UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

PAULO ROBERTO LUMERTZ

Metamodelo de Interfaces do Usuário Baseado em Grafos

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Profa. Dra. Leila Ribeiro
Orientadora

Prof. Dr. Lucio Duarte
Co-orientador

Porto Alegre, novembro de 2013.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lumertz, Paulo Roberto

Metamodelo de Interfaces do Usuário Baseado em Grafos / Paulo Roberto Lumertz – 2013.

124 f.:il.

Orientador: Leila Ribeiro; Co-orientador: Lucio Duarte.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2013.

1. Interfaces do usuário. 2. Metamodelos. 3. Padrões de interfaces do usuário. I. Ribeiro, Leila, orient. II. Duarte, Lucio. coorient. III. Metamodelo de Interfaces do Usuário Baseado em Grafos.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-Chefe do Instituto de Informática: Alexsander Borges Ribeiro

AGRADECIMENTOS

Quando fazemos um trabalho com dedicação e esforço é necessário deixar outras atividades de lado e assim gostaria de agradecer a minha família, principalmente às minhas filhas, pelo tempo que não pude estar com elas, pelas brincadeiras que não pude participar e pelos jantares que não pude ir. Mas todo esforço valeu!

Não poderia deixar de agradecer ao corpo docente do Curso de Pós-Graduação em Ciência da Computação, pelas excelentes disciplinas que cursei e agradecimento em especial aos meus orientadores, prof^{a}. Dr^{a}. Leila Ribeiro e prof. Dr. Lucio Duarte, pelo apoio e pelas sugestões que muito me auxiliaram neste trabalho.

Também gostaria de agradecer a todos que, direta ou indiretamente, contribuíram para que este trabalho fosse desenvolvido.

User Interfaces Metamodel Based on Graphs

ABSTRACT

Software systems have been widely used in all business areas. These systems typically are formed by features that implement business rules and maintain the data in databases. Users interact with these systems through user interfaces, which are composed of menus where the user can browse and select the user interface that contains the desired functionality. Normally one system has a lot of these user interfaces.

The motivation of this work comes from the great difficulty to maintain information systems consistent from the point of view of user interfaces in a software production line. To ensure that the user interfaces have the same standards of appearance and behavior for each new system or maintenance on existing system require a great effort of verification and validation, which can be minimized by a process, where the structure of user interfaces is in a pattern based model. Whenever an appearance or behavior changes to a pattern it can be reapplied on all modeled systems and thus not only an improvement in productivity as well as a gain in quality.

The goal of this work is to define and to validate a metamodel that allows modeling the user interface structure for an information system. To build this metamodel was selected a graph structure. This choice was due to the ease which a user interface can be represented as a vertex and relationships by edges. Initially were identified and normalized patterns of user interfaces of a large sample of information systems. The metamodel was built based on these patterns. Using this metamodel is possible to build complete models for a hypothetical system and for three real systems and these models were used to test the metamodel, proving that it can be used in modeling the user interfaces the others similar systems.

Keywords: metamodel, graphs, graphs transformation, user interfaces, information systems, user interfaces patterns.

RESUMO

Atualmente, o uso de sistemas de informação está amplamente difundido, sendo que praticamente todas as áreas de negócio têm necessidade de tais sistemas. Estes sistemas são formados por funcionalidades que implementam regras do negócio e persistem os dados em bases de dados. Os usuários podem utilizar estes sistemas através das interfaces de usuário, que são as unidades onde estão implementadas as funcionalidades, que estão estruturadas por meio de menus. Através destes menus, o usuário pode navegar e selecionar aquela interface de usuário que contenha a funcionalidade que ele está buscando.

A motivação para este trabalho veio da grande dificuldade que é manter sistemas, em uma linha de produção de software, íntegros do ponto de vista das interfaces do usuário. A cada sistema novo ou manutenção em sistema já existente, garantir que as interfaces do usuário tenham os mesmos padrões de aparência e comportamento, exige um grande esforço de verificação e validação, o que pode ser minimizado por um processo onde a estrutura das interfaces do usuário esteja em um modelo baseado em padrões. Sempre que uma aparência ou comportamento for alterado para um padrão, ele pode ser replicado em todos os sistemas modelados, permitindo, assim, não somente uma melhoria na produtividade como também um ganho em qualidade.

O objetivo principal deste trabalho é definir e validar um metamodelo que permita modelar a estrutura destas interfaces de usuário de um sistema de informação. Para construir este metamodelo, foi escolhida uma estrutura de grafos. Esta escolha foi devido à naturalidade com que uma interface de usuário pode ser representada como um vértice e os relacionamentos por arestas. Inicialmente foram identificados e normalizados os padrões das interfaces de usuário de uma grande amostra de sistemas de informação. O metamodelo foi construído com base nestes padrões. Utilizando este metamodelo, foi possível construir modelos completos para um sistema hipotético e para três sistemas reais, comprovando que ele pode ser usado na modelagem das interfaces de usuário de outros sistemas similares.

Palavras-Chave: metamodelo, grafos, transformações de grafos, interfaces do usuário, sistemas de informação, padrões de interfaces do usuário.

LISTA DE FIGURAS

Figura 2.1: Exemplos de workbench. *20*
Figura 2.2: Exemplos de janelas. *20*
Figura 2.3: Exemplo de barra de menu de um sistema no MAC OS X (APPLE 2011). *21*
Figura 2.4: Exemplo de um formulário. *22*
Figura 2.5: Exemplo de uma lista. *22*
Figura 2.6: Exemplo de uma janela com abas. *23*
Figura 2.7: Exemplo de árvores. *23*
Figura 2.8: Exemplos de outros controles. *24*
Figura 2.9: Exemplo de grafo. *26*
Figura 2.10: Notação gráfica da gramática do exemplo. *28*
Figura 2.11: Notação gráfica da gramática do exemplo. *31*
Figura 2.12: Atributos do Workbench. *31*
Figura 2.13: Regra r1: Adição de MenuGroup no Workbench. *31*
Figura 2.14: Regra r2: Adição de MenuGroup2 ao MenuGroup1. ..... *32*
Figura 2.15: Regra r3: Adição da View1 ao MenuGroup1. ..... *32*
Figura 2.16: Exemplo de um modelo de sistema hipotético. *33*
Figura 3.1: Construção de Interfaces do Usuário com "Drag and Drop". *34*
Figura 4.1: Criação de um Padrão. *38*
Figura 4.2: Parte do Metamodelo. *43*
Figura 4.3: Relacionamento do Workbench e MenuGroup. *44*
Figura 4.4: Modelagem usando o Workbench e MenuGroup. *44*
Figura 4.5: Relacionamento MenuGroup com MenuFlag, MenuAction e ViewForm. *44*
Figura 4.6: Modelagem usando MenuGroup com MenuAction e ViewForm. *45*
Figura 4.7: Menu concreto de um sistema de informação. *45*
Figura 4.8: Uma ViewForm concreta de um sistema de informação. *46*
Figura 4.9: Modelo Classes do Sistema de Matrículas. *47*
Figura 4.10: Modelo do Sistema de Matrículas. *47*
Figura 4.11: Modelo detalhado para Disciplina. *48*
Figura 4.12: Modelo detalhado para Turma. *49*
Figura 4.13: Modelo detalhado para Matrícula. *49*
Figura 4.14: Menus de um sistema de informação MDI. *50*
Figura 4.15: Modelo do sistema GUI. *51*
Figura 4.16: Uma UI do Sistema Books Online. *52*
Figura 4.17: Modelo do sistema WUI. *52*
Figura 4.18: Menu do sistema no celular. *53*
Figura 4.19: Opção contagem, seleção do setor e do lote. *53*
Figura 4.20: Opção contagem, informando cabeças, peso e estado. *54*
Figura 4.21: Opção potreiros, lançando informações. *54*
Figura 4.22: Modelo do sistema Caravana Mobile. *55*
Figura 5.1: Itens de Menu com grau 10. ..... *58*
Figura 5.2: Modelagem de Menu com profundidade 3. ..... *59*
Figura 5.3: Grafo do modelo de um sistema GUI. *61*
Figura 5.4: Grafo com densidade do modelo de um sistema GUI. *62*
Figura 5.5: Regra r1: transformação de List em Table. *63*
Figura 5.6: Aplicação dá regra r1: ViewListForm em ViewTableForm. *63*
Figura 5.7: Detalhamento da Regra. *63*

Figura A.1: Exemplo de Workbench com Painéis Móveis. *75*
Figura A.2: Exemplo de Workbench com MDI. *75*
Figura A.3: Exemplo de Workbench com Web. *76*
Figura A.4: Áreas de um Workbench. *76*
Figura A.5: Exemplo de View com filtro e lista. *77*
Figura A.6: Exemplo de View com formulário e lista. *78*
Figura A.7: Outro exemplo de View com formulário e lista. *78*
Figura A.8: Exemplo de View com diversos formulários e listas. *79*
Figura A.9: Exemplo de View com árvore e lista. *79*
Figura A.10: Exemplo de View com referência cruzada. *80*
Figura A.11: Exemplo de Menu de um sistema. *80*
Figura B.1: Exemplo de Title Bar. *81*
Figura B.2: Exemplo de Menu Bar. *82*
Figura B.3: Exemplo de Tool Bar. *83*
Figura B.4: Exemplo de Work Area de Workbench MDI. *83*
Figura B.5: Exemplo de Work Area de Workbench Movable Panels. *84*
Figura B.6: Exemplo de Status Bar de Workbench Movable Panels. *84*
Figura B.7: Exemplos de Single Views. *85*
Figura B.8: Exemplos de Composite View. *86*
Figura B.9: Exemplo de Multi Composite View. *86*
Figura B.10: Exemplos de uma Crosstab View. *87*
Figura B.11: Exemplo de um Form. *89*
Figura B.12: Exemplo de um List. *89*
Figura B.13: Exemplo de um Tree. *90*
Figura B.14: Exemplo de um Tab. *90*
Figura B.15: Exemplo de um Filter. *91*
Figura D.1: Metamodelo Completo. *115*
Figura D.2: Metamodelo One View Graph. *116*
Figura D.3: Metamodelo Double Views Graph. *117*
Figura D.4: Metamodelo List Master Graph. *118*
Figura D.5: Metamodelo Tree Master Graph. *119*
Figura D.6: Metamodelo Master Multi Graph. *120*
Figura D.7: Metamodelo Crosstab Graph. *121*
Figura D.8: Metamodelo Primitive Graph. *122*
Figura D.9: Metamodelo Primitive Tab Graph. *122*
Figura D.10: Metamodelo Attributes Graph. *123*
Figura D.11: Metamodelo Attribute Actions Graph. *124*

LISTA DE TABELAS

Tabela 2.1: Regras de Criação *32*
Tabela 4.1: Elementos primitivos *39*
Tabela 4.2: Workbench e Estrutura de Menu *40*
Tabela 4.3: Single Views *40*
Tabela 4.4: Composite Views *41*
Tabela 4.5: Multi Composite Views *41*
Tabela 4.6: Crosstab Views *42*
Tabela 5.1: Estatística das UIs do Sistema GUI *60*
Tabela 6.1: Comparação com trabalhos relacionados *66*
Tabela A.1: Sistemas Web Analisados. *74*
Tabela A.2: Sistemas Desktop Analisados. *74*

SUMÁRIO

ABSTRACT 4

RESUMO 5

LISTA DE FIGURAS 6

LISTA DE TABELAS 8

1 INTRODUÇÃO 13

1.1 Objetivo 15
1.1.1 Objetivos Específicos 15
1.1.2 Contribuições 15

1.2 Estrutura do Trabalho 16

2 CONCEITOS TEÓRICOS 17

2.1 Conceitos de Interfaces de Usuário 17
2.1.1 Evolução das Interfaces do Usuário 17
2.1.2 Características de uma boa Interface do Usuário 17
2.1.3 Estrutura das UIs de um sistema de informação 19
2.1.3.1 Janelas 20
2.1.3.2 Menus 20
2.1.4 Controles da interface do usuário 21
2.1.4.1 Controles de Texto 21
2.1.4.2 Controle de Listas 22
2.1.4.3 Controles de Abas 22
2.1.4.4 Controles de Árvores 23
2.1.4.5 Outros controles 23
2.1.5 Comportamentos padronizados da interface do usuário 24
2.1.5.1 CRUD 24
2.1.5.2 Formulários 24
2.1.5.3 Lista e formulário 24
2.1.5.4 Mestre e Detalhe 25

2.2 Conceitos de Grafos 25
2.2.1 Introdução 25
2.2.2 Conceitos fundamentais 25
2.2.3 Grafos com atributos e transformações de grafos 27

2.2.4 Exemplo de Grafos com atributos 30

3 TRABALHOS RELACIONADOS 34

3.1 Construção de Interfaces de Usuário com "Drag and Drop" 34

3.2 Especificação de Interfaces de Usuário por Modelos UML 35

3.3 Abordagem baseada em metamodelo e padrões 36

3.4 Interfaces do Usuário com Transformações de Grafos 36

4 PROPOSTA E VALIDAÇÃO DO METAMODELO 38

4.1 Padrões de UIs identificados 38

4.2 Definição do Metamodelo 43
4.2.1 A Estrutura do Metamodelo 46

4.3 Validação do Metamodelo 46
4.3.1 Exemplo de um sistema hipotético 46
4.3.2 Exemplo de um sistema Desktop 49
4.3.3 Exemplo de um sistema Web 51
4.3.4 Exemplo de um sistema Mobile 53

5 ANÁLISES E EVOLUÇÃO DAS UIS DOS SISTEMAS 56

5.1 Ferramentas de análise de grafos 56

5.2 Ferramentas de análise de gramáticas de grafos 56

5.3 Melhorias do modelo 58

5.4 Consistência do modelo 59

5.5 Análises sobre os modelos 60

5.6 Evolução do metamodelo 62

6 CONCLUSÕES 65

6.1 Contribuições 65

6.2 Comparação com os trabalhos relacionados 65

6.3 Trabalhos futuros 66

REFERÊNCIAS 68

APÊNDICE A ANÁLISE DAS UIS DE SISTEMAS DE SOFTWARE 73

Sistemas Analisados 73

Bancada de Trabalho 74

Views dos Sistemas **77**

Menus e Tool Bar **80**

APÊNDICE B NORMALIZAÇÃO DOS PADRÕES 81

Workbench **81**
Title Bar 81
Menu Bar 82
Tool Bar 83
Work Area 83
Status Bar 84

Views **84**
Single Views 84
Composite Views 85
Multi Composite Views 86
Crosstab Views 87

Componentes básicos de interfaces do usuário **87**
Actions 87
Inserir (Insert) 87
Editar (Edit) 88
Deletar (Delete) 88
Salvar (Save) 88
Atualizar (Refresh) 88
Sair (Exit) 88
Copiar (Copy) 88
Cortar (Cut) 88
Colar (Paste) 88
Form 88
List 89
Tree 89
Tab 90
Filter 90

APÊNDICE C PADRÕES DE ELEMENTOS DE UI 92

Padrões Primitivos (Primitive Patterns) **92**
Actions Patterns 92
View Patterns 94
Search Patterns 95

Padrões Básicos (Basic Patterns) **96**
Padrões de Workbench (Workbench Patterns) 96
Padrões de Menu (Menu Patterns) 97
Single View Patterns 98
Composite View Patterns 102
View List Master Detail 103
View Tree Master Detail 105
Multi Composite View Patterns 108
View List Master Multi Detail 108
View Tree Master Multi Detail 109
Crosstab View Pattern 111

APÊNDICE D METAMODELO COMPLETO 114

Metamodelo com uma View (One View Graph) 116

Metamodelo com duas Views (Double View Graph) 117

Metamodelo com Composite List Master (List Master Graph) 118

Metamodelo com Composite Tree Master (Treet Master Graph) 119

Metamodelo com Multi Composite (Master Multi Graph) 120

Metamodelo Crosstab (Crosstab Graph) 121

Metamodelo Elementos Primitivos (Primitive Graph) 122

Metamodelo Elementos Primitivos Tab (Primitive Tab Graph) 122

Metamodelo com Atributos (Attributes Graph) 122

Metamodelo com Atributos de ações (Attribute Actions Graph) 124

1 INTRODUÇÃO

Hoje em dia, os sistemas de informação são largamente utilizados por qualquer tipo de empresa e nas mais diversas áreas de negócio e os usuários interagem através das interfaces de usuário (em inglês UI - *User Interface*). Estas UIs são as unidades operacionais do sistema e é onde estão implementadas as funcionalidades disponíveis para o usuário. Normalmente o usuário navega por *menus* para selecionar uma funcionalidade desejada. Estes sistemas podem ser desenvolvidos para os mais diversos ambientes, como *desktop*, *web* e dispositivos móveis e também podem ter diferentes padrões de UIs. As áreas de negócio dos clientes, onde são utilizados estes sistemas, estão em continua evolução, sendo necessário que estes sistemas sejam continuamente evoluídos.

Desenvolver e manter estas UIs, em um sistema de informação, normalmente não é uma tarefa fácil e muito esforço deve ser dedicado. As empresas que desenvolvem sistemas de informação para automatizar o negócio de seus clientes têm diversos desafios, como: a) manter estes sistemas atualizados tecnologicamente; b) manter as interfaces do usuário consistentes em aparência e comportamento durante seu desenvolvimento e nas manutenções futuras, pois esta consistência facilita a operação do sistema pelos usuários; c) manter o sistema consistente, mesmo com a troca de recursos humanos que trabalham no seu desenvolvimento. Uma troca de tecnologia, como mudar a linguagem de desenvolvimento, implica reescrever grande quantidade de código e deve ser garantido que as funcionalidades continuem ativas e de acordo com os requisitos originais. Para garantir que as UIs tenham a mesma aparência e comportamento, mesmo na ocorrência de alterações, pode ser necessário um grande esforço de revisão. A troca de pessoal técnico é outro problema, pois normalmente o conhecimento dos sistemas pode não estar bem documentado, ou se documentado, pode não estar atualizado. Uma pesquisa conduzida por Myers e Rosson [MYERS, 1992] concluiu que o esforço na construção das UIs pode chegar a quase 50% do esforço total da construção de um sistema. Também tem sido percebido que o projeto e a implementação das UIs raramente têm o mesmo nível da especificação e modelagem que os dados e funcionalidades [PASTOR, 2006], mesmo sendo a construção das UIs reconhecidas como uma das etapas mais demoradas de qualquer processo de produção de *software*. Ainda nos dias de hoje esta é uma realidade presente e propostas para solucionar os problemas citados não são novas.

Desde a década de 90 muitas pesquisas já foram desenvolvidas visando aumentar a facilidade de criação das UIs. Inicialmente as pesquisas na linha de *Model Based User Interface Development* (MBUID) [SCHLINGBAUM, 1997], onde diversos modelos foram criados com o objetivo de gerar as UIs [SCHLINGBAUM, 1996]. Muitas ferramentas foram criadas com esta finalidade e Myers [MYERS, 1995] introduziu uma classificação para estas ferramentas. Iniciativas de estender a UML com elementos para

a modelagem de UIs também foram pesquisadas, criando a UMLi [DA SILVA, 2003]. Nos últimos anos, a comunidade científica tem pesquisado a *Model Driven Engineering* (MDE) [SCHMIDT, 2006] e também o uso de padrões e componentes reutilizáveis [PETRASCH, 2010]. A utilização de padrões tem o objetivo de facilitar o reuso de *software*, estimulando a especificação de UIs com estes padrões. Assim, são utilizadas sintaxes abstratas de UIs para guiar a fase de projeto, com a geração automática de código [MOLINA, 2002]. A MDE aparece com uma alternativa para auxiliar na diminuição do esforço na construção de UIs, possibilitando a geração automática destas UIs por transformação de modelos, mas para utilizar seus conceitos, existe a necessidade de um metamodelo. Um metamodelo define o nível mais abstrato de uma hierarquia de modelos, onde os modelos de um determinado nível são sempre mais abstratos que os do nível inferior e mais concretos do que os dos níveis superiores a eles. Desta forma, o metamodelo define os elementos básicos a serem utilizados nos demais modelos, os quais podem refinar a definição atribuindo significados mais concretos a estes elementos dentro de um contexto específico. Para aumentar o reuso a premissa é ter um conjunto de padrões reutilizáveis.

Também é importante referenciar que o poder computacional tem crescido muito nos últimos anos, tanto para computadores *desktop* como para dispositivos móveis. Isto possibilita novas formas de interação e assim as ferramentas para construção de UIs devem facilitar a construção de UIs também para estas novas formas de interação. Nos dias atuais, de acordo com Olsen & Scott [OLSEN, 2005], ainda existe uma grande lacuna no projeto de UIs e também, o trabalho de construção de UI requer alto nível de conhecimento técnico. No processo de modelagem de um sistema de informação tem sido dedicado maior esforço na definição das propriedades do sistema e poucas técnicas são oferecidas para especificar a estrutura e a semântica das UIs. Utilizar padrões conceituais para as UIs é uma alternativa [PETRASCH, 2010] [PASTOR, 2007].

Em complemento as abordagens de pesquisa referenciadas anteriormente, este trabalho pesquisa uma estratégia de solução para os desafios identificados, definindo um metamodelo a partir de padrões de UI. Este metamodelo permite a criação de modelos para as UIs de sistemas que pertençam ao nicho de sistemas de informação para automação de negócios.

A estratégia de solução partiu do estudo de padrões de UI e a normalização destes padrões, chegando a um conjunto que representa os sistemas analisados. Com estes padrões foi criado um metamodelo para facilitar a modelagem das UIs de sistemas de informação. O metamodelo foi construído embasado na teoria dos grafos, pois as UIs de um sistema podem facilmente serem representadas por vértices e seus relacionamentos por arestas. Além disso, a teoria dos grafos tem um formalismo matemático que pode ser aplicado para as transformações de modelos. Esta mesma abordagem já foi utilizada em outros trabalhos [BIERMANN, 2008]. Assim, os elementos de UIs podem ser facilmente representados como elementos de um grafo, o que torna este formalismo apropriado para modelar UIs. Usando grafos podemos aplicar transformações de grafos [EHRIG, 2006] e para isto existe uma vasta teoria que pode ser aplicada nas transformações de modelo. É importante referenciar que o uso de grafos, além de facilitar o entendimento da estrutura, por ser uma construção visual, possibilita um formalismo matemático para as transformações e análises dos modelos.

A solução pesquisada gerou um metamodelo que possibilita modelar as UIs de sistemas de informação e foi validado em sistemas reais. Os modelos são construídos a partir de componentes reusáveis e modelados, de forma incremental, em uma

ferramenta de modelagem. Neste trabalho é utilizada a ferramenta *Enterprise Architect* [EA, 2012]. Um metamodelo permite validar e analisar os modelos e também facilita a evolução dos próprios modelos, pois podem ser definidas regras de transformação no metamodelo com objetivo de contemplar as evoluções que se tornem necessárias. Não é objetivo deste trabalho a etapa de geração de código, mas se forem geradas, estas UIs podem ter uma melhoria na usabilidade, pois dão uma maior garantia na consistência das experiências anteriores do usuário [MYERS, 2007]. Este trabalho contempla a criação do metamodelo possibilitando a modelagem da estrutura das UIs para o nicho de sistemas de informação de automação de negócios. A criação do metamodelo teve por base padrões das interfaces de usuário obtidos através da análise de um conjunto de 19 sistemas de informação reais e em uso em empresas. Estes padrões foram identificados e validados com padrões de UI aceitos pela comunidade científica [TIDWELL, 2011] [GALITZ, 2007]. Após a criação do metamodelo, ele foi aplicado em exemplos reais de sistemas com o objetivo de validar a construção de modelos. Com o metamodelo proposto é possível modelar as UIs de sistemas de informação, o que é feito com bastante simplicidade utilizando uma ferramenta de modelagem. O modelo criado possibilita uma visualização completa da estrutura do sistema, inclusive fornecendo uma ideia bastante clara do tamanho e da complexidade das UIs do sistema. Por fim, é ilustrado como é possível aplicar regras de transformação nos modelos, usando grafos, para contemplar a evolução do metamodelo.

1.1 Objetivo

O objetivo deste trabalho é definir e validar um metamodelo que permita modelar a estrutura das interfaces de usuário de um sistema de informação a partir de padrões de UIs identificados pela análise de sistemas de informação.

1.1.1 Objetivos Específicos

Este trabalho tem os seguintes objetivos específicos:

- Analisar um conjunto de sistemas de informação para identificar aparências e comportamentos;
- Normalizar as UIs analisadas para definir padrões que possam ser reusáveis e criar uma representação gráfica da sintaxe abstrata;
- Definir um metamodelo que permita modelar integralmente a estrutura das interfaces do usuário de um sistema de informação que venha a ser desenvolvido, ou de um existente por engenharia reversa das UIs;
- Validar a aplicação do metamodelo na modelagem destes sistemas de informação;
- Ilustrar que o metamodelo é evolutivo, ou seja, suporta novas definições e elementos e através de regras de transformações de grafos os modelos podem ser evoluídos.

1.1.2 Contribuições

O trabalho mostra como um metamodelo, baseado em grafos, pode ser utilizado como uma ferramenta de engenharia de *software* na modelagem de interfaces de usuário de um sistema de informação. Os padrões identificados e normalizados são de uso

genérico em tais sistemas, e podem ser utilizados como fonte de consulta para outros trabalhos que necessitem utilizar interfaces de usuário em sistemas de informação.

1.2 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma: No Capítulo 2 são apresentados os conceitos teóricos de interfaces do usuário e de grafos necessários ao entendimento do trabalho; no Capítulo 3 são apresentados os trabalhos relacionados; no Capítulo 4 é apresentado como foi criado o metamodelo e sua validação; no Capítulo 5 são identificadas ferramentas que podem ser utilizadas para analise de grafos e são ilustradas as análises que podem ser feitas sobre um sistema modelado dessa forma, e como o modelo pode sofrer evolução. No Capítulo 6 é feito um comparativo com os trabalhos relacionados e por fim é apresentada a conclusão e a sugestão de trabalhos futuros que podem ser desenvolvidos partindo deste trabalho.

2 CONCEITOS TEÓRICOS

Para este trabalho, uma base teórica sobre interfaces de usuário e teoria de grafos é necessária e aqui são apresentados os conceitos que são importantes para o entendimento do trabalho.

2.1 Conceitos de Interfaces de Usuário

Uma interface de usuário, aqui chamada do inglês de "UI - *User Interface*" é o meio em que um usuário opera um sistema de informação. Uma boa *interface* fornece uma experiência "amigável", permitindo que o usuário interaja com o *software* de uma forma natural e intuitiva. As interfaces do usuário de um sistema de informação permitem o diálogo do usuário com o sistema executando duas funções básicas: a entrada de dados e a apresentação de informações. Para a entrada de dados, a interação pode ser feita por meio de diversos dispositivos como teclado, mouse, telas sensíveis ao toque, entre outros. A apresentação dos dados normalmente é feita através de um monitor.

2.1.1 Evolução das Interfaces do Usuário

Nos primeiros sistemas de informação, as interfaces do usuário eram textuais. Hoje, elas são gráficas e com grande resolução, já havendo, inclusive, interfaces do usuário em multimídia. O grande avanço das interfaces do usuário aconteceu com o avanço da tecnologia que permitiu as *interfaces* gráficas, o que também teve um peso maior no processo de desenvolvimento do *software*, pois a construção dessas *interfaces* começou a consumir mais recursos de desenvolvimento. As primeiras *interfaces* gráficas foram criadas pela *Xerox Corporation* no *Palo Alto Research Center* até chegar às *interfaces* atuais, como *Windows 8* da *Microsoft* [WINDOWS8, 2014], *OS X* da *Apple* [APPLE-OSX, 2014] e também as *interfaces* para dispositivos móveis como *IOS* [APPLE-IOS, 2014], *Android* [ANDROID, 2014] e outros. Todas essas interfaces de usuário, por mais que tenham evoluído, ainda tem características comuns já identificadas em *Palo Alto*, como o conceito de janelas, ícones, *menus*, ponteiros e outros componentes de *interface*.

2.1.2 Características de uma boa Interface do Usuário

Desenvolver uma boa *interface* de usuário (UI) não é uma tarefa fácil, o grupo de usuários que pode vir a utilizar as *interfaces* do sistema pode ser bastante heterogêneo, com maior ou menor grau de conhecimento de uso, o que pode dificultar o projeto da UI. Existem algumas características padronizadas [TIDWELL, 2011] que podem ajudar quando do projeto da UI e estão descritas a seguir as características que este trabalho garante por construção:

Exploração segura: A UI permite que o usuário sinta que ele pode usar a interface e não sofre consequências, ou seja, a UI permite que o usuário explore um sistema com o qual ainda não esteja familiarizado e possa voltar atrás, sem erros ou problemas no sistema.

Gratificação instantânea: Os usuários gostam de ver os resultados imediatamente a partir de suas ações, isso é da natureza humana. Se um usuário inicia um sistema e obtém uma experiência de sucesso isto é muito gratificante.

Satisfatório o suficiente (*good enough*): A interface não deve exigir do usuário mais do que ele necessita naquele exato momento e não deve fazê-lo pensar mais do que o necessário.

Mudanças de percurso: É bastante normal os usuários mudarem de objetivo durante o uso de um sistema, ou seja, vão utilizar uma interface e resolvem usar outra. Uma boa interface deve propiciar as mudanças de percurso naturalmente.

Escolhas adiadas: Uma boa interface deve prever que o usuário possa responder posteriormente questões para obter o resultado desejado, ou seja, deve-se questionar o mínimo essencial.

Construção incremental: Permitir mudanças, pois normalmente o usuário não faz tudo de uma vez.

Hábito: Quando um usuário usa uma interface repetidamente, algumas ações tornam-se reflexivas (ele utiliza por puro reflexo) e o usuário não precisa pensar sobre estas ações, elas se tornam habituais. Uma boa interface deve explorar esta habitualidade, pois isto torna o usuário mais eficiente no uso da interface. Deve-se também adotar os hábitos de outros sistemas que são bastante conhecidos e usados.

Memória espacial: Quando os usuários manipulam as *interfaces*, eles normalmente encontram as opções lembrando onde elas estão e não quais são os nomes. Assim, favorecer a memória espacial é muito efetivo na construção de *interfaces*.

Uso de teclado somente: Alguns usuários são mais eficientes usando somente o teclado, então a interface deve prover esta capacidade também. Isto normalmente é feito por teclas de atalho e mnemônicos.

Outras características de uma interface de usuário não supridas por este trabalho, mas que poderiam ser implementas posteriormente em um sistema de informação, caso seja necessário:

Comentários de outras pessoas: Como as pessoas são sociais, a opinião de outros usuários pode influenciar no uso, então é importante que a UI suporte comentários do usuário e que isto fique disponível para todos os usuários.

Memória prospectiva: É o fato de criar algum mecanismo para lembrar-se de uma atividade futura, o que pode facilmente ser adicionado na interface do usuário.

Repetição simplificada: Muitas vezes o usuário necessita fazer uma determinada atividade diversas vezes, neste caso, o ideal é que a interface tenha mecanismo de facilitar esta repetição. Um exemplo é o caso de substituição de texto, onde temos a opção de substituir tudo, para que o usuário não necessita repetir um a um.

Na construção de uma boa interface do usuário, existem princípios que devem ser considerados [JOHNSON, 2010]. Estes princípios podem incrementar o sucesso no projeto da UI e normalmente são usados combinados. Um exemplo onde praticamente

todos são utilizados é na *interface* do *Mac OS X* [APPLE, 2011]. No projeto dos padrões de UI deste trabalho estes princípios foram considerados e abaixo é apresentada uma descrição destes princípios:

Proximidade: A distância entre os objetos pode afetar a percepção de como os objetos estão organizados na UI.

Similaridade: Procurar agrupar elementos similares ajuda incrementar a percepção e entendimento da UI.

Continuidade: Em adição aos dois princípios anteriores, o princípio da continuidade incrementa a percepção visual melhor do que em segmentos descontínuos.

Encerramento: O sistema visual humano tem tendência a ver um objeto como completo, mesmo que ele tenha uma parte em branco, imagine um círculo com alguns trechos faltantes, mesmo assim a visão tende a ver como um círculo.

Simetria: Quando é utilizada a simetria a percepção de complexidade diminui, então é bem útil utilizar este princípio no projeto da UI.

Frente e fundo: O sistema visual tem facilidade em separar a frente e o fundo e isto deve ser explorado no projeto da UI para identificar elementos que necessitem maior atenção.

Destino comum: Quando existe um grupo de elementos e eles se movimentam juntos reforça a percepção de grupo.

2.1.3 Estrutura das UIs de um sistema de informação

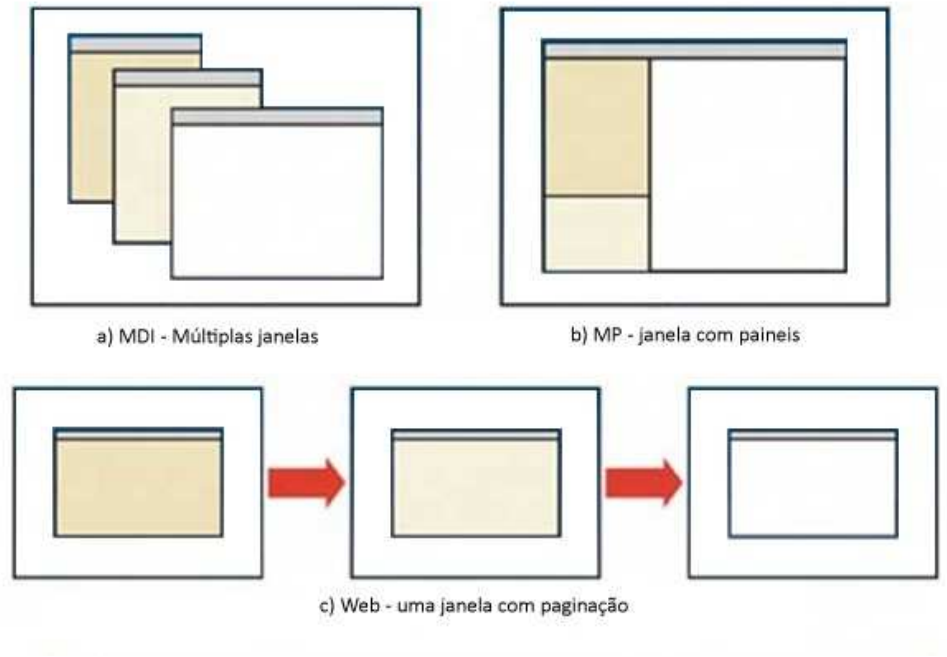
Um sistema de informação, do ponto de vista das interfaces do usuário, é composto por diversos elementos. Inicialmente temos a janela principal do sistema, conhecida como *workbench*. Um *workbench* contém os *menus*, barras de ferramentas, áreas de mensagens e uma área onde as janelas específicas das funcionalidades são apresentadas. A classificação de um *workbench* é feita com base na forma como as janelas são apresentadas:

- *Multi Document Interface* (MDI): as janelas são apresentadas umas sobre as outras e podem ter várias abertas ao mesmo tempo.
- *Movable Panels* (MP): as janelas são apresentadas como painéis lado a lado.
- *Web*: as janelas são apresentadas uma a uma e pode haver uma paginação quando houver necessidade de sequenciamento.

Um sistema pode ser desenvolvido com as UIs em múltiplas janelas ou janelas com painéis. A decisão depende do tipo de aplicação e onde ela vai ser executada e também da decisão do projetista, pois em um sistema de informação o usuário pode ficar confuso com múltiplas janelas, outras vezes, o usuário pode realmente necessitar de múltiplas janelas para poder ver informações em paralelo. Aplicações *web* trabalham melhor com o modelo de paginação com uma única janela, onde uma página é mostrada a cada vez. Esta também será a melhor escolha para dispositivos móveis pela limitação do tamanho da tela. Uma janela com painéis é uma boa escolha, desde que o número de painéis não seja muito grande, mas isto para aplicações que sejam executadas em *desktop*.

A Figura 2.1 apresenta os tipos de *workbench*.

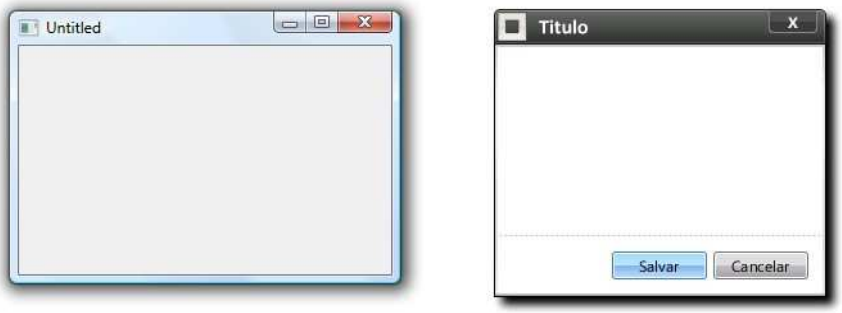
Figura 2.1: Exemplos de *workbench*.



2.1.3.1 Janelas

As janelas são os elementos da UI onde as funcionalidades de um sistema são implementadas. Normalmente elas são compostas de diversos outros elementos, como será visto mais à frente. Uma janela normalmente tem um título e controles de fechamento e ajuste de tamanho e opcionalmente pode ter um ícone associado a ela. Adicionalmente uma janela pode ser definida como *modal*, o que significa que o usuário não pode abrir outra janela sem finalizar a operação corrente nesta janela *modal*. A Figura 2.2 apresenta exemplos típicos de janelas de um sistema de informação.

Figura 2.2: Exemplos de janelas.



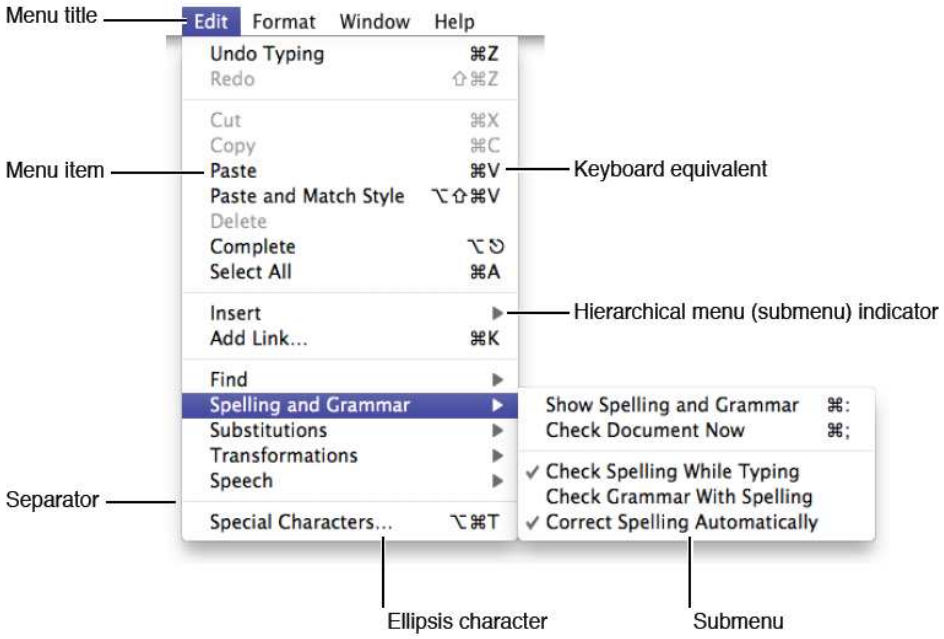
2.1.3.2 Menus

Os *menus* apresentam listas de itens que são comandos, atributos ou estados a partir dos quais o usuário pode fazer escolhas. Os usuários utilizam os *menus* frequentemente, especialmente quando eles estão procurando por funcionalidades na interface. Garantir

que os *menus* estejam organizados corretamente e classificados claramente é crucial para garantir a habilidade do usuário em explorar facilmente as funcionalidades do sistema.

Normalmente um *menu* pode ser uma barra de *menu* ou um *menu* contextual, que pode ser acionado em um determinado contexto. A barra de *menu* normalmente é uma estrutura hierárquica composta de vários níveis de itens, como o exemplo apresentado na Figura 2.3. Um *menu* de contexto normalmente tem um único nível com os itens que se aplicam ao contexto.

Figura 2.3: Exemplo de barra de *menu* de um sistema no MAC OS X (APPLE 2011).



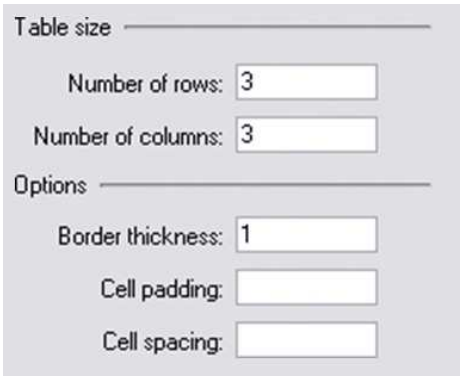
2.1.4 Controles da interface do usuário

Uma interface do usuário de um sistema de informação é formada por um conjunto de controles. Estes controles são mecanismos de apresentação da informação ou interação com o usuário ou até ambos. Eles são elementos que podem ser utilizados para a construção da UI, sendo colocados dentro de uma janela.

2.1.4.1 Controles de Texto

Os controles de texto mostram texto ou aceitam texto com entrada como apresentado na Figura 2.4. Um campo de texto estático é utilizado para mostrar informações e não pode ser modificado pelo usuário. Um campo de entrada de texto pode ser modificado pelo usuário. Com estes controles podem ser criados formulários para apresentar ou editar dados. Na edição podem eles ser utilizados para inserir ou atualizar registros de informações.

Figura 2.4: Exemplo de um formulário.



2.1.4.2 Controle de Listas

O controle de lista permite criar listas com uma ou mais colunas. Listas são muito utilizados em sistemas de informação. Recursos de classificação da coluna normalmente já estão disponíveis nestes controles. Os campos podem ser definidos para serem editáveis ou não. Na Figura 2.5 é apresentado um exemplo de uma lista.

Figura 2.5: Exemplo de uma lista.

▲	Song Name	Time	Artist	Album
3	Silver Thunderbird	4:38	Marc Cohn	Marc Cohn
4	Dig Down Deep	5:08	Marc Cohn	Marc Cohn
5	Walk on Water	4:01	Marc Cohn	Marc Cohn
6	Miles Away	3:23	Marc Cohn	Marc Cohn
7	Saving the Best for Last	5:35	Marc Cohn	Marc Cohn
8	Strangers in a Car	2:47	Marc Cohn	Marc Cohn
9	29 Ways	3:06	Marc Cohn	Marc Cohn
10	Perfect Love	4:23	Marc Cohn	Marc Cohn
11	True Companion	4:10	Marc Cohn	Marc Cohn

2.1.4.3 Controles de Abas

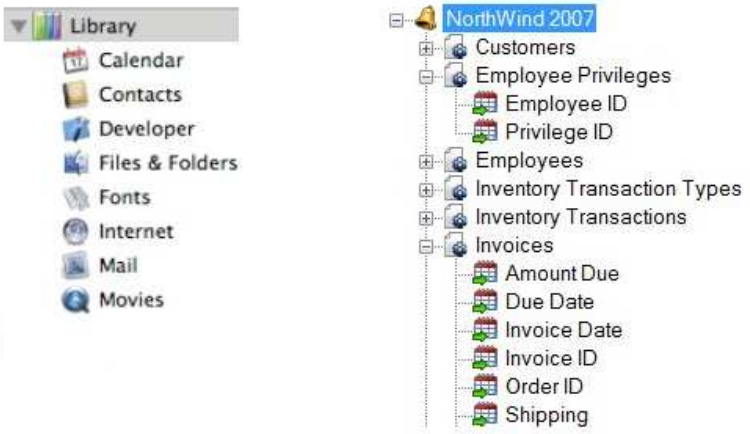
Os controles de abas são utilizados para ter duas ou mais abas dentro de uma janela. Dentro de uma aba podem ser colocados outros controles. As abas são utilizadas para separar grupos de informações. Na Figura 2.6 é apresentado um exemplo de abas.

Figura 2.6: Exemplo de uma janela com abas.

2.1.4.4 Controles de Árvores

Os controles de árvores são utilizados para mostrar dados estruturados de forma hierárquica. Uma árvore pode ter vários níveis. Para cada nível pode ser atribuído um ícone. Na Figura 2.7 estão apresentados exemplos de árvores.

Figura 2.7: Exemplo de árvores.



2.1.4.5 Outros controles

Existem outros controles para a construção de uma interface do usuário, como botões, controles de seleção como *radio buttons* e *check box* entre outros, mas para efeitos deste trabalho eles serão utilizados dentro dos padrões de UI.

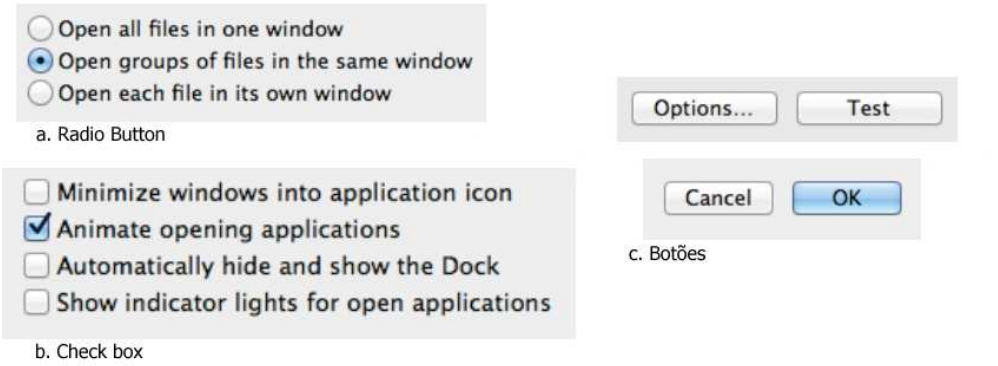
Radio Button: utilizado para selecionar um item dentre vários, pois a seleção é mutuamente exclusiva.

Check box: utilizado para selecionar mais de um item dentre vários. Ele descreve um estado, ação ou valor que pode ser ligado ou desligado.

Botões: são utilizados para iniciar uma ação e existem diversos tipos de botões. Os botões podem conter ícones ou textos.

Na Figura 2.8 são apresentados exemplos destes controles.

Figura 2.8: Exemplos de outros controles.



2.1.5 Comportamentos padronizados da interface do usuário

Em sistemas de informação existem diversos comportamentos que são usados para manter os dados. Estes comportamentos operam sobre dados de uma mesma classe de dados que são persistidos em uma tabela na base de dados ou sobre classes de dados relacionadas.

2.1.5.1CRUD

É acrônimo de *Create*, *Read*, *Update* e *Delete*, usado para as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface de usuários para criação, consulta, atualização e exclusão de dados. As operações CRUD são relevantes em interfaces de usuários de vários níveis. Por exemplo, em um catálogo de endereços, uma entrada de um contato individual pode ser considerada a unidade básica de persistência. As operações mínimas são o *Create*, para criar ou adicionar novas entradas, o *Read* para ler, recuperar ou ver entradas existentes, o *Update* para atualizar ou editar entradas existentes e o *Delete* para remover entradas existentes.

2.1.5.2 Formulários

Os formulários normalmente estão dentro de uma janela e são formados por diversos controles que permitem exibir e alterar dados de um registro. Neste formulário normalmente estão associadas às operações de CRUD.

2.1.5.3 Lista e formulário

Neste caso existe adicionalmente uma lista que permite selecionar uma entrada e a partir dela os dados são apresentados em um formulário. Normalmente tanto a lista quanto o formulário estão dentro da mesma janela e associados às operações de CRUD. O importante neste caso é que os dados da lista como do formulário pertençam à mesma origem ou tabela. Um exemplo seria uma lista de pessoas e um formulário para apresentar os dados cadastrais da pessoa que estiver selecionada.

2.1.5.4 Mestre e Detalhe

O mestre e detalhe é um comportamento muito comum em sistemas de informação. O exemplo clássico é de um pedido de compra e seus itens, onde o pedido é considerado o mestre e os itens o detalhe. Veja que existe a necessidade de um relacionamento entre o pedido e os itens e normalmente eles estão em origens ou tabelas diferentes. Neste caso as operações de CRUD são aplicadas tanto ao mestre quanto ao detalhe. É importante salientar que para um mestre pode haver vários detalhes.

2.2 Conceitos de Grafos

2.2.1 Introdução

Muitas situações do mundo real podem ser descritas por pontos e linhas unindo pares destes pontos. Por exemplo, os pontos podem representar pessoas e as linhas identificando pares de amigos; ou os pontos podem representar centros de comunicação e as linhas representando as ligações de comunicação. Note que o interesse é se dois pontos estão ligados por uma linha. Isso nada mais é que a representação de grafos. Os grafos fornecem uma abordagem simples e poderosa para uma variedade de problemas que são típicos da ciência da computação em geral e da engenharia de *software* em particular [HECKEL, 2006]. As UIs de um sistema podem facilmente serem representadas por vértices e seus relacionamentos por arestas o que torna a escolha de uma estrutura de grafos uma forma natural de representar estes tipos de elementos.

A teoria dos grafos vem sendo estudada desde o século XVI e o artigo de Leonhard Euler, publicado em 1736 [EULER, 1736], sobre o problema das sete pontes de Königsberg, é considerado o primeiro resultado da teoria dos grafos. Desde então muitos estudos sobre grafos já foram feitos e hoje existe um grande ferramental matemático sobre grafos, o que torna uma excelente abordagem para construir um metamodelo com o formalismo necessário para este trabalho.

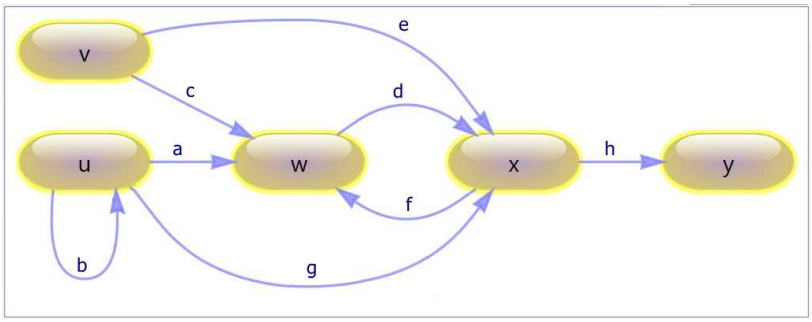
2.2.2 Conceitos fundamentais

Grafos podem ser representados graficamente onde cada vértice é indicado por um ponto e cada aresta por uma linha e dessa forma facilita o entendimento de suas propriedades. Não existe um caminho correto para desenhar o grafo, a posição relativa dos pontos representando os vértices e a forma das linhas representando as arestas usualmente não tem significado. O diagrama simplesmente mostra o relacionamento entre vértices e arestas.

Muitas das definições e conceitos de grafos na teoria dos grafos são sugeridas por sua representação gráfica. Quando existir uma orientação em cada aresta é dito que o grafo é um **grafo orientado** e um vértice é considerado **origem** (*source*) e o outro o seu **destino** (*target*). Neste trabalho são utilizados grafos orientados.

Um grafo orientado G é uma tupla (vertG, edgeG, sourceG, targetG), onde vertG é um conjunto de vértices, edgeG é um conjunto de arestas e sourceG e targetG: edgeG → vertG são funções totais, definindo a fonte e destino de cada aresta, respectivamente [BONDY, 2006] [DIESTEL, 2000].

Figura 2.9: Exemplo de grafo.



Seja G = (vertG, edgeG, sourceG, targetG), cuja a representação gráfica está na Figura 2.9, onde

vertG = {u, v, w, x, y}

edgeG = {a, b, c, d, e, f, g, h}

sourceG = { a→u, b→u, c→v, d→w, e→v, f→x, g→u, h→x }

targetG = { a→w, b→u, c→w, d→x, e→x, f→w, g→x, h→y }

Uma aresta é **incidente** (*incident*) ao vértice quando ela se liga a ele. Dois vértices que tem a mesma aresta incidente são ditos **adjacentes** (*adjacent*). Se duas arestas são incidentes a um vértice comum, os dois vértices adjacentes são **vizinhos** (*neighbours*). O conjunto de vizinhos de um grafo G é denotado por $N_G(V)$. Uma aresta com extremidades idênticas é chamada de **laço** (*loop*) e com extremidades distintas é chamada de **link**. Dois ou mais links sobre o mesmo par de extremidades são ditos **arestas paralelas** (*parallel edges*) [DIESTEL, 2000].

Um grafo será dito **finito** se os vértices e as arestas forem também finitos. Um grafo sem nenhum vértice (consequentemente sem arestas) é um **grafo nulo**, e é de interesse por questões matemáticas. Um grafo com apenas um vértice é conhecido como **trivial**. Todos os outros grafos são **não triviais**. Um grafo é **simples** se ele não tiver laços e nem arestas paralelas, senão ele será um **multigrafo**.

Um **caminho** é um grafo simples cujos vértices podem ser dispostos em uma sequência linear de modo que dois vértices são adjacentes se são consecutivos na sequência [NETTO, 2006]. Da mesma forma, um **ciclo** de três ou mais vértices é um grafo simples cujos vértices podem ser dispostos em uma sequência cíclica, de modo que dois vértices são adjacentes se são consecutivos na sequência; um ciclo de um vértice consiste de um único vértice com um laço e um ciclo de dois vértices consiste em dois vértices unidos por um par de arestas paralelas. O **comprimento** ou **profundidade** de um caminho ou de um ciclo é o número de suas arestas. Um caminho ou ciclo de comprimento k é chamado **k-caminho** ou **k-ciclo**; o caminho ou ciclo é par ou ímpar de acordo com a paridade de k. Um 3-ciclo é chamado de triângulo, um 4-ciclo de quadrilátero, um 5-ciclo de pentágono, um 6-ciclo de hexágono, e assim por diante. Um caminho em um grafo é chamado **euleriano** se o caminho usa cada aresta exatamente uma vez e os vértices podem se repetir. Um caminho em um grafo é chamado **hamiltoniano** se o caminho visita cada vértice do grafo exatamente uma vez.

Embora os desenhos sejam uma forma conveniente de especificar grafos, eles não são adequados para armazenar grafos em computadores, ou para aplicar métodos matemáticos para estudo de suas propriedades. Dessa forma, é mais conveniente usar matrizes associadas a um grafo, a **matriz de incidência** e a **matriz de adjacência**. Seja G um grafo, com conjunto de vértices V e aresta E. A matriz de incidência de G é a matriz m × n de forma que M_G: = (m_{ve}), onde m_{ve} é o número de vezes (0, 1 ou 2) que o vértice **v** e aresta **e** são incidentes. Claramente, a matriz de incidência é apenas outra maneira de especificar o grafo. A matriz de adjacência de G é a matriz n × n de forma que A_G: = (a_{uv}), onde o a_{uv} é o número de arestas unindo vértices **u** e **v**, cada laço conta como duas arestas [DIESTEL, 2000].

Como a maioria dos grafos tem mais arestas do que vértices, a matriz de adjacência é geralmente menor do que a matriz de incidência e assim requer menos espaço de armazenamento. Ao lidar com grafos simples, uma representação mais compacta é possível. Para cada grafo **v**, os vizinhos de **v** são listados em alguma ordem. A lista (N(v): v ϵ V) destas listas é chamada de **lista de adjacência** do grafo. Grafos simples são normalmente armazenados em computadores como listas de adjacência.

O **grau** de um vértice **v** em um grafo G, denotado por dG(v), é o número de arestas de G incidente com **v**, cada laço contando como duas arestas. Em particular, se G é um grafo simples, o dG(v) é o número de vizinhos de v em G. Um vértice de grau zero é chamado de **vértice isolado**. Em um grafo orientado o número de arestas que incidem no vértice é chamado de grau de entrada (*indegree*) do vértice e o número de arestas que deixam o vértice é chamado de grau de saída (*outdegree*).

Dois algoritmos úteis para este trabalho são a busca em largura e a busca em profundidade. A **busca em largura** (*Breadth-First Search* ou BFS) pode ser feita para grafