uma *factory* que tem por função criar outras *factories*, sendo ela denominada *manager*. A figura 10 apresenta o diagrama de classes do *manager*. No início da execução, o *manager* recebe uma lista de *requests* que determina quais *factories* ele deve construir e armazenar. Quando um objeto qualquer deve ser instanciado, envia-se uma *request* para o *manager*, que se comporta como um roteador e procura qual *factory* é responsável por tratar a *request*, retornando-a. Envia-se então esta *request* para a *factory* retornada, e esta instancia e configura um novo objeto, ou procura no cache o objeto desejado.

Figura 10: Diagrama de classes do *manager*

Com *factories* pode-se mudar o comportamento da aplicação apenas trocando as *factories* do *manager*, essa mudança pode inclusive ser feita em tempo de execução.

Todas as camadas utilizam *factories*, observando-se assim o quão essenciais elas são para o funcionamento do sistema. Há diversas ocasiões nas quais um certo objeto precisa instanciar diversos outros objetos para que possa realizar suas funções, e sem a utilização de *factories*, a complexidade de gerenciamento dos objetos aumentaria drasticamente. Além disso, quando há atualização em tempo real, é importante que exista apenas uma instância de cada tipo para que não haja desperdício no consumo de processamento e memória.

Neste projeto, foram utilizados um *manager* e sessenta *factories* com funções diferentes.

5.5.1.2 Patterns

Os diversos *design patterns* utilizados foram organizados em um *package* com suas interfaces. Algumas delas estão ressaltadas a seguir junto de suas principais aplicações:

- *Publish-Subscribe*: Utilizado em objetos que se atualizam em tempo real.
- *Memento*: Utilizado para salvar e carregar estados de objetos.
- *Iterator*: Utilizado para percorrer sequência de objetos.
- *Command*: Utilizado para troca de mensagens no PAC.
- *Command Processor*: Utilizado para gerenciar *commands*.

5.5.1.3 PAC, Command e Command Processor

A *architectural pattern* PAC define uma estrutura para sistemas de software interativos na forma de hierarquia de agentes cooperativos. Cada agente é responsável por um aspecto específico da funcionalidade da aplicação e consiste de três componentes: *presentation*, *abstraction* e *control*. Esta subdivisão separa aspectos da interação homem-computador (*presentation*) das funções principais (*abstraction*) e da comunicação com outros agentes (*control*). Cada agente é independente dos outros agentes, podendo a comunicação entre agentes ser feita pelo componente *control* através de mensagens.

Para implementar as mensagens, optou-se pelo *design pattern Command* (GAMMA et al., 1995), que encapsula uma requisição como um objeto e fornece suporte para desfazer certas operações. O *command* desacopla o objeto que invoca a operação daquele que sabe como tratá-la.

A classe de controle é única para todos os agentes e foi implementada como um processador de *commands* descrito no *design pattern Command Processor*, que separa a requisição de um serviço (*command*) de sua execução (BUSCHMANN et al., 1996). Este componente gerencia requisições como objetos separados, agenda suas execuções, e provê serviços adicionais como armazenamento das *commands* em pilhas, permitindo que se desfça sua ação futuramente.

A solução que integra os *patterns* PAC, *Command* e *Command Processor* não resolve o problema do desacoplamento entre os agentes, uma vez que era desejável que nenhum

agente possuíisse uma referência direta com os outros. Utilizou-se para isso uma solução similar à encontrada nas *factories*: cada componente *control* implementa a interface *IRouter*, e as mensagens, como por exemplo a interface *IPACCommand*, implementam a interface *IRouteable*. Dessa forma, quando se deseja enviar uma *command* para outro agente, a camada de apresentação ou a camada de abstração enviam as *commands* para o componente *control* que, como um roteador, encaminha sua mensagem para os roteadores de outros agentes. Este roteamento ocorre até que a mensagem chegue nos agentes de destino. Com isso, nenhum agente sabe que outro existe por meio de referências diretas. Utilizando essa solução, é possível inclusive enviar *commands* em *broadcast*.

Cada agente possui um nome único e uma lista de *tags* para sua identificação. Quando se envia um *command*, é necessário que se especifique o nome ou *tags* do agente que se deseja atingir. A possibilidade de combinações está descrita na tabela 4.

Tabela 4: Regras para nomes e *tags* em *commands*

Nome	Tags	Alvos
*		Todos os agentes (<i>broadcast</i>)
*	Tag1, Tag2 ... TagN	Agentes que possuam as <i>tags</i> Tag1 ou Tag2 ... ou TagN
Nome1		Agentes que possuam o nome Nome1
Nome1	Tag1, Tag2 ... TagN	Agentes que possuam o nome Nome1 (nesse caso as <i>tags</i> são desprezadas)

5.5.1.4 Acesso a Dados

O *package* de acesso a dados é um componente de execução de *queries* no banco de dados. Trata-se de um conjunto de classes que facilitam o trabalho implementando uma arquitetura baseada em *factories*.

A *factory* que executa *queries* no banco de dados se chama *SqlExecutorFactory*. Esta é responsável por tratar *requests* do tipo *SqlExecutorRequest*, que possuem as seguintes propriedades: prioridade, *FireAndForget* e *DoCache*. Além disso, como uma *SqlExecutorRequest* precisa de um objeto que implemente a interface *ISql*, possui também a propriedade *Query* (*string* que contém o comando SQL) e a propriedade *SqlType* (que pode ser *Read* ou *Write*).

Quando a opção *FireAndForget* é escolhida (somente aplicável para *ISql* com valor *Write* para *SqlType*), o *ISql* é enviado para um servidor de execução de *queries* por meio

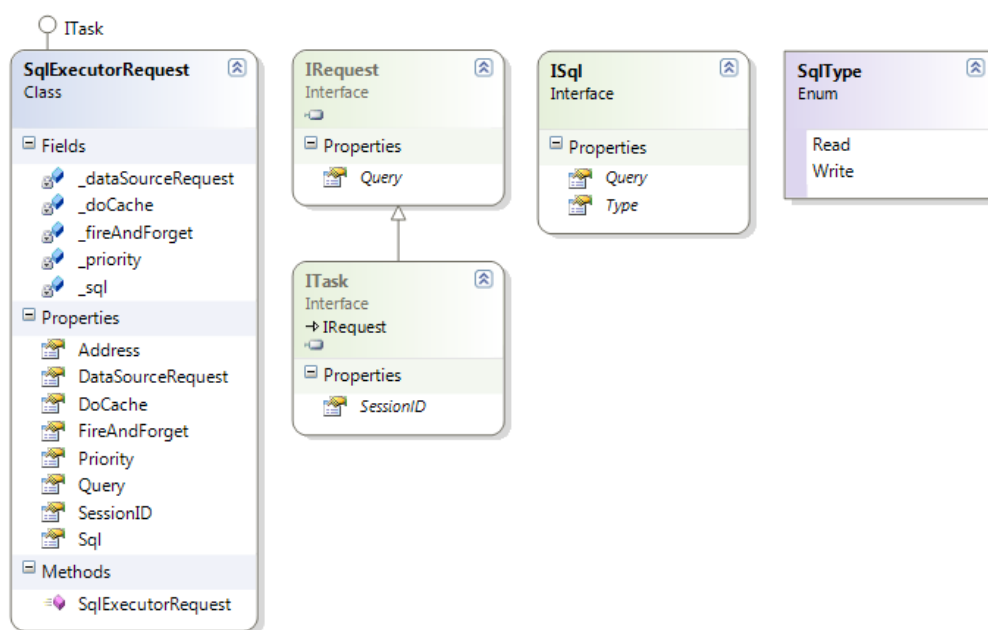


Figura 11: Diagrama de classes relacionado com o acesso a dados

de filas (ver *message queuing* na seção 6.4.1.1). Dessa forma, pode-se, por exemplo, enviar execuções mesmo com o banco de dados fora do ar. Nesse caso, cabe ao outro sistema esperar até que o banco de dados seja reestabelecido. Quando se utiliza a opção `DoCache` (somente aplicável para `ISql` do com valor `Read` para `SqlType`), os dados que são recebidos do banco de dados são armazenados na memória, dessa forma, se um pedido exatamente igual for feito novamente, o resultado é buscado na memória ao invés de ser buscado no banco de dados novamente.

Ao pedir uma `SqlExecutorRequest` ao *manager* pelo método `GetTarget()`, obtém-se normalmente uma `SqlExecutorFactory`, e ao se enviar uma `SqlExecutorRequest` ao `SqlExecutorFactory` pelo `GetObject()` a *query* é executada e obtém-se um `IRawData` caso seja uma operação de leitura.

A parte de acesso a dados é padronizada e pode ser reutilizada em outros projetos, estando por isso dentro de *Framework*, e não em uma camada separada.

Como toda escrita e leitura no banco de dados são feitas por meio de *factories*, toda a comunicação e complexidade ficam transparentes para o programador que usa o componente de acesso a dados.

5.5.1.5 Segurança de Acesso

Devido à demanda de controle de acesso de conteúdo no SPCRCA, verificou-se a necessidade de um método de autenticação de usuário. A motivação principal é devida ao fato de que diferentes tipos de usuários poderão executar ações distintas e um acesso indevido pode gerar prejuízos financeiros para a empresa cliente. Desta forma, deve-se controlar os recursos que cada entidade ativa pode acessar, e o método de acesso a um dado recurso. Para tal, o *design pattern* intitulado *Authorization* (SCHUMACHER et al., 2005) foi utilizada como modelo inicial para acomodar com flexibilidade diferentes usuários, recursos e permissões. Sua principal vantagem é facilitar mudanças de responsabilidades determinando permissões a usuários de acesso a determinados recursos.

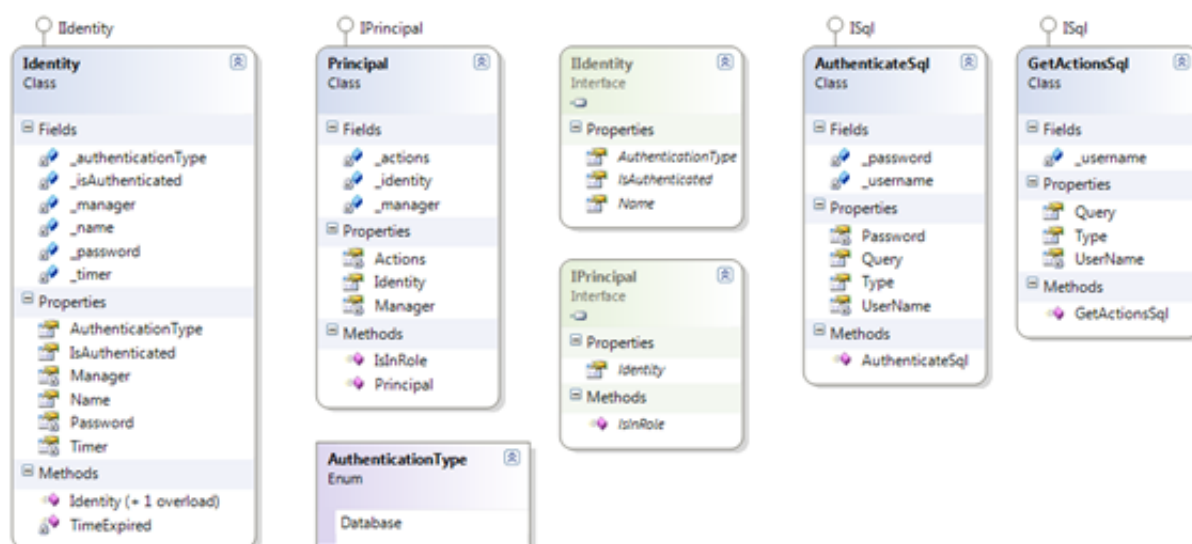


Figura 12: Diagrama de classes para a segurança de acesso

A *Authorization* sozinha possui como desvantagem a dificuldade em fazer a manutenção de permissões, visto que havendo uma grande quantidade de usuários, a modificação de permissões um a um torna-se muito trabalhosa. Por isso foi adicionada a *design pattern Role-Based Access Control* (SCHUMACHER et al., 2005) à solução. Este padrão descreve como distribuir permissões baseando-se na função ou tarefa do usuário no ambiente em que o controle de acesso a recursos é necessário, e em sistemas em que há um grande número de usuários e uma grande variedade de recursos. Os usuários são agrupados em papéis levando em conta suas funções.

O acesso a informações não deve ser verificado apenas na entrada do sistema: um indivíduo mal intencionado pode utilizar o sistema na ausência do usuário no controle do computador, ou o sistema pode permitir acesso a vários níveis, devido à falta de alguma

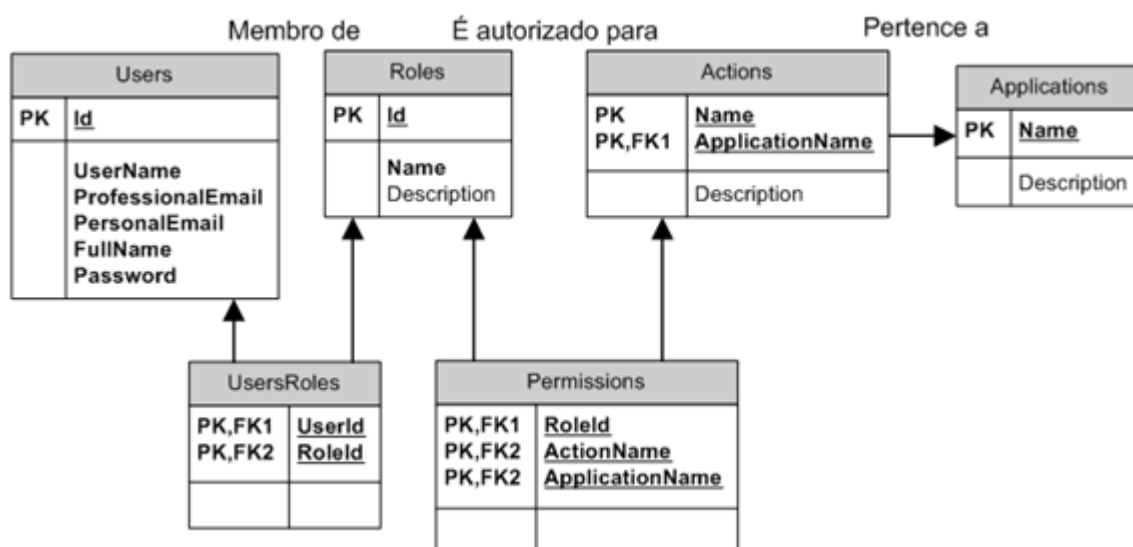


Figura 13: Modelo do banco de dados para a segurança de acesso

verificação na hora da codificação, permitindo brechas na segurança. Para resolver esses problemas foi adicionado à arquitetura o *design pattern Reference Monitor* (SCHUMACHER et al., 2005), que descreve como definir um processo abstrato que intercepta todas as requisições de recursos e verifica a permissão de autorizações em qualquer nível da aplicação. Este mecanismo, no entanto, pode causar perda de desempenho. Para evitar uma degradação de desempenho muito grande, as informações de autorizações são guardadas na memória desde a primeira verificação até a seção expirar, ao invés de serem verificadas a cada vez no banco de dados.

A figura 13 indica o modelo de dados referente à autenticação. As informações de usuários, funções e ações são configuradas no banco de dados, os usuários (*Users*) fazem parte de tarefas ou funções (*Roles*) numa relação n para n; cada função está associada com ações (*Actions*) numa relação n para n; e cada ação está associada com uma aplicação (*Applications*) numa relação 1 para 1. As senhas dos usuários são salvas na base de dados como seus *hashs*.

A primeira autenticação no sistema é feita no início. Há um temporizador configurado com um tempo de expiração de 10 minutos para que ocorra o término da seção. Ao fim deste período, quando o usuário tenta executar novamente uma ação que requer permissão, a tela de autenticação é chamada novamente ao invés de executar a ação. Cada ação do usuário será verificada para ver se ele pode executar a ação.

A figura 12 mostra o diagrama de classes com interfaces, enumerações e classes utilizadas para controlar o acesso dos usuários aos recursos do sistema.

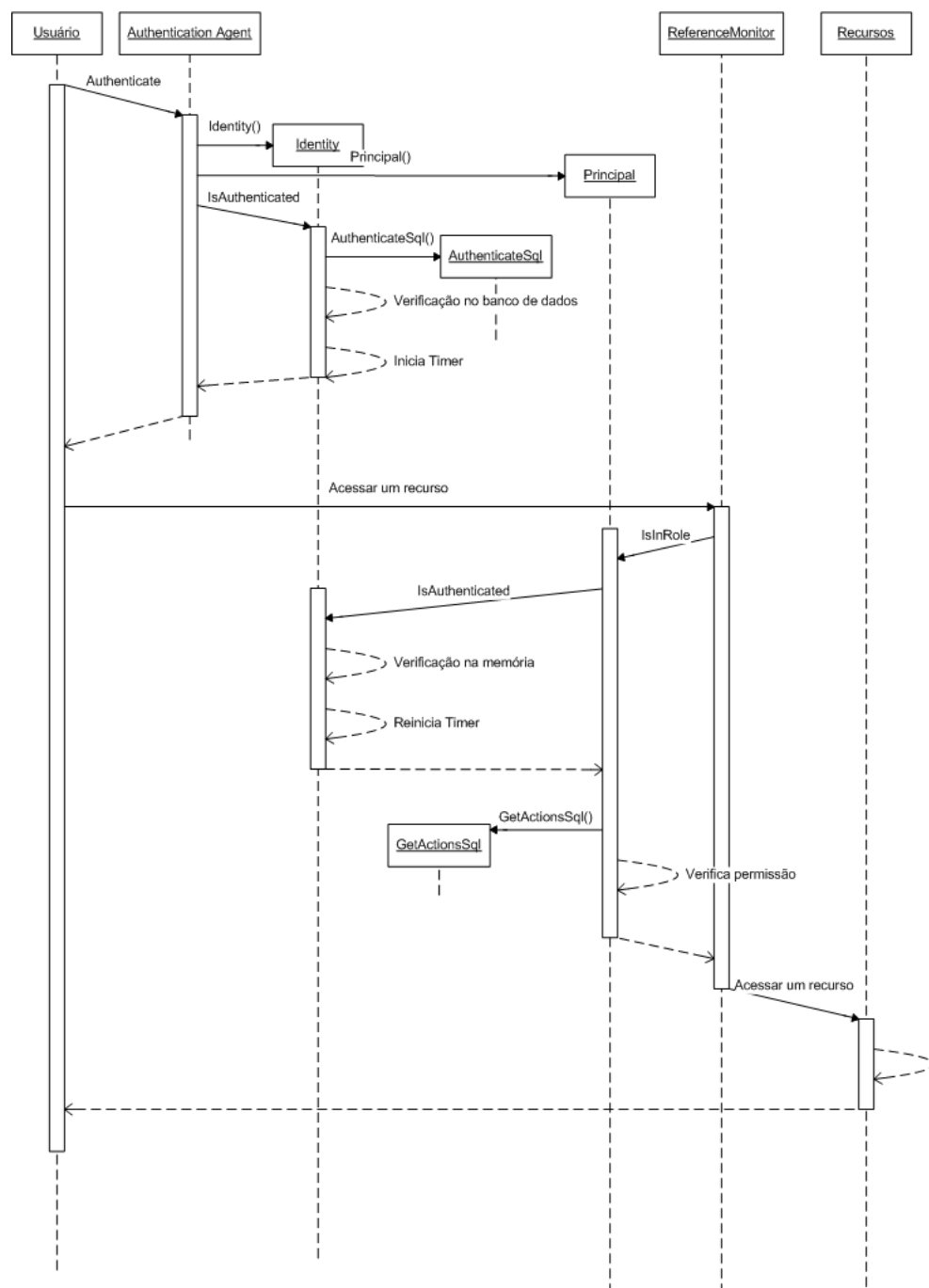


Figura 14: Diagrama de sequência da autenticação e do acesso a um recurso

A figura 14 mostra o diagrama de sequência desde a operação de autenticação do usuário no sistema até o acesso a um recurso.

5.5.2 Business

O bloco *Business* da arquitetura possui *packages* que modelam a camada de negócios do sistema. Esta seção apresenta os principais *packages* do bloco *Business* representado

na arquitetura da figura 6.

5.5.2.1 Securities e Trades

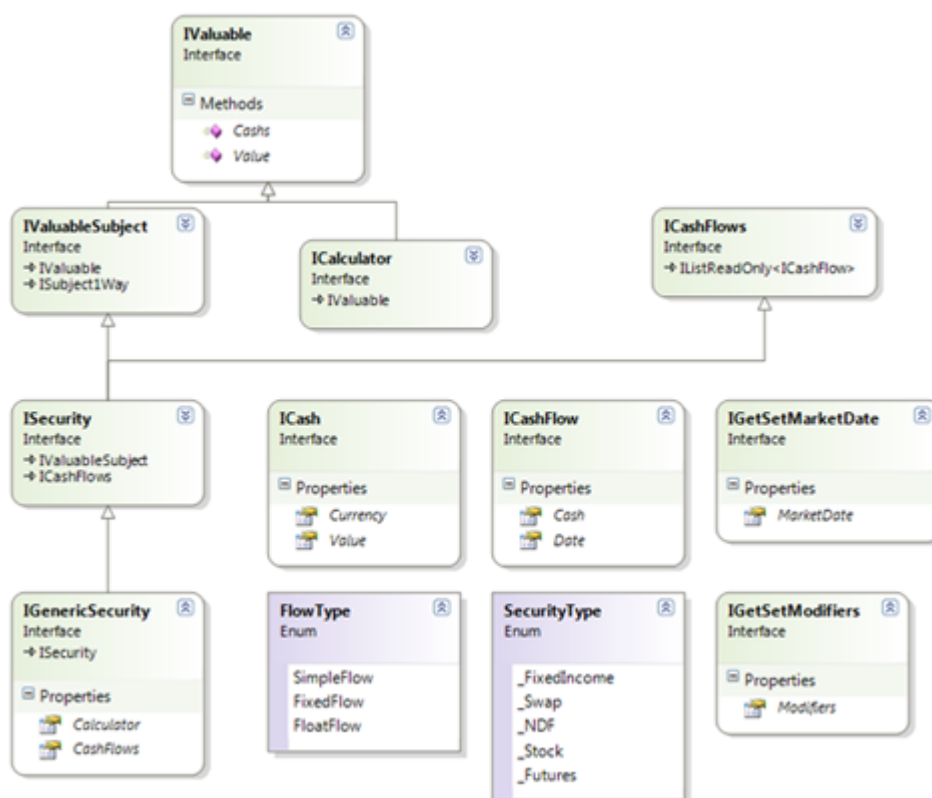


Figura 15: Diagrama de classes das interfaces e enumerações necessárias à implementação de *securities*

O primeiro passo na modelagem foi a definição das interfaces das *securities* e *trades*. Esta abordagem foi feita pois existem diferentes tipos de *securities* como *Stock*, *Bonds* e *NDFs* e queria-se que o sistema funcionasse bem para todos os tipos, inclusive os que fossem adicionados futuramente.

A figura 15 mostra esta interface. Toda *security* possui fluxos de caixa e possuem um preço, por isso que ela implementa a interface *ICashFlows* e *IValuable*. A interface *ICalculator* é necessária pois existem *securities* como *Options* que podem ter seu valor calculado por diferentes fórmulas, sendo uma delas a fórmula de Black-Scholes (HULL, 2006), explicada no apêndice D.

Quanto a *trades*, existe apenas uma classe. O *trade* é uma operação realizada sobre uma *security*, e sua implementação foi feita para ser independente de qual *security* for, aproveitando as definições de interface descritas anteriormente.

Todas as interfaces foram elaboradas para atender as funcionalidades de todos os módulos do SPCRCA.

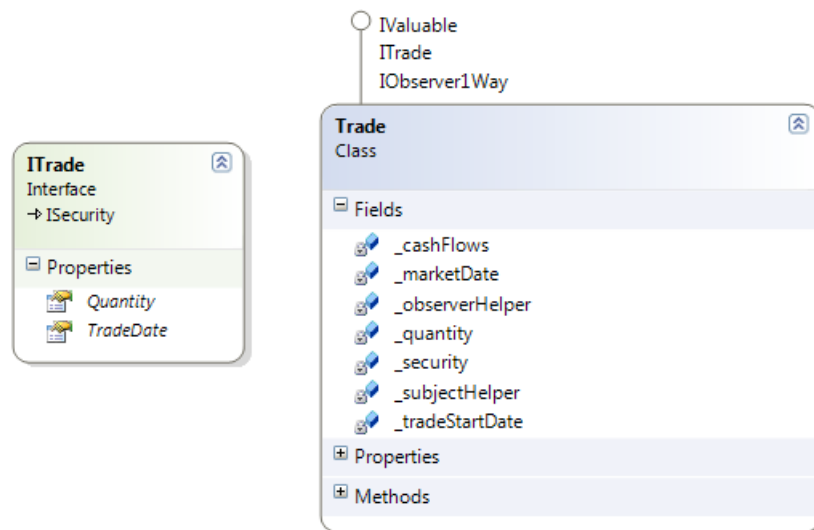


Figura 16: Diagrama de classes de *trades*

Todos os objetos são instanciados com a ajuda de *factories*, e para isso foi necessário criar no mínimo uma *request* e uma *factory* para cada tipo de *security*.

A figura 17 mostra um exemplo de todas as classes utilizadas para a implementação de *Stocks*.

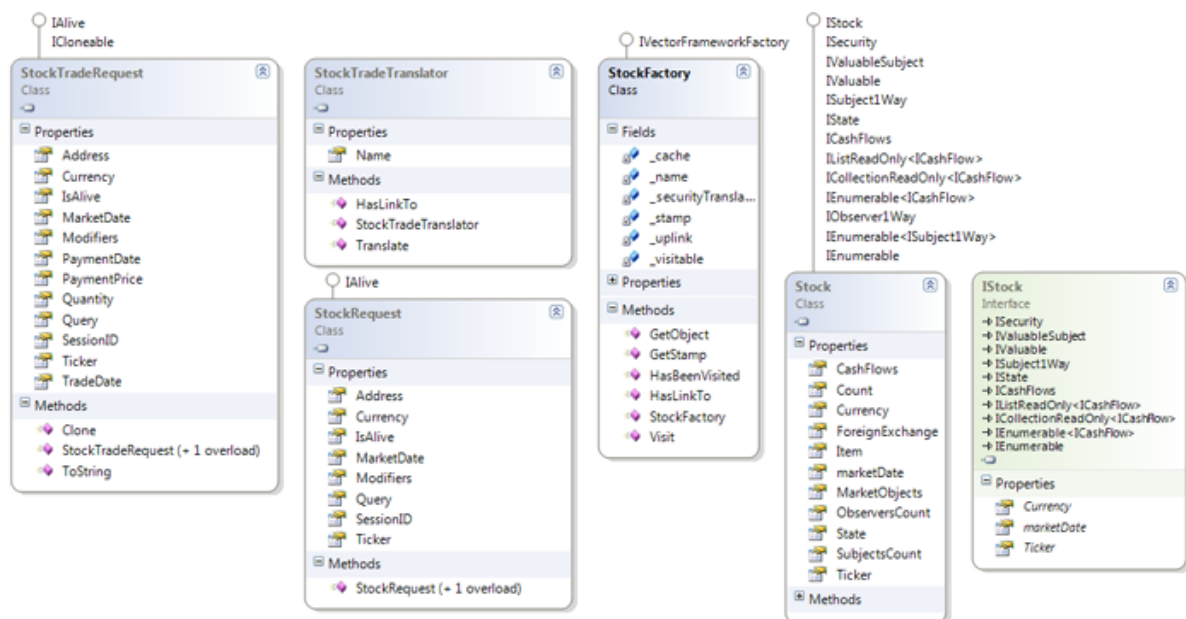


Figura 17: Diagrama de classes envolvendo criação de *securities* e *trades* de *stocks*

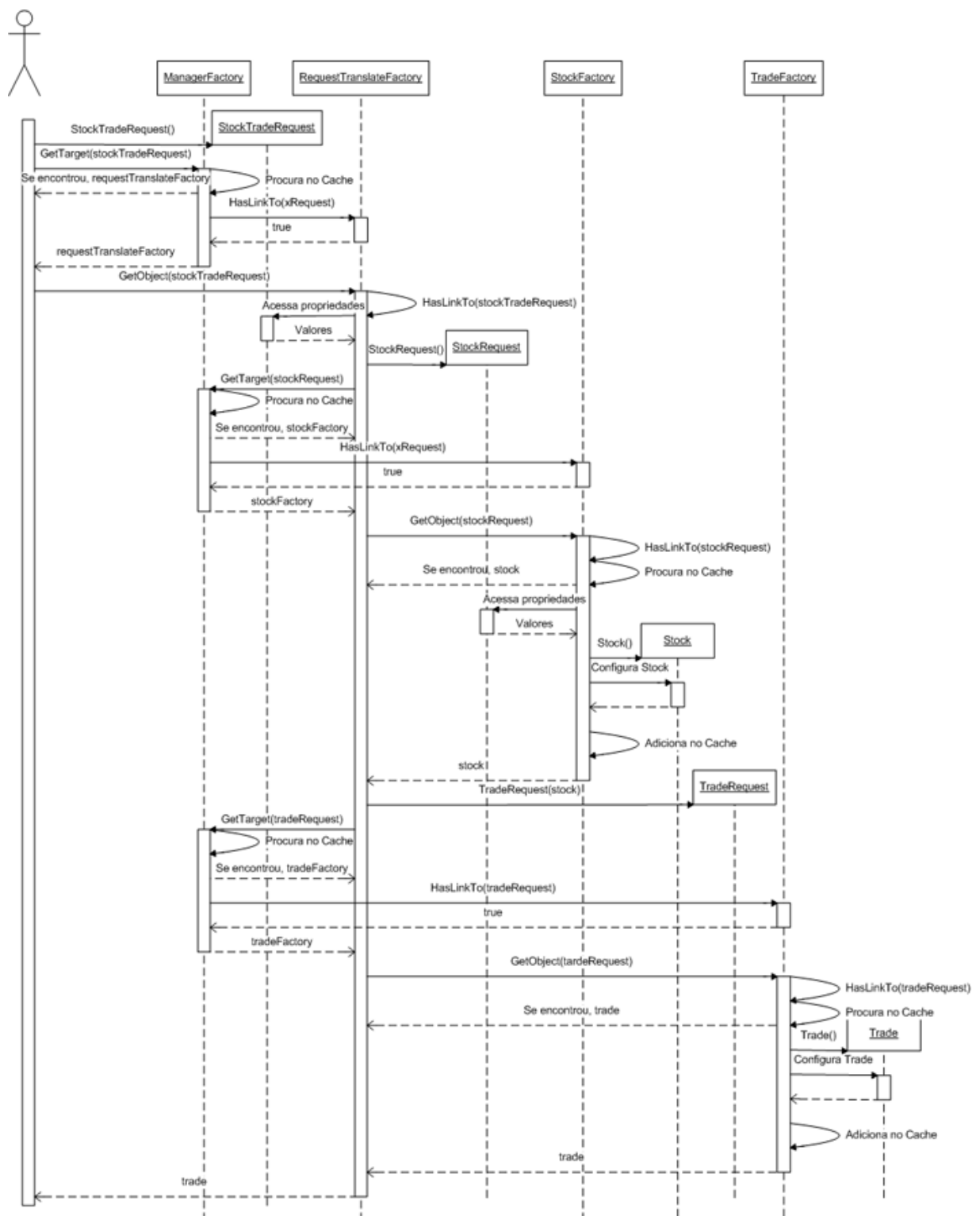


Figura 18: Diagrama de sequência envolvendo criação de *securities* e *trades* de *stocks*

5.5.2.2 Risk

O risco de um *trade* é determinado pelo risco causado pela variação de suas variáveis de mercado que podem ser agrupadas em objetos de mercado. Cada *security* possui um

conjunto de variáveis de mercado. Não necessariamente um tipo de *security* terá sempre o mesmo conjunto de variáveis de mercado, pois isso depende da configuração que foi atribuída à *security*.

O risco de primeira ordem é a primeira derivada aplicada ao cálculo do valor do *trade* sobre a variável. Esse cálculo é feito graças a variação do valor da *security* com objetos modificadores. A figura 19 mostra os passos representados por um diagrama de sequência para se calcular o risco de um *trade*.

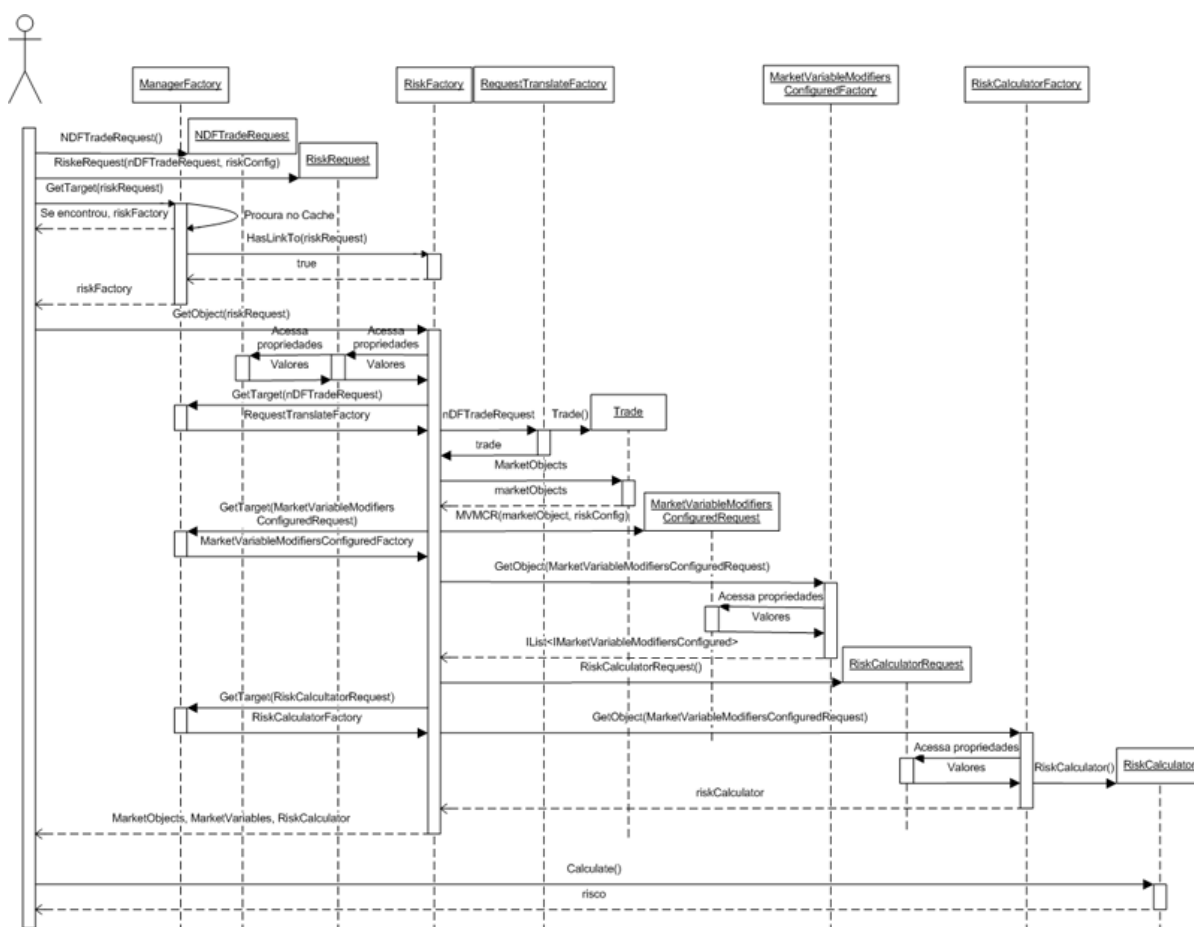


Figura 19: Diagrama de sequência do cálculo do risco

5.5.2.3 Servidor Tempo Real

O sistema Real Time Provider tem como função primordial obter dados do mercado em tempo real. Para isso, é necessário requisitar ao *manager* as *factories* DataSourceFactory, QueueFactory e QueueManagerFactory, que futuramente criarão objetos de gerenciamento das fontes de dados, filas e gerenciadores de filas.

A SubscribeRequest é uma IRequest necessária para fazer requisições de objetos que

implementam a interface *IPublisher*. Ao enviar um *SubscribeRequest* ao *Manager* pelo método *GetTarget()* deste, obtém-se normalmente a *factory* *RealTimeMessagingFactory*, e ao enviar o *SubscribeRequest* ao *RealTimeMessagingFactory* pelo método *GetObject()* deste, obtém-se um *publisher*. O *publisher* é responsável por receber os *RealTimeData* das filas, e implementa a interface *ISubject1Way* que é uma implementação do *design pattern* Publish-Subscriber (GAMMA et al., 1995).

Para utilizar objetos *publisher* deve-se criar classes que implementem *IObserver1Way*, e conectar ao *publisher*. Assim, toda vez que houver uma atualização num dado pedido, o *publisher* avisará todos os *IObserver1Way* que estão em sua lista, e deve notificá-los com o novo dado. Dessa forma é possível obter dados em tempo real.

É importante notar que no caso de se utilizar objetos que se atualizam em tempo real, deve-se montar uma cadeia de *ISubject1Way* e *IObserver1Way* para que a atualização ocorra em todos os objetos e de forma ordenada.

Para evitar processamento em objetos que não estão mais sendo utilizados, deve-se chamar o método *DetachFromAllSubjects()* de cada *IObserver1Way* que não for mais utilizado. Caso o *publisher* não tenha mais nenhum *IObserver1Way* em sua lista, ele automaticamente se desconecta da fila e avisa o roteador de *RealTimeData* para parar de enviar para ele. Caso outro *IObserver1Way* se conectar, esse *publisher* se conecta ao roteador novamente.

5.5.3 Application

O sistema SPCRCA pode ter várias telas abertas ao mesmo tempo, devendo estas telas trocar mensagens entre si. Por exemplo, na tela *Book View*, quando um *trade* é selecionado, é possível enviá-lo para uma tela de *Historic Pricing*. Assim, o bloco *Application*, que engloba todas as telas do sistema, deve implementar os *patterns* PAC e *commands*, explicitados anteriormente, para que os módulos sejam o mais desacoplados possível.

Cada um dos módulos acessa pelo componente *abstraction* o bloco *Business*, através de *factories* e *requests* pertencentes a esse bloco. Observa-se que o componente *presentation* não pode interagir com o *Business*, com a exceção de se ele receber um objeto do *Business* através de *commands*.

O componente *control* possui uma implementação única para todos os módulos, podendo sempre ser reutilizada. Assim, esta utiliza apenas classes e interfaces do bloco *Framework*.

Essa configuração entre os blocos é representada pela figura 6.

5.5.4 Considerações sobre Requisitos Não-Funcionais

- Qualidade do Processo: As falhas devem ser descritas em um *log*;
- Configurabilidade da Interface e Usabilidade: Estes dois requisitos são previstos pelo projeto de IHC, e são garantidos através de iterações com o cliente e com os usuários do sistema;
- Desempenho e Escalabilidade: Para estes dois requisitos, as seguintes abordagens são tomadas relacionadas à melhor utilização dos recursos de hardware e rede:
 - *Lazy instantiation*: Atrasa a inicialização de um objeto até o momento que ele é de fato utilizado, diminuindo a carga na memória no período inutilizado;
 - Cache: Reutiliza objetos já instanciados;
 - *Command local*: Roteamento da *command* no PAC pode ser definida como local quando o destino faz parte do mesmo agente, não passando pelo roteamento da malha de agentes;
 - *SQLExecutor*: A propriedade *FireAndForget* envia a *request* para uma fila (*message queuing*) para processamento em um servidor, desacoplando o cliente.
- Disponibilidade: Para o requisito de disponibilidade, utiliza-se, no escopo da modelagem, abordagens relacionadas à reconexão com os servidores;
- Segurança de Acesso: Os aspectos de modelagem que visam a garantir segurança de acesso estão descritos na seção 5.5.1.5;
- Testabilidade: O requisito de testabilidade é obtido através da utilização de ferramentas semi-automáticas de teste do sistema.

5.6 Modelo de Dados

O modelo de dados consta no apêndice C.

5.7 Critérios de Aceitação

5.7.1 Mapeamento dos Requisitos

Deve-se verificar que os requisitos levantados no capítulo 4 foram mapeados no sistema. As funções de negócio representam análises financeiras e têm, portanto, módulos correspondentes no sistema. As análises e os módulos no sistema compartilham o mesmo nome. Deve-se então verificar se as funções previstas estão presentes no produto final. As funções de infra-estrutura, no entanto, estão distribuídas no sistema. Deve-se verificar então quais componentes mapeiam a funcionalidade, tarefa esta simples devido à organização lógica existente.

5.7.2 Testes

Os testes foram realizados pela equipe de desenvolvimento, de modo a verificar a consistência do sistema com os requisitos levantados. Foram realizados testes unitários, modulares e de integração e os comentários se encontram no capítulo 7.

5.7.3 Aceitação

A aceitação foi feita por partes. A cada iteração do desenvolvimento, o produto construído na etapa foi avaliado pelo cliente. Este foi convidado a testar todas as funcionalidades já implementadas, simulando diversas situações relevantes para as suas necessidades profissionais. Essa abordagem permitiu a análise do funcionamento de partes do software dentro do escopo geral e diminuiu o risco relacionado ao não cumprimento de algum requisito, erros nos dados, não satisfação com o produto sendo gerado, entre outros.

A aprovação formal do sistema foi iniciada ao final da construção, com a homologação seguindo critérios observados em um Plano de Testes aprovado anteriormente. Esta homologação deve englobar a verificação das funcionalidades desejadas e dos dados calculados.

5.8 Considerações Finais

Devido ao foco na arquitetura, observou-se a importância da fase de Elaboração dentro do processo de desenvolvimento do SPCRCA. Uma preocupação inerente é que não

se apresentou a maturidade desejada em relação ao o produto final no fim da fase de Concepção. Com isso, vê-se a importância de se utilizar um processo iterativo e incremental, de forma que o conhecimento agregado nas fases seguintes fosse repassado para os requisitos.

Com os modelos apresentados neste capítulo, esperou-se a simplificação das fases de Construção e Transição, uma vez que a complexidade do sistema ficou mais facilmente gerenciável devido à modularização e à padronização dos componentes, justificando assim a maior quantidade de recursos gastos nesta fase.

6 *Construção*

“Um gênio é 1% de inspiração e 99% de transpiração.”

Thomas Alva Edison

6.1 Introdução

Neste capítulo são apresentados o protótipo mínimo construído para validação da arquitetura proposta utilizando *factories* e PAC, e considerações acerca da implementação do SPCRCA. O resultado dos esforços nas atividades descritas é uma versão do sistema a ser testada.

Após a construção, os seguintes módulos do SPCRCA estavam construídos e integrados:

- *Main*
- *Book View*
- *Trade Builder*
- *Security Builder*
- *Cash Flows*
- *Historic Pricing*
- *Market Scenario*
- *Market Pricing and Risk*
- *Drawing*
- Simulador de Servidor Tempo Real

6.2 Protótipo Mínimo

O protótipo mínimo teve sua construção iniciada na fase de Elaboração, como forma de validar a arquitetura, e encerrada no início da fase de Construção do Processo Unificado, de modo que uma funcionalidade do sistema estivesse construída verticalmente desde a interface com o usuário até o acesso a dados (ver figura 6).

A funcionalidade escolhida para o protótipo foi o controle de acesso aos dados. Embora este não faça parte do núcleo do sistema sob a visão de negócios, faz parte do núcleo de segurança de acesso e ilustra muito bem o funcionamento das *factories*, de *commands* e o acesso ao banco de dados, sem que seja necessária a implementação da lógica de

negócios neste momento. A construção em si iniciou-se expandindo o protótipo de interface (*presentation*), adicionando-se principalmente lógica ao *control* para que a comunicação entre os agentes já pudesse ocorrer.

Com este protótipo, verificou-se que a familiarização com esta arquitetura seria mais difícil inicialmente, mas, no decorrer da implementação, o desacoplamento e a consistência facilitariam a evolução do trabalho.

6.3 Visão Geral da Construção

A construção respeitou, em geral, a arquitetura e os modelos apresentados no capítulo 5. Detalhamentos da construção estão apresentados na seção a seguir.

6.4 Detalhamento da Construção

6.4.1 Framework

6.4.1.1 Arquitetura Message Queuing

Message Queuing faz parte do sistema operacional Windows. Suas principais vantagens são:

- Mensagens podem ser enviadas em ambientes sem conexão, não sendo necessário que a aplicação que envia e recebe mensagens esteja rodando ao mesmo tempo. Isso é uma grande vantagem quando se trabalha com objetos que se atualizam em tempo: quando um cliente para de responder por um tempo, pode talvez estar sobrecarregado. Ele pode tratar as informações depois quando ele retornar;
- Pode-se ter mensagens salvas na memória ou no disco rígido. No primeiro caso, a velocidade é maior, e no segundo, pode-se reiniciar a máquina e as mensagens ainda estarão lá para serem enviadas;
- Por um mecanismo de recuperação, pode se ter garantia de entrega;
- Pode-se definir controle de acesso às filas e criptografar mensagens;
- Pode-se definir prioridades às mensagens.

Com *Message Queuing*, mensagens são escritas e lidas a partir de uma fila. Todo o manuseio com mensagens e filas foi implementado utilizando *factories* e objetos que implementam os *design patterns* Proxy e Publish-Subscribe (GAMMA et al., 1995).

6.4.1.2 Drawing

Os gráficos do sistema são feitos a partir de um *Adapter* (GAMMA et al., 1995) sobre um componente de uma biblioteca gráfica *open source* chamada ZedGraph (ZEDGRAPH.ORG, 2008). O *Adapter* foi necessário para adequação quanto à arquitetura do sistema.

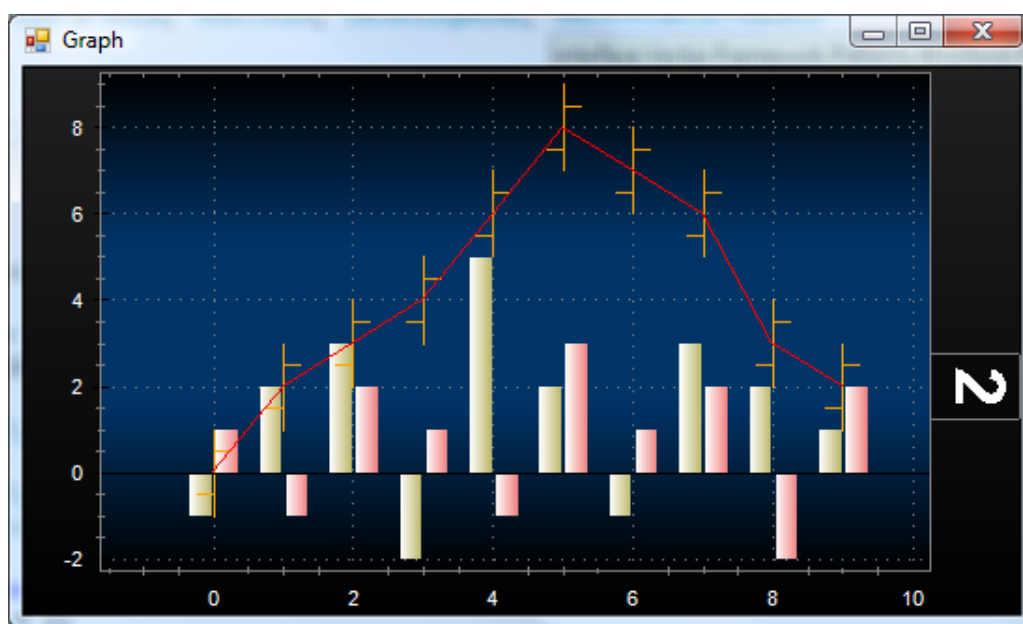


Figura 20: Exemplo de gráfico utilizando o componente *Drawing*

O *package Drawing* contém um agente gráfico. Ele faz parte do bloco *Framework* porque é mais utilizado como ferramenta para aplicações do que é uma aplicação propriamente dita. Implementa o *architectural pattern* PAC (BUSCHMANN et al., 1996), e deve ser manuseado por meio de *commands* para controle de suas funções.

O agente também implementa o *design pattern* Publish-Subscribe (GAMMA et al., 1995), é portanto capaz de mostrar um gráfico se atualizando em tempo real.

Pode desenhar vários tipos de gráficos como linhas, barras, contornos, *Open High Low Close* (OHLC), *candle sticks* e pizzas. No SPCRCA foram utilizados apenas gráficos de linha.

O agente gráfico foi feito para ser genérico de maneira a dar suporte a várias outras aplicações além do SPCRCA.

A figura 20 ilustra um exemplo de interface misturando vários tipos de gráficos numa única janela.

6.4.2 Business

6.4.2.1 Simulação de Servidor Tempo Real

Como neste projeto, a equipe de desenvolvimento não esteve necessariamente alocada no espaço físico provido pela Vector Investimentos, os dados em tempo real não estiveram sempre disponíveis uma vez que a fonte envia os dados apenas para o IP da empresa. Portanto, viu-se a necessidade de se simular o módulo de dados do mercado em tempo real.

Um simulador que similar ao servidor em tempo real original foi então construído. Para garantir o mínimo de trabalho ajustando-se entre as duas fontes de dados distintas, utilizou-se do desacoplamento proposto pelo sistema. A modificação ocorrida na aplicação consistia de se enviar uma *request* para o *manager*. Esta *request*, ao invés de pedir uma instância da classe *RealTimeMessagingFactory*, pedia uma *RealTimeMessagingStubFactory*. Tendo recebido este objeto, o restante do processo de tratamento de dados em tempo real é transparente para a aplicação, uma vez que tanto o servidor original quanto a simulação compartilham da mesma interface.

Do mesmo modo que o servidor em tempo real original, o simulador também publica os dados gerados. A diferença reside no fato que o original pede os dados para um servidor por *message queuing* e configura um *publisher* (GAMMA et al., 1995) para ler de uma fila os dados recebidos do servidor e repassar a informação para os objetos que pediram, enquanto o simulador deve gerar dados com um comportamento semelhante ao do mercado, não utilizando *message queuing*. Conforme Wilmott (1998), é possível utilizar a fórmula 6.1 para tal intuito.

$$S_i = S_{i-1} \cdot (1 + \mu \cdot \Delta t + \sigma \cdot \Delta X \cdot \sqrt{\Delta t}) \quad (6.1)$$

Nesta fórmula, seja S_i o preço de uma ação num determinado momento, então este valor é calculado como função de seu último valor S_{i-1} , do retorno esperado (*drift*) μ , do período de tempo entre as atualizações Δt , da volatilidade σ e de um valor aleatório retirado de uma distribuição normal ΔX . Observa-se que S_{i-1} é facilmente obtido como o valor do último fechamento presente no banco de dados. Além disso, a efeito de simplifi-

cação, consideram-se como valores condizentes com o comportamento real do mercado um *drift* de 0,1 e uma volatilidade de 0,4.

O valor aleatório proveniente da distribuição normal, por sua vez, pode ser calculado pela fórmula 6.2 (WILMOTT, 1998).

$$\Delta X = -6 + \sum_{i=1}^{12} x_i \quad (6.2)$$

Cada x_i representa um valor aleatório diferente entre 0 e 1. Com isso, apenas uma questão ainda não está resolvida: o período de tempo entre as atualizações não é constante. Portanto, o comportamento simulado mais próximo do real é obtido quando a publicação dos valores calculados ocorrer com uma certa probabilidade para cada período de tempo. Por exemplo, utilizou-se probabilidade de 10% de se publicar o valor a cada meio segundo.

6.4.3 Application

6.4.3.1 Módulos Funcionais

Cada módulo funcional do sistema SPCRCA é um agente da arquitetura PAC, o que significa que ele possui um componente *abstraction*, um componente *presentation* e um componente *control*. O módulo *control* é apenas um roteador e processador de *commands* e por isso a classe utilizada é a mesma em todos os módulos. Com a implementação em agentes, cada módulo é independente dos outros, eles podem ser reutilizados em outros sistemas.

A janela do componente *presentation* foi feita com base no protótipo de telas mostrado no apêndice B. Algumas melhorias de interface foram feitas, algumas telas foram unidas em uma só e outras separadas em duas.

A construção dos módulos funcionais foi muito fácil graças à utilização de *factories* para criação e configuração de objetos de elevada complexidade.

Quando o módulo funcional trabalha com objetos que se atualizam em tempo real, a lógica de checar as mudanças fica para o componente *abstraction* dos agentes. A cada mudança, ele instancia uma *command* que é enviada ao *control* que determina que o seu *abstraction* deve a executar. Então o componente *abstraction* atualiza a janela com os novos dados.

Os módulos construídos são apresentados na seção 7.3.

6.4.3.2 Integração entre Módulos

A abertura de novos módulos e a comunicação entre módulos foi feita utilizando *commands* que são roteadas entre os componentes *control* de cada agente da arquitetura PAC.

Alguns objetos podem ser transferidos pelo método *drag and drop* de um módulo para outro, a implementação dessa funcionalidade também utiliza *commands*.

Múltiplas Áreas de Trabalho

Um dos requisitos funcionais do sistema é a possibilidade de agrupamento de janelas em áreas de trabalho. O usuário pode escolher qual área de trabalho cada janela pertence e então escolher na janela principal qual área de trabalho se deseja ativar. O número de áreas de trabalho que se tem disponível é determinado pelo usuário na janela principal.

A implementação dessa funcionalidade foi muito simples graças à arquitetura, já que cada janela pertence a um agente do PAC. Cada componente *abstraction* de cada agente foi implementado para tratar as *commands* *SetWorkAreaNumberCommand* e *SetWorkAreaCommand* que são enviadas a partir do *abstraction* do agente principal, que tem a tela principal como interface ao usuário. A *command* *SetWorkAreaCommand* avisa todos os outros agentes qual área de trabalho está selecionada. Se o agente pertence à mesma área de trabalho, sua janela fica visível, caso contrário, fica invisível. A *command* *SetWorkAreaNumberCommand* determina qual é o número de áreas de trabalho que o usuário pode escolher e é enviado a todos os agentes.

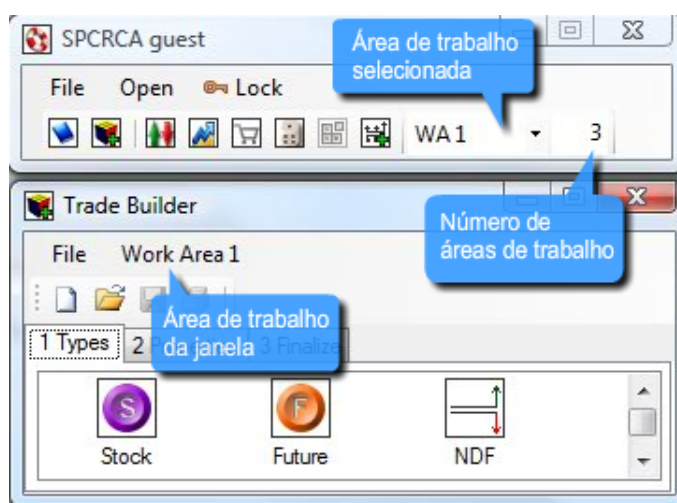


Figura 21: Implementação de áreas de trabalho

Drag and Drop

A utilização de *drag and drop* foi motivada pela facilidade que seu uso apresentaria para o usuário final do sistema. Desta forma, a comunicação entre diversos módulos poderia ser feita utilizando este recurso. Por exemplo, um *trade* criado no *Trade Builder* seria arrastado para a tela de *Book View*, adicionando-o assim na carteira. Em seguida, um grupo de ativos da carteira poderia ser arrastado para o *Market Pricing and Risk*, de forma a obter uma análise de risco para este grupo.

Para se atingir este resultado, os objetos arrastados a partir da janela remetente são encapsulados em uma *command*, e esta é enviada para a janela destinatária através do recurso *drag and drop* provisionado pelo sistema operacional.

Esta abordagem apresenta um problema: a janela destinatária que define quais os tipos de objeto que ela aceita. Contudo, o objeto deve ser encapsulado no início da operação *drag*. Como garantir que o tipo do objeto esperado pelo destino seja o mesmo encapsulado na mensagem se essa informação só estará disponível depois da operação *drop*?

Para tal, modificou-se a comunicação entre agentes para o *drag and drop*. Na nova configuração, a janela originária envia uma mensagem na qual ela encapsula sua própria identificação. A janela destinatária, tendo em mãos a identificação do remetente, envia os tipos de objeto que ela aceita, e a primeira envia o objeto do tipo em questão.

6.5 Considerações Finais

Durante a etapa de construção se comprovou entre os membros do grupo a importância de ter uma arquitetura bem definida. Toda a implementação dos módulos do bloco *Application* foi de baixa dificuldade, já que a comunicação estava simplificada pelo PAC e a construção e configuração de objetos complexos da camada *Business* era feito a partir de *factories*.

A parte mais custosa da etapa de construção foi a implementação das classes de *securities* e *trades* que possuem elevada complexidade, pois durante sua construção notou-se a necessidade de remodelagem para atender requisitos antes não esclarecidos.

7 *Transição*

*“Um programa de reembolso por software defeituoso
talvez fosse ótimo, exceto pelo fato de levar à
bancarrota toda a indústria de software no
primeiro ano.”*

Andrew S. Tanenbaum

7.1 Introdução

Neste capítulo são apresentados os testes realizados sobre o sistema, foram realizados testes unitários, modulares, de integração e de requisitos não-funcionais. É apresentado também o produto final do trabalho, especificamente a versão final do sistema. No final, o capítulo apresenta a aceitação das partes competentes quanto ao sistema desenvolvido.

7.2 Testes

7.2.1 Testes Unitários

Nos testes unitários, verificou-se se o funcionamento de uma classe ou grupo de classes para verificar se sua implementação foi efetuada corretamente. Esta classe ou grupo de classes não necessariamente mapeiam uma funcionalidade do sistema. Os testes unitários foram realizados sobre componentes de todos os blocos do sistema, incluindo o *Framework* e as classes de negócio.

Em geral, para esta categoria de testes criaram-se projetos especiais no Visual Studio com intuito de verificar e validar os dados de saída, em função de dados de entrada pré-definidos. Utilizou-se a biblioteca NUnit (NUNIT.ORG, 2008), uma plataforma de testes para o Microsoft.NET que permite maior automação e controle sobre o processo de verificação.

7.2.2 Testes Modulares

Os testes modulares foram realizados sobre as interfaces do sistema, verificando se as funcionalidades realizadas condizem com os requisitos. Desta forma, verificou-se o funcionamento de cada módulo como uma caixa preta, testando-se o comportamento conjunto de diversos objetos ou grupos de objetos já validados pelos testes unitários, sem haver ainda a preocupação de teste quanto à comunicação entre módulos.

7.2.3 Testes de Integração

Devido à utilização do PAC e dos *command*, os testes de integração se resumiram em verificar o fluxo de mensagens entre os agentes. Os testes devem:

- Verificar se a mensagem é gerada corretamente;

- Verificar se ela é uma mensagem local (a mensagem é endereçada para algum componente dentro do próprio agente PAC) ou externa (a mensagem deve ser roteada para outro agente);
- Verificar se a mensagem é recebida apenas pelo destinatário correto, dado seu nome ou *tags* (ver tabela 4 da seção 5.5.1.3);
- Verificar se a mensagem é tratada corretamente pelo destinatário;
- Verificar se o conteúdo da mensagem está correto.

Observa-se que, com o desacoplamento, os testes de integração são mais simples de serem realizados, garantindo assim uma manutenção do sistema mais fácil.

7.2.4 Testes de Requisitos Não-Funcionais

- Qualidade do Processo: Incluem a análise de relatórios de erros, levantando os tempos até falhas do sistema;
- Configurabilidade da Interface: São qualitativos e baseados na experiência do usuário com o sistema;
- Desempenho: Incluem testes de carga sobre a utilização de memória, de rede, de tempo de resposta, entre outros;
- Disponibilidade: Incluem a análise da disponibilidade dos servidores da qual o sistema depende (banco de dados e fontes de dados externas);
- Escalabilidade: Incluem análise de gargalo e montagem de cenários com diferentes configurações para determinar a tendência do desempenho nos diferentes módulos com o aumento de usuários, *trades*, entre outros;
- Manutenibilidade: Inclui a verificação da documentação do sistema através de uma análise de qualidade;
- Segurança de Acesso: Inclui testes da resistência a ataques de segurança de acesso;
- Usabilidade: São qualitativos e baseados na experiência do usuário com o sistema.

7.3 Sistema Final

A seguir, apresenta-se brevemente a interface com o usuário dos módulos do sistema final. Observam-se diversas mudanças em relação ao protótipo de telas apresentado no apêndice B, que indicam um aumento com relação à maturidade em relação ao sistema durante o processo de desenvolvimento.

7.3.1 Módulo Principal

A tela do módulo principal permite a abertura dos outros módulos funcionais e a alteração da *work area* (figura 22). As *work areas* permitem ao usuário agrupar as janelas de modo a facilitar a visualização de certa atividade. No sistema final, permite-se que haja um número customizável de *work areas*. Algumas diferenças observadas em relação ao protótipo inicial é a janela para autenticação separada do módulo principal e a possibilidade de se bloquear as funcionalidades do sistema.

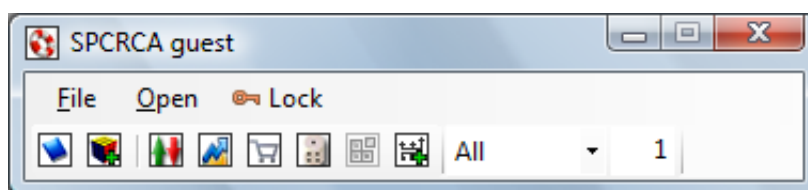


Figura 22: Tela final do Módulo Principal

7.3.2 Book View

A tela *Book View* apresenta a carteira de ativos como uma árvore (figura 23). É possível realizar funções sobre um *trade* ou um grupo de *trades*. Os *trades* podem agrupados em pastas, e estas recebem *tags*, sendo permitida a reordenação da carteira através da hierarquização pelas *tags* definidas. À direita, visualizam-se as propriedades da *trade* selecionada.

7.3.3 Trade Builder e Security Builder

O módulo *Trade Builder* apresentou diversas mudanças em relação à sua concepção inicial. No protótipo, projetou-se que um *trade* seria uma árvore que representava a composição de diversos valores. No sistema final, este módulo foi dividido em um módulo de mesmo nome e um denominado *Security Builder*.

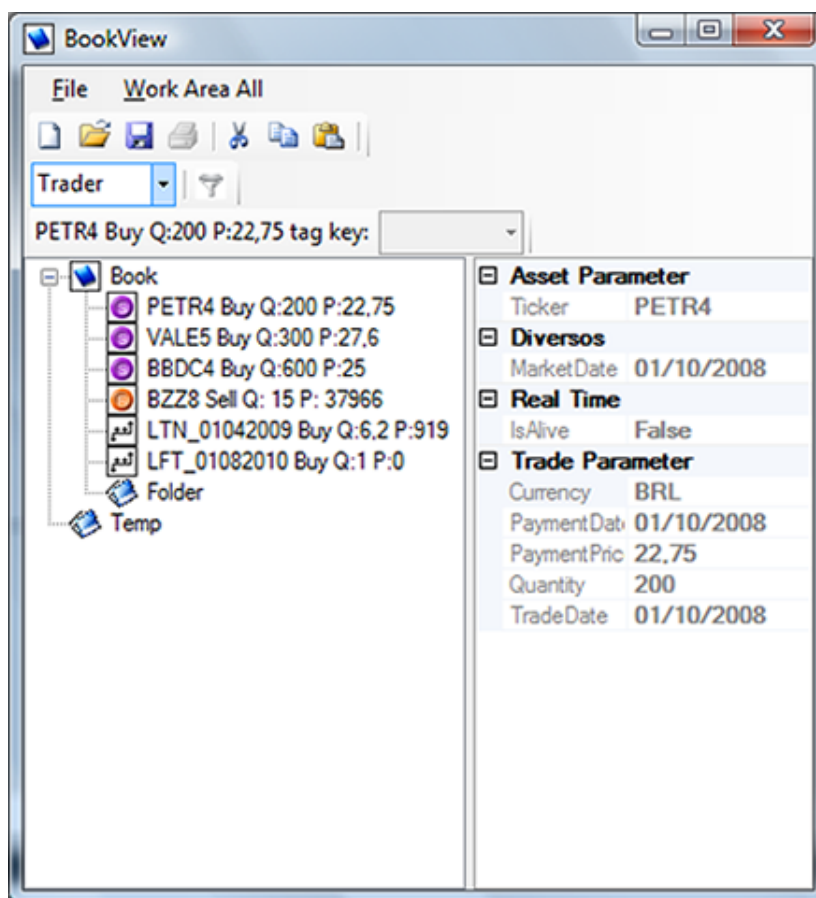


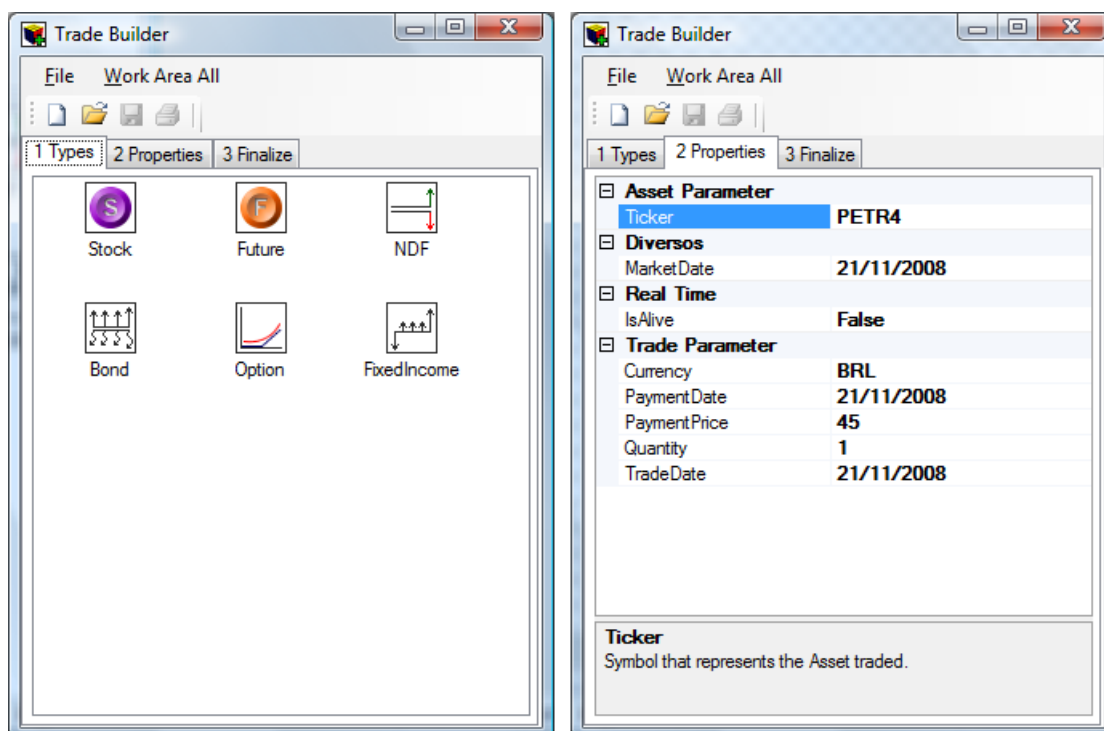
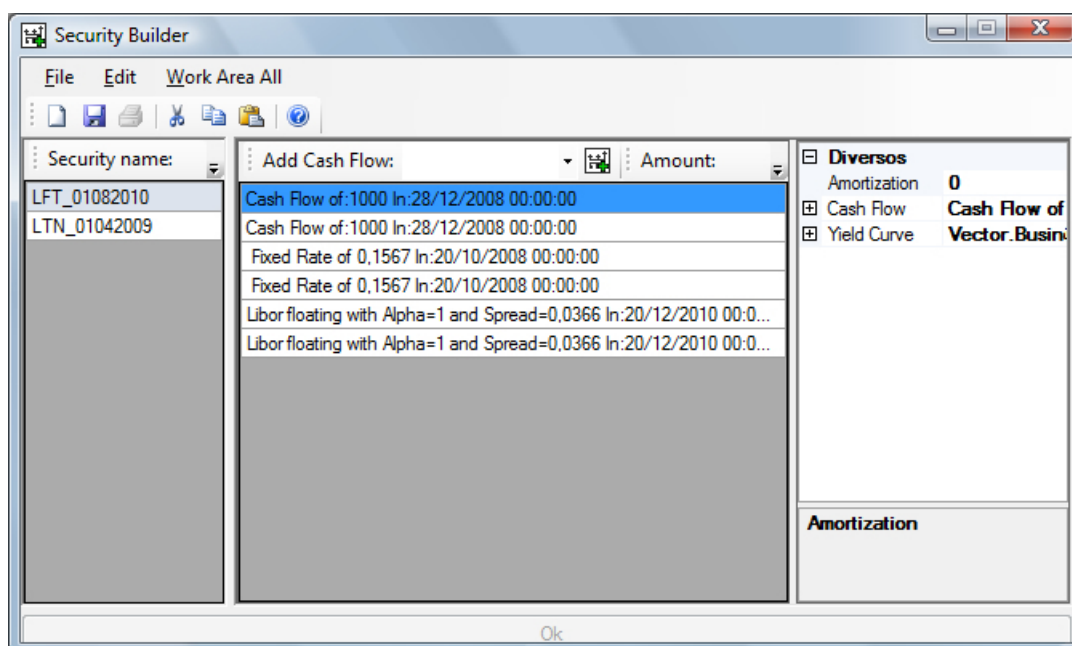
Figura 23: Tela final do módulo *Book View*

No novo *Trade Builder*, a criação de um *trade* foi dividida em três etapas, sendo a primeira a escolha do tipo, a segunda a atribuição de valores para os parâmetros de configuração, e a terceira o *trade* a ser utilizado nos outros módulos do sistema (figura 24).

No *Security Builder*, é permitida uma maior customização, uma vez que a criação da *security* é obtida através da agregação de vários fluxos de caixa, conforme a figura 25.

7.3.4 Market Pricing and Risk

No sistema, as análises de precificação e de risco de mercado foram incorporadas em um único módulo, devido à sua similaridade funcional. O novo *Market Pricing and Risk* mostra o preço de um ativo e variáveis relevantes para a análise de risco na forma de uma tabela, sendo possível escolher a moeda na qual os valores são exibidos, a data do mercado e a soma dos valores selecionados (figura 26).

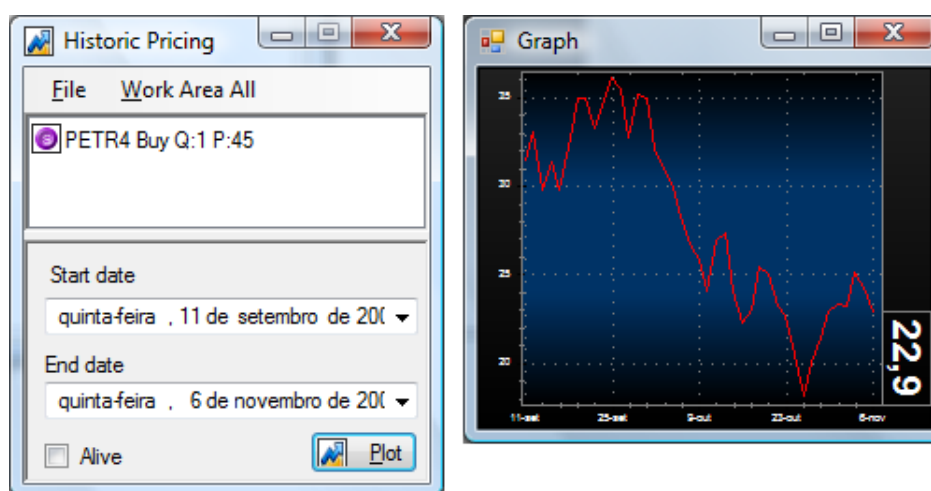
Figura 24: Tela final do módulo *Trade Builder*Figura 25: Tela final do módulo *Security Builder*

7.3.5 Historic Pricing

A tela de *Historic Pricing*, assim como identificado no protótipo, consiste de um gráfico do preço em um dado período de tempo. Viu-se necessária a construção de uma tela anterior para a identificação da data inicial e final da análise, conforme a figura 27.

Market Object	Market Variable	NDF RUSD PBRL 1.6 1
FX	BRL	2.15163206943179
FX	USD	0
Pricing	Price	0.592235498359147

Sum of the selected cells: 2.74386756779093

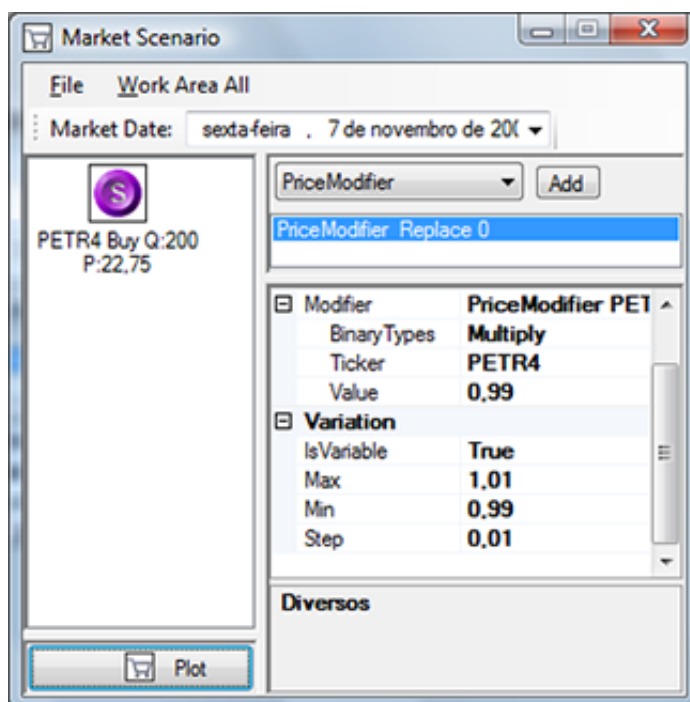
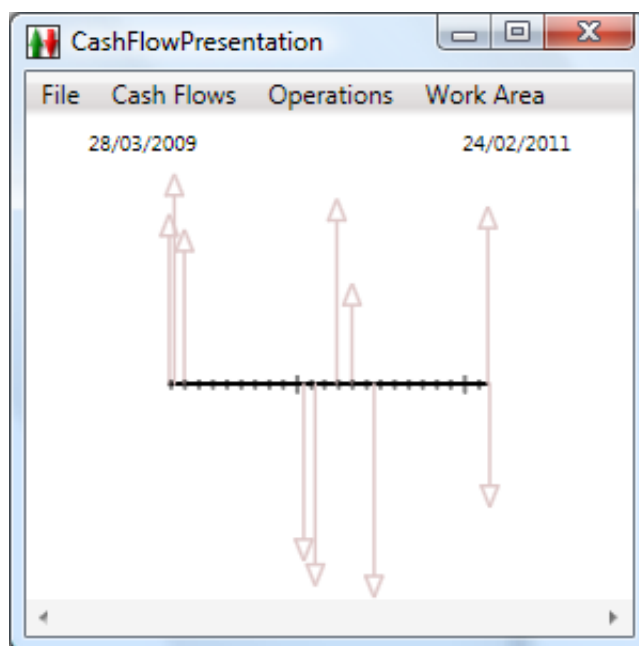
Figura 26: Tela final do módulo *Market Pricing and Risk*Figura 27: Tela final do módulo *Historic Pricing*

7.3.6 Market Scenario

Pela tela de *Market Scenario*, é possível configurar um cenário de mercado, selecionando-se modificadores para uma variável em valores absolutos ou porcentagem (figura 28). Esses modificadores permitem uma análise mais elaborada que aquela definida inicialmente no protótipo. O resultado final é plotado em um gráfico.

7.3.7 Cash Flows

Igualmente ao previsto pelo protótipo inicial, a tela *Cash Flows* apresenta os fluxos de caixa do *trade* ou conjunto de *trades* em uma forma gráfica, conforme ilustra a figura 37. Contudo, é possível identificar entre os fluxos desenhados quais pertencem a um determinado *trade*.

Figura 28: Tela final do módulo *Market Scenario*Figura 29: Tela final do módulo *Cash Flows*

7.4 Aceitação do Sistema

A primeira versão do sistema foi aprovada por todas as partes competentes e foi, portanto, aceita.

7.5 Considerações Finais

A ênfase na arquitetura que dirigiu todo o processo de desenvolvimento apresenta frutos positivos já na fase de Transição. Com o desacoplamento inerente ao padrão arquitetural adotado, observou-se a facilidade na realização dos testes. Depois de aplicados testes unitários automatizados para as unidades funcionais mínimas dos diversos *packages*, a independência entre estes componentes permitiu testes modulares consistentes e simplificados, sendo possível concentrar os esforços na verificação da funcionalidade do módulo como um todo ao invés de se checar a dependência entre as diversas unidades constituintes do módulo. Da mesma forma, os testes de integração consideravam cada módulo como uma caixa preta, validando-se apenas a comunicação entre os agentes do PAC.

Além disso, com o sistema final implementado, é possível facilmente justificar a utilização de um processo de desenvolvimento iterativo e incremental o Processo Unificado. A crescente maturidade em relação ao sistema por todos os interessados no projeto mesmo após o início da construção exige que escopo, requisitos e modelos sejam revistos constantemente de modo a garantir não só um sistema que atenda plenamente a necessidade do cliente, mas também que o sistema se mantenha consistente mesmo com as mudanças ocorridas.

8 *Considerações Finais*

“No fim tudo dá certo. Se não deu certo é porque ainda não chegou ao fim.”

Anônimo

8.1 Conclusões

Este trabalho permitiu conclusões em diversas áreas do conhecimento. Primeiramente, ressalta-se a justificativa de negócios do sistema. O SPCRCA de fato provê uma solução mais robusta quando comparada com as anteriormente utilizadas, estas essencialmente baseadas em *spreadsheets*, uma vez que possibilita tratamento de uma grande quantidade de dados sem perda considerável de desempenho, permite fácil inserção de novas análises e maior configurabilidade do sistema, proporciona manutenção simplificada, entre outros. Mais que isso, proporciona ferramentas que antes não eram viáveis, como por exemplo acesso a dados em tempo real e restrição de acesso a nível de funcionalidades do sistema.

Em seguida, apresentam-se aspectos da vertente técnica. A técnica de construção de um protótipo de IHC no início da fase de Elaboração do sistema, antes mesmo da especificação dos casos de uso, mostrou-se eficiente no contexto deste sistema, por prover tanto à equipe de desenvolvimento quanto principalmente ao cliente uma idéia mais concreta em relação ao produto final, havendo assim maior consistência dos requisitos com as necessidades de negócio, diminuindo, conseqüentemente, os riscos de projeto.

A utilização de Arquitetura de Software orientada a padrões possibilitou o atendimento de requisitos não-funcionais anteriormente levantados, como a reusabilidade, escalabilidade e manutenibilidade. O maior esforço despendido na especificação e na modelagem, embora causasse a princípio um maior desconforto no início da construção devido à falta de familiaridade com o uso de *patterns*, foi responsável por uma maior agilidade na fase de Construção como um todo, devido à maior organização que a ênfase na arquitetura proporcionou. Na fase de Transição, o desacoplamento proporcionou ganhos principalmente nos testes de integração por diminuir a interdependência entre os diversos módulos.

Como exemplos concretos da vantagem de se utilizar POSA, ressaltam-se o fácil gerenciamento de objetos proporcionado pela utilização de *factories* e a integração do simulador do provedor de dados em tempo real, este possível com modificação de apenas uma única linha de código. A própria adição de novos módulos no sistema apresentou-se trivial. O próprio módulo de *Market Pricing and Risk* necessitou de cerca de 10 homens-hora para a codificação e testes, dada a biblioteca de funções de negócios já implementada. Mais que isso, o controle de toda a complexidade do sistema foi facilitada. Foram contabilizadas 504 classes construídas, mas há também as classes pertencentes a bibliotecas externas ao projeto. A gestão de todas elas seria de dificuldade tamanha se não fosse a abordagem sistemática aos modelos escolhidos.

Por fim, a escolha do processo de desenvolvimento mostrou-se apropriada. O Processo Unificado, por sua natureza iterativa e incremental, forneceu suporte ao aumento de maturidade em relação ao sistema final por parte dos interessados no projeto em todas as fases do ciclo de vida. Um exemplo claramente visível é observável nas diferenças do protótipo de telas inicial, proposto no início da Elaboração, e o sistema final. Contudo, além do aumento de maturidade, outra justificativa para a escolha deste processo é a orientação para a arquitetura do sistema. É necessária que ela esteja condizente com os requisitos funcionais e não-funcionais, estes levantados ainda na fase de Concepção, e que a modelagem expresse os aspectos definidos na arquitetura, sendo assim imprescindível a comunicação entre os processos de levantamento de requisitos, análise e *design*.

Desta forma, pode-se afirmar que as escolhas técnicas e gerenciais realizadas durante a evolução deste trabalho permitiu o atendimento dos requisitos levantados junto ao cliente, e o sistema como um todo tanto satisfaz as necessidades previstas dos usuários como também aumentou a robustez das soluções anteriormente utilizadas para este intuito.

8.2 Contribuições

As contribuições obtidas com o SPCRCA são observadas primordialmente no âmbito do mercado financeiro, no que diz respeito a um sistema de avaliação de risco para uma carteira de ativos. As características do sistema, principalmente quando confrontadas com as das soluções utilizadas atualmente, indicam vantagens da associação de Tecnologia da Informação com as áreas de negócio de uma empresa, provendo soluções que melhor atendam as necessidades existentes.

Da vertente técnica, obteve um estudo de caso positivo quanto à aplicação de ferramentas das disciplinas de Engenharia de Software e Gerência de Projetos no processo de desenvolvimento de um sistema de software, provando mais uma vez que a complexidade pode ser gerenciada mais facilmente quando o próprio sistema é fundamentado sobre técnicas que já prevêm estes problemas.

8.3 Trabalhos Futuros

Para as novas versões do SPCRCA, prevê-se a criação do módulo de *Profit and Loss Explain* (ou *P&L Explain*), que gera a explicação de fatores de risco da variação do preço de um *trade* entre duas datas, sendo assim utilizada para analisar e interpretar a origem

dos lucros ou prejuízos decorrentes. Além disso, novas análises de risco são previstas para o módulo *Market Pricing and Risk*.

Referências

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. Boston, MA, EUA: Addison-Wesley, 1997. ISBN 0-2011-9930-0.

BHATTI, S. N. Why quality?: ISO 9126 software quality metrics (functionality) support by UML suite. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, EUA, v. 30, n. 2, p. 1–5, 2005. ISSN 0163-5948.

BM&F BOVESPA. *Contrato Futuro de Cupom Cambial*. 2008. URL: <http://www.bmf.com.br>. Acessado em: 26/08/2008.

BM&F BOVESPA. *Contrato Futuro de Ibovespa - Especificações*. 2008. URL: <http://www.bmf.com.br>. Acessado em: 26/08/2008.

BM&F BOVESPA. *Especificações do Contrato Futuro de Taxa de Câmbio de Reais por Dólar Comercial*. 2008. URL: <http://www.bmf.com.br>. Acessado em: 26/08/2008.

BM&F BOVESPA. *Especificações do Contrato Futuro de Taxa Média de Depósitos Interfinanceiros de Um Dia*. 2008. URL: <http://www.bmf.com.br>. Acessado em: 26/08/2008.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *The Unified Modeling Language User Guide*. Redwood City, CA, EUA: Addison-Wesley, 1999. ISBN 0-2015-7168-4.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *The Unified Software Development Process*. Boston, MA, EUA: Addison-Wesley, 1999. ISBN 0-2015-7169-2.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*. [S.l.]: Wiley and Sons, 2007. ISBN 0-4700-5902-8.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. [S.l.]: Wiley and Sons, 2007. ISBN 0-4714-8648-5.

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture: A System of Patterns*. [S.l.]: Wiley and Sons, 1996. ISBN 0-4719-5869-7.

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. [S.l.]: Wiley and Sons, 2000. ISBN 0-4716-0695-2.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, EUA: Addison-Wesley, 1995. ISBN 0-2016-3361-2.

HULL, J. C. *Options, Futures and Other Derivatives*. [S.l.]: Prentice Hall, 2006. ISBN 0-13-149908-4.

IBM. *IBM Rational Software*. 2008. URL: <http://www.rational.com/>. Acessado em: 07/08/2008.

IEEE. IEEE standard 982.2-1988, IEEE guide for the use of IEEE standard dictionary of measures to produce reliable software. 1988.

IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. [S.l.], 1990. (IEEE Standard, 610.121990).

IEEE, ACM. *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. 2004.

KIRCHER, M.; JAIN, P. *Pattern-Oriented Software Architecture: Patterns for Resource Management*. [S.l.]: Wiley and Sons, 2007. ISBN 0-4708-4525-2.

MAK, J. K. H.; CHOY, C. S. T.; LUN, D. P. K. Precise modeling of design patterns in UML. In: *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, EUA: IEEE, 2004. p. 252–261. ISBN 0-7695-2163-0.

MEI, H. A complementary approach to requirements engineering: Software architecture orientation. *SIGSOFT Software Engineering Notes*, ACM, New York, NY, EUA, v. 25, n. 2, p. 40–45, 2000. ISSN 0163-5948.

NETO, A. A. *Mercado Financeiro*. [S.l.]: Ed. Atlas, 2005. ISBN 85-224-3971-0.

NEWELL, M. W.; GRASHINA, M. N. *The Project Management Question and Answer Book*. [S.l.]: Amacom, 2004. ISBN 0-8144-7164-1.

NUNIT.ORG. *NUnit*. 2008. URL: <http://www.nunit.org>. Acessado em: 14/11/2008.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 6th. ed. Boston, MA, EUA: McGraw Hill, 2004. ISBN 0-0728-5318-2.

PROJECT MANAGEMENT INSTITUTE. *A Guide to the Project Management Body of Knowledge (PMBOK)*. 6th. ed. [S.l.], 2004. ISBN 1-9306-9945-X.

PROJECT MANAGEMENT INSTITUTE. *Practice Standard for Work Breakdown Structures*. 2th. ed. [S.l.], 2006. ISBN 1-9338-9013-4.

PROJECT MANAGEMENT INSTITUTE. *Practice Standard for Scheduling*. [S.l.], 2007. ISBN 1-9306-9984-0.

ROSENBERG, T. H. L.; SHAW, J. *Software Metrics and Reliability*. 1998.

SCHUMACHER, M. et al. *Security Patterns: Integrating Security and Systems Engineering*. [S.l.]: Wiley and Sons, 2005. ISBN 0-4708-5884-2.

SODHI, J.; SODHI, P. *IT Project Management Handbook*. Vienna, VA, Austria: Management Concepts, 2001. ISBN 1-5672-6098-5.

TAIBI, T. Formal specification of design patterns: A balanced approach. In: *ACST*. [S.l.]: IASTED/ACTA, 2006. p. 310–315. ISBN 0-8898-6547-7.

WILMOTT, P. *Derivatives: The Theory and Practice of Financial Engineering*. [S.l.]: Wiley and Sons, 1998. ISBN 0-4719-8366-7.

YOUNG, R. R. *The Requirements Engineering Handbook*. Boston, MA, EUA: Artech House, 2004. ISBN 1-5805-3266-7.

ZEDGRAPH.ORG. *ZedGraphWiki*. 2008. URL: <http://zedgraph.org>. Acessado em: 14/11/2008.

APÊNDICE A - Planejamento de Atividades

A.1 Visão Geral

Identificou-se, para as atividades inicialmente apresentadas no WBS, as tarefas necessárias para a execução deste projeto.

A.2 Estimativa das Atividades e Cronograma

A seguir, apresenta-se uma tabela que relaciona, para cada tarefa, sua respectiva duração, data de início, data de fim e quantidade de homens/hora para completá-la.

Tabela 5: Estimativa das Atividades e Cronograma

Tarefa	Duração (Horas)	Qtd HH	Início	Fim
PLANEJAMENTO	26	58,8	14/01/2008	20/02/2008
Definição dos objetivos	5	10,5	14/01/2008	18/01/2008
Definição do escopo de projeto	5	10,5	21/01/2008	25/01/2008
Análise de viabilidade	4	8,4	28/01/2008	31/01/2008
Análise de riscos	4	8,4	01/02/2008	06/02/2008
Estimativas do projeto	10	21	07/02/2008	20/02/2008
Artefato: Documento de "Planejamento de Projeto v1.0"	0	0	20/02/2008	20/02/2008
CONCEPÇÃO	50	308,7	14/01/2008	21/03/2008
Definição dos requisitos funcionais	19	39,9	30/01/2008	25/02/2008
Definição dos requisitos não-funcionais	19	39,9	30/01/2008	25/02/2008
Definição das funções do sistema	19	39,9	30/01/2008	25/02/2008
Definição das interfaces	19	39,9	30/01/2008	25/02/2008
Definição dos critérios de aceitação	19	39,9	30/01/2008	25/02/2008
Artefato: Documento de "Especificação de Requisitos v1.0"	0	0	25/02/2008	25/02/2008
Detalhamento dos requisitos funcionais	8	16,8	27/02/2008	07/03/2008
Detalhamento dos requisitos não-funcionais	8	16,8	27/02/2008	07/03/2008
Detalhamento das interfaces	8	16,8	27/02/2008	07/03/2008
Geração do projeto de IHC preliminar	10	21	27/02/2008	11/03/2008
Artefato: Documento de "Projeto de IHC v1.0"	0	0	11/03/2008	11/03/2008
Geração do modelo de casos de uso preliminar	9	18,9	11/03/2008	21/03/2008
Artefato: Documento de "Modelo de Casos de Uso v1.0"	0	0	21/03/2008	21/03/2008
Geração do modelo de classes preliminar	9	18,9	11/03/2008	21/03/2008
Artefato: Documento de "Modelo de Classes v1.0"	0	0	21/03/2008	21/03/2008
Artefato: Documento de "Especificação de Requisitos v2.0"	0	0	21/03/2008	21/03/2008
ELABORAÇÃO	29	128,1	24/03/2008	01/05/2008
Detalhamento do projeto de IHC	3	6,3	24/03/2008	26/03/2008
Artefato: Documento de "Projeto de IHC v2.0"	0	0	26/03/2008	26/03/2008
Detalhamento do modelo de casos de uso	5	10,5	24/03/2008	28/03/2008
Artefato: Documento de "Modelo de Casos de Uso v2.0"	0	0	28/03/2008	28/03/2008
Detalhamento do modelo de classes	5	10,5	24/03/2008	28/03/2008
Artefato: Documento de "Modelo de Classes v2.0"	0	0	28/03/2008	28/03/2008
Artefato: Documento de "Especificação de Requisitos v3.0"	0	0	28/03/2008	28/03/2008
Definição da arquitetura do sistema	10	21	31/03/2008	11/04/2008
Artefato: Documento de "Arquitetura do Sistema v1.0"	0	0	11/04/2008	11/04/2008
Geração do modelo de dados	8	16,8	31/03/2008	09/04/2008
Artefato: Documento de "Projeto de Banco de Dados v1.0"	0	0	09/04/2008	09/04/2008
Geração de diagramas de estados	8	16,8	31/03/2008	09/04/2008
Geração de diagramas de sequência e matriz de incidência	8	16,8	31/03/2008	09/04/2008
Artefato: Documento de "Modelo Dinâmico v1.0"	0	0	09/04/2008	09/04/2008
Artefato: Documento de "Especificação de Requisitos v4.0"	0	0	11/04/2008	11/04/2008
Detalhamento do plano de testes	12	25,2	14/04/2008	29/04/2008
Artefato: Documento de "Plano de Testes v1.0"	0	0	30/04/2008	01/05/2008
Revisão de todos os modelos	2	4,2	14/04/2008	15/04/2008
CONSTRUÇÃO	122	422,1	02/05/2008	20/10/2008
Desenvolvimento de protótipos de cada módulo	12	50,4	02/05/2008	19/05/2008
Integração dos protótipos dos módulos	7	29,4	20/05/2008	28/05/2008
Artefato: Protótipo mínimo do sistema	0	0	28/05/2008	28/05/2008
Desenvolvimento do banco de dados	2	8,4	29/05/2008	30/05/2008
Desenvolvimento da camada de dados	5	21	02/06/2008	06/06/2008
Artefato: Camada de dados	0	0	06/06/2008	06/06/2008
Desenvolvimento das interfaces com sistemas externos	4	16,8	09/06/2008	12/06/2008
Integração com sistemas externos	4	16,8	13/06/2008	18/06/2008
Artefato: Interface com sistemas externos	0	0	18/06/2008	18/06/2008
Desenvolvimento da interface com o usuário	2	8,4	19/06/2008	20/06/2008
Artefato: Interface com o usuário	0	0	20/06/2008	20/06/2008
Desenvolvimento de componentes abstratos e interfaces da camada de negócios	7	29,4	23/06/2008	01/07/2008
Desenvolvimento de módulo funcional unitário	11	46,2	02/07/2008	16/07/2008
Artefato: Módulo funcional unitário	0	0	16/07/2008	16/07/2008
Desenvolvimento dos módulos funcionais	50	157,5	17/07/2008	24/09/2008
Artefato: Módulos funcionais	0	0	24/09/2008	24/09/2008
Integração do sistema	14	29,4	25/09/2008	14/10/2008
Artefato: Integração do sistema	0	0	14/10/2008	14/10/2008
Artefato: Sistema SPCRCA v1.0 Alpha 1	0	0	14/10/2008	14/10/2008
Revisão de todos os documentos	4	8,4	15/10/2008	20/10/2008
TRANSIÇÃO	29	86,1	21/10/2008	28/11/2008
Testes finais modulares	1	2,1	21/10/2008	21/10/2008
Correção de erros	2	4,2	22/10/2008	23/10/2008
Artefato: Sistema SPCRCA v1.0 Alpha 2	0	0	23/10/2008	23/10/2008
Testes finais de integração	1	2,1	24/10/2008	24/10/2008
Correção de erros	2	4,2	27/10/2008	28/10/2008
Artefato: Sistema SPCRCA v1.0 Alpha 3	0	0	28/10/2008	28/10/2008
Testes automatizados	1	2,1	29/10/2008	29/10/2008
Correção de erros	4	8,4	30/10/2008	04/11/2008
Testes alfa e correção de erros detectados	5	10,5	29/10/2008	04/11/2008
Artefato: Sistema SPCRCA v1.0 Release Candidate 1	0	0	04/11/2008	04/11/2008
Homologação	2	4,2	05/11/2008	06/11/2008
Correção de inconformidades	5	10,5	07/11/2008	13/11/2008
Artefato: Sistema SPCRCA v1.0 Build 001	0	0	28/11/2008	28/11/2008
Artefato: Monografia	12	25,2	20/11/2008	20/11/2008
Revisão de todos os documentos	6	12,6	21/11/2008	28/11/2008

APÊNDICE B – Protótipo de Telas

B.1 Visão Geral

O protótipo de telas apresenta uma concepção inicial do sistema a ser desenvolvido no que diz respeito à interface com o usuário. Ele foi desenvolvido como forma de aumentar a maturidade do sistema, tanto para a empresa cliente quanto para a equipe de desenvolvimento.

B.2 Telas

B.2.1 Módulo Principal

A tela do módulo principal permite a abertura dos outros módulos funcionais e a alteração da *work area*. As *work areas* permitem ao usuário agrupar as janelas de modo a facilitar a visualização de certa atividade. No protótipo, a autenticação é feita no módulo principal, como mostra a figura 30.

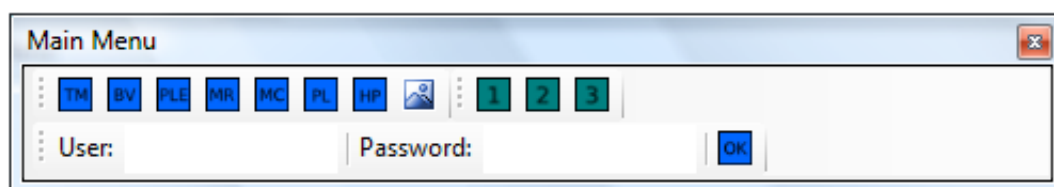
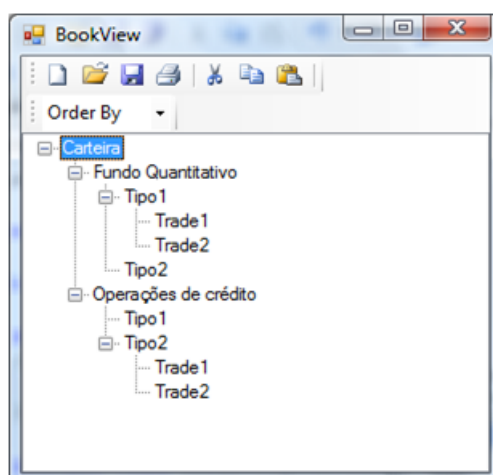


Figura 30: Protótipo de tela do Módulo Principal

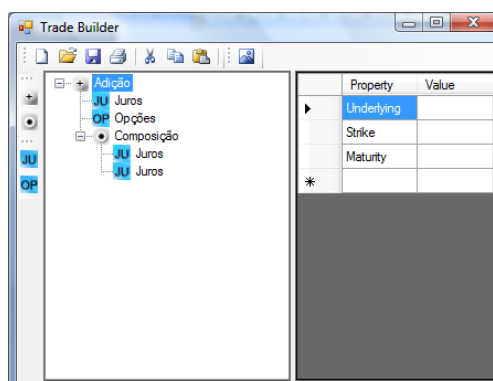
B.2.2 Book View

A tela *Book View* apresenta a carteira de ativos como uma árvore, na qual é possível agrupar *trades* de forma a customizar a visão que o usuário tem sobre a carteira (figura 31). É possível realizar funções sobre um *trade* ou um grupo de *trades*.

Figura 31: Protótipo de tela do *Book View*

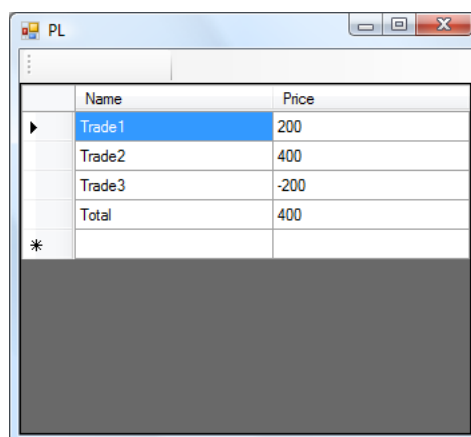
B.2.3 Trade Builder

A tela *Trade Builder* possui um menu com os tipos de *trades* e de operações, uma árvore na qual se monta o *trade* e uma tela de configuração (figura 32). Esta árvore permite a composição de um *trades* através da customização das operações previstas para valores diversos.

Figura 32: Protótipo de tela do *Trade Builder*

B.2.4 Market Pricing

A análise de precificação de ativos, obtida através do módulo *Market Pricing*, é feita no protótipo através de uma tabela que indica o preço de cada *trade*, e possivelmente a soma dos valores (figura 33).



A interface do Pricing (PL) apresenta uma tabela com as seguintes informações:

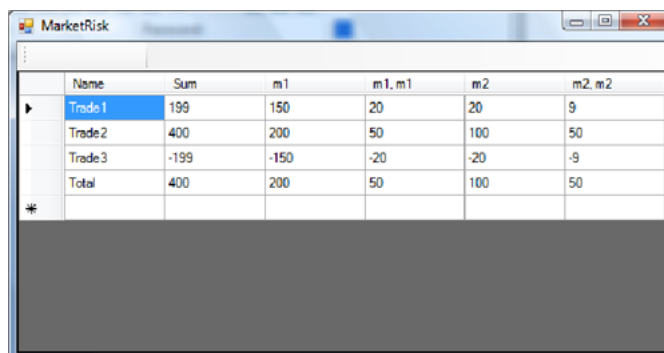
	Name	Price
▶	Trade1	200
	Trade2	400
	Trade3	-200
	Total	400
*		

Abaixo da tabela, há uma área cinza reservada para outras funcionalidades.

Figura 33: Protótipo de tela do Pricing

B.2.5 Market Risk

O risco de mercado é analisado pelo módulo *Market Risk*. Similarmente ao *Market Pricing*, o sistema exibe uma tabela que indica, para cada *trade*, as variáveis relevantes para o entendimento do risco (figura 34).



A interface do MarketRisk apresenta uma tabela com as seguintes informações:

	Name	Sum	m1	m1, m1	m2	m2, m2
▶	Trade1	199	150	20	20	9
	Trade2	400	200	50	100	50
	Trade3	-199	-150	-20	-20	-9
	Total	400	200	50	100	50
*						

Abaixo da tabela, há uma área cinza reservada para outras funcionalidades.

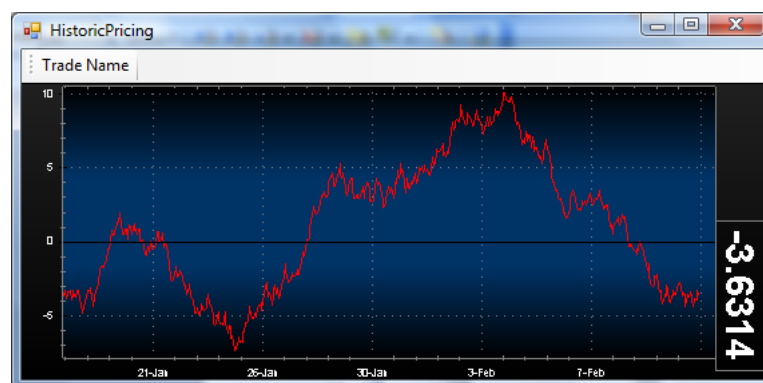
Figura 34: Protótipo de tela do *Market Risk*

B.2.6 Historic Pricing

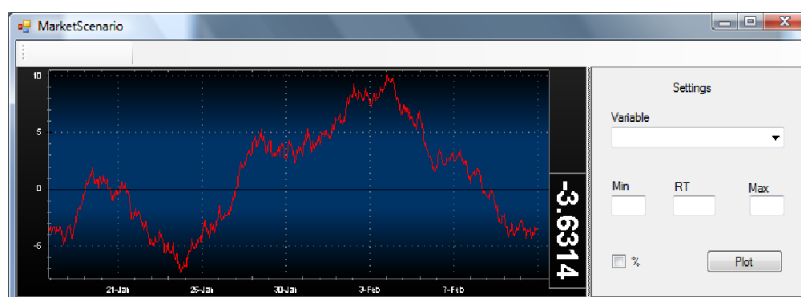
No protótipo, a tela de *Historic Pricing* consiste de apenas um gráfico que plota, para um dado *trade*, seu preço em um período de tempo (figura 35).

B.2.7 Market Scenario

A tela *Market Scenario* do protótipo consiste de uma área de configuração do cenário do mercado, onde é selecionada uma variável a ser analisada (como juros, câmbio, volatilidade, entre outros) e a variação dos valores desta (em valores absolutos ou porcentagem),

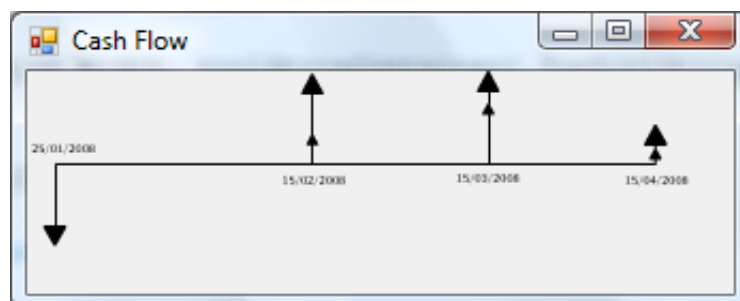
Figura 35: Protótipo de tela do *Historic Pricing*

conforme a figura 36. O gráfico desenhado apresenta o preço em função desta variável.

Figura 36: Protótipo de tela do *Market Scenario*

B.2.8 Cash Flows

A tela *Cash Flows* apresenta os fluxos de caixa do *trade* ou conjunto de *trades* em uma forma gráfica, conforme ilustra a figura 37.

Figura 37: Protótipo de tela do *Cash Flows*

APÊNDICE C - Modelo de Dados

C.1 Visão Geral

O modelo de dados é uma ferramenta conceitual para descrição dos dados do sistema. Esta descrição pode ser feita através do modelo entidade-relacionamento, identificando-se esses dados como entidades que se relacionam entre si. Para o SPCRCA, as entidades são organizadas em tabelas cuja estrutura física é apresentada na próxima seção.

Deve-se ressaltar que uma parte do modelo não foi desenvolvida no escopo deste projeto, mas sim utilizado pelo sistema em suas diversas funcionalidades. Isso decorre do fato que os dados presentes são utilizados em outros aplicativos da empresa cliente. Para estes casos, utilizou-se uma ferramenta de Engenharia Reversa para se obter o modelo de dados.

C.2 Estrutura

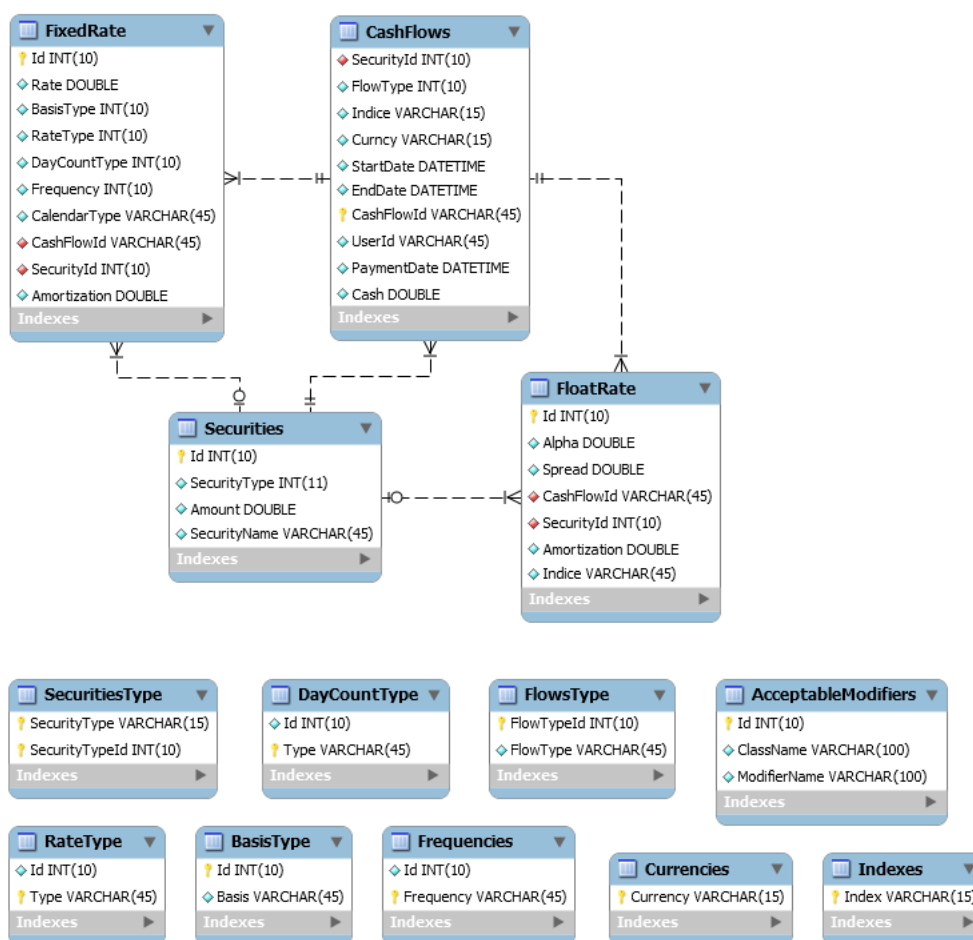
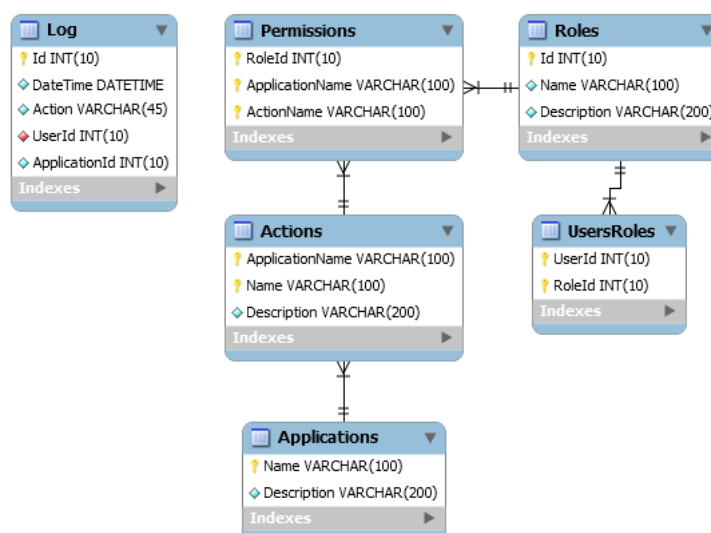
A estrutura física do modelo de dados para cada um dos *schemas* é apresentada a seguir.

C.2.1 Schema SPCRCA

Identifica os dados relevantes para a aplicação SPCRCA.

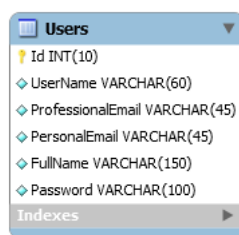
C.2.2 Schema Application

Define dados a serem utilizados por diversos sistemas, mas no nível da aplicação. Indica, por exemplo, os papéis que os usuários podem ter, e suas respectivas permissões.

Figura 38: Diagrama entidade-relacionamento do *schema* SPCRCAFigura 39: Diagrama entidade-relacionamento do *schema* Application

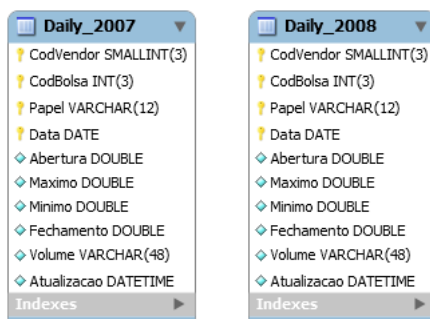
C.2.3 Schema Personal

Define os dados pessoais dos usuários.

Figura 40: Diagrama entidade-relacionamento do *schema* Personal

C.2.4 Schema Daily

Armazena dados do mercado diários.

Figura 41: Diagrama entidade-relacionamento do *schema* Daily

C.2.5 Schema Framework

Tabelas utilizadas para as bibliotecas presentes no *Framework*. Por exemplo, o Class-Translator guarda informações de *assembly* para diversas classes, sendo assim utilizada pelas *factories* para instanciação de objetos a partir de um nome.

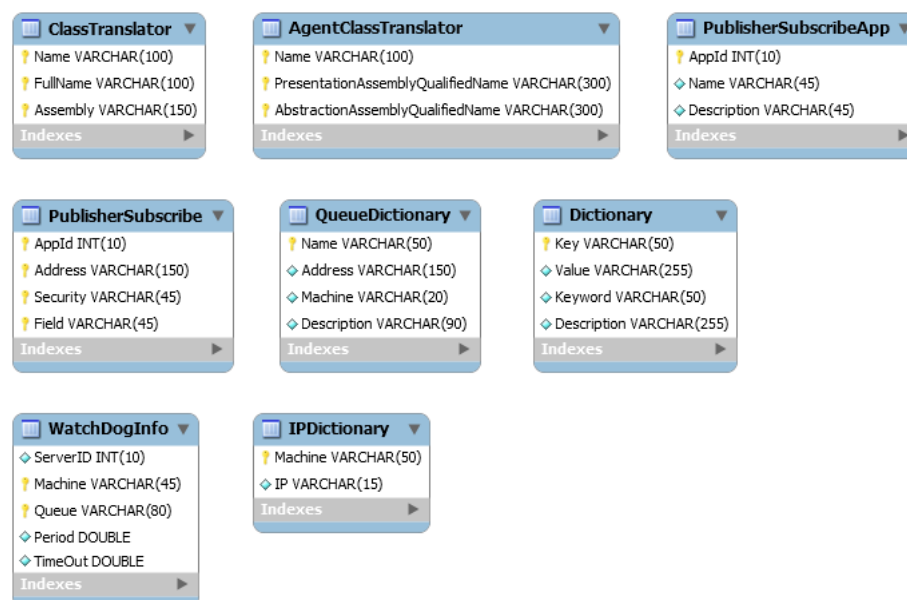


Figura 42: Diagrama entidade-relacionamento do *schema* Framework

APÊNDICE D – Métodos de Precificação e Risco

D.1 Visão Geral

A seguir, são abordados os métodos de formação de preço e de apuração de risco dos ativos financeiros tratados pelo SPCRCA .

D.2 Métodos de Precificação

O método de precificação a ser aplicado pelo SPCRCA dependerá do tipo do ativo financeiro.

D.2.1 Ações

o preço será determinado diretamente pela cotação atual do papel. Em alguns casos especiais, o preço do papel poderá ser determinado por um spread do papel em relação ao principal índice da Bovespa, o Ibovespa, por exemplo, o preço da ação PN da Petrobrás seria dado pela seguinte relação:

$$P_{PETRA} = Ibov + Spread_{PETRA} \quad (D.1)$$

No caso da apuração de fluxos históricos de pagamento de dividendos, será importante observar a ocorrência de eventos como bonificação, desdobramento e agrupamento. A idéia é que se um investidor possuir 100 ações na data x_1 e receber R\$ 0,50 por ação na mesma data, ao observar este evento na data x_2 , após a ocorrência de um desdobramento de 1 para 2 ações, o dividendo pago deverá ser visto como de R\$0,25 ao invés R\$0,50. Para os ajustes desse tipo, serão utilizados fatores provenientes de uma base de informações externa.

D.2.2 Contratos Futuros

os preços dos contratos futuros serão determinados da mesma forma que as Ações, ou seja, pela cotação do mercado. Estes, porém, devem ter os valores dos ajustes diários, inclusive para o dia da operação, calculados conforme as regras estabelecidas pela BM&F. Os ajustes diários agem sobre a margem de garantia do investidor, e servem de instrumento para evitar inadimplência por uma das partes envolvidas no contrato (HULL, 2006), contudo, ao invés da diferença entre o preço acordado em contrato e o preço à vista ser liquidada somente no vencimento, ela é ajustada aos poucos, com o passar do tempo.

Os ajustes a serem calculados serão para os seguintes contratos:

- Futuro de Índice Bovespa
- Futuro de Taxa Média DI
- Futuro de Taxa de Câmbio - Reais por Dólar
- Futuro de Cupom Cambial

D.2.2.1 Futuro de Índice Bovespa

O preço de fechamento do contrato futuro do Ibovespa é expresso em pontos de índice, que traduzem a média do Ibovespa à vista (BM&F BOVESPA, 2008b).

- ajuste das operações realizadas no dia:

$$AD_t = (PA_t - PO) * M * N \quad (D.2)$$

- ajuste das operações realizadas no dia anterior:

$$AD_t = (PA_{t-1} - PA) * M * N \quad (D.3)$$

onde

AD_t = valor do ajuste, em reais, referente à data t ;

PA_t = preço de ajuste do contrato, em pontos, na data t para o vencimento respectivo;

PO = preço da operação, em pontos;

M = valor em reais de cada ponto de índice, estabelecido pela BM&F

N = número de contratos negociados

D.2.2.2 Futuro de Taxa Média DI

O mercado futuro de DI negocia as taxas médias diárias DI de um dia, apuradas pela CETIP. Os contratos têm seu valor, em pontos, no vencimento fixados em 100.000 e são trazidos a valor presente descontando-se a taxa negociada (NETO, 2005). Os contratos de DI futuro são negociados em PU (preço unitário), sendo que cada PU vale R\$1,00 (BM&F BOVESPA, 2008d).

- ajuste das operações realizadas no dia:

$$AD_t = (PA_t - PO) * M * N \quad (D.4)$$

- ajuste das operações realizadas no dia anterior:

$$AD_t = [PA_t - (PA_{t-1} * FC_t)] * M * N \quad (D.5)$$

onde

AD_t = valor do ajuste, em reais, referente à data t;

PA_t = preço de ajuste do contrato, em pontos, na data t para o vencimento respectivo;

M = valor em reais de cada PU, estabelecido pela BM&F

N = número de contratos negociados

PA_{t-1} = preço de ajuste do contrato na data "t-1", para o vencimento respectivo;

FC_t = fator de correção do dia t, definido pelas seguintes fórmulas:

–quando houver saque-reserva entre o último pregão e o dia do ajuste:

$$FC_t = (1 + \frac{DI_{t-1}}{100})^{\frac{1}{252}} \quad (D.6)$$

–quando houver mais de um saque-reserva entre o último pregão e o dia do ajuste:

$$FC_t = \prod_{1 \leq j \leq n} (1 + \frac{DI_j}{100})^{\frac{1}{252}} \quad (D.7)$$

DI_{t-1} = taxa DI referente ao dia útil anterior ao dia que o ajuste se refere, com até seis casas decimais.

PO = preço da operação, em PU, calculado após o fechamento dos negócios pela seguinte fórmula:

$$PO = \frac{100000}{(1 + \frac{i}{100})^{\frac{n}{252}}} \quad (D.8)$$

onde

i = taxa de juro negociada;

n = número de saques-reserva (alterações na margem de garantia) compreendido entre a data de negociação, inclusive, e a data de vencimento do contrato;

D.2.2.3 Futuro de Taxa de Câmbio - Reais por Dólar

Os contratos futuros de câmbio de reais por dólar dos Estados Unidos são cotados em reais por R\$1.000, logo, se a taxa de câmbio futuro é de 1,678, um contrato futuro estaria cotado em R\$1.678,000 (os contratos são cotados até a terceira casa decimal). Cada contrato é negociado com valor correspondente a US\$ 50.000,00 (BM&F BOVESPA, 2008c).

- ajuste das operações realizadas no dia:

$$AD = (PA_t - PO) * M * N \quad (D.9)$$

- ajuste das operações realizadas no dia anterior:

$$AD = (PA_{t-1} - PA) * M * N \quad (D.10)$$

onde

AD = valor do ajuste diário;

PA_t = preço de ajuste do contrato no dia;

PO = preço da operação;

M = multiplicador do contrato estabelecido em 50, estabelecido pela BM&F

N = número de contratos negociados

D.2.2.4 Futuro de Cupom Cambial

O cupom cambial negociado no mercado futuro é a “taxa de juro obtida a partir a partir do cálculo da diferença entre a acumulação da taxa DI, no período compreendido entre a data de operação, inclusive, e a data de vencimento do contrato” (BM&F BOVESPA, 2008a). Os contratos são negociados de forma semelhante aos contratos de DI, ou seja, possuem valor em pontos correspondentes a 100000, e são descontados pela taxa negociada no contrato.

- ajuste das operações realizadas no dia:

$$AD_t = (PA_t - PO) * M * N * TC_{t-1} \quad (D.11)$$

- ajuste das operações realizadas no dia anterior:

$$AD_t = [PA_t - (PA_{t-1} * FC_t)] * M * N * TC_{t-1} \quad (D.12)$$

onde

AD_t = valor do ajuste, em reais, referente à data t;

PA_t = preço de ajuste do contrato, em pontos, na data t para o vencimento respectivo;

M = valor em reais de cada PU, estabelecido pela BM&F

N = número de contratos negociados

PA_{t-1} = preço de ajuste do contrato na data "t-1", para o vencimento respectivo;

TC_{t-1} = taxa de câmbio verificada na data "t-1";

FC_t = fator de correção do dia t, definido pela seguinte fórmula:

$$FC_t = \frac{\prod_{1 \leq j \leq m} (1 + \frac{DI_{t-j}}{100})^{\frac{1}{252}}}{\frac{TC_{t-1}}{TC_{t-k}}} \quad (D.13)$$

m = número de saques-reserva (alterações na margem de garantia) compreendido entre a data t e a data do pregão anterior;

DI_{t-j} = taxa DI referente ao j-énésimo dia útil anterior à data t, com até seis casas decimais.

PO = preço da operação, em PU, calculado após o fechamento dos negócios pela seguinte fórmula:

$$PO = \frac{100000}{(\frac{i_{op}}{100} * \frac{n}{360}) + 1} \quad (D.14)$$

onde

i_{op} = taxa de juro negociada;

n = número de dias corridos compreendidos no período entre a data da operação, inclusive, e a data do vencimento, exclusive;

D.2.3 Títulos de Renda Fixa

Existem diversos tipos de Títulos de Renda Fixa no mercado nacional, mas de uma maneira geral pode-se dizer que o preço de um título é igual ao valor presente de seu

fluxo de caixa esperado, ou seja, a somatória dos valores presentes de cada fluxo. Considerando o fluxo de caixa abaixo, o preço do título será determinado do seguinte modo:

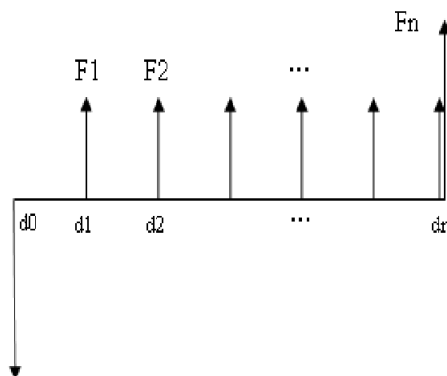


Figura 43: Representação dos fluxos gerados por um título de renda fixa

$$P = \sum \frac{F_n}{(1 + t)^{\frac{d_n}{252}}} \quad (\text{D.15})$$

onde

F_n = Valor futuro de cada fluxo;

d_n = número de dias úteis entre a data presente e o data de ocorrência do fluxo;

t = taxa de desconto praticada no mercado.

D.2.4 Opções

Por representarem o direito de exercer uma compra ou venda de um determinado ativo, como por exemplo uma ação, o preço de uma opção dependerá sempre do preço do seu ativo base (*underlying asset*), o que faz com que os modelos de precificação de opções sejam mais sofisticados que os utilizados por outros tipos de ativos. O modelo a ser utilizado pelo SPCRCA será o de Black & Scholes (HULL, 2006). Não cabe aqui, neste documento, demonstrar toda teoria por trás do modelo, porém, de maneira geral podemos dizer o preço de uma opção será uma função do seguinte tipo:

$$P = (T, K, S, \sigma, r) \quad (\text{D.16})$$

onde

T = tempo até o exercício da opção;

K = preço de exercício da opção;

S = preço atual do ativo de referência;

σ = volatilidade do ativo de referência

r = taxa de juro livre de risco;

D.3 Método de Análise de Risco

Sabendo que para cada tipo de ativo há um método de precificação correspondente, poderíamos fazer o mesmo para o cálculo do risco, utilizando assim um modelo específico de risco para cada ativo. Desse modo, porém, o sistema perderia flexibilidade em casos de carteiras compostas por mais de um tipo de arquivo. Para se ter uma medida mais detalhada do risco, faremos a derivada parcial do preço do ativo em relação a cada variável que compõe o preço utilizando a definição fundamental de derivada:

Através dessa definição é possível saber exatamente como o preço de um ativo se comportará em relação à variação de cada elemento que determina o preço, mais do que isso, pode-se explicar possíveis lucros ou prejuízos na performance de um ativo único ou ainda de uma cesta de ativos (*Profit & Loss Explanation*).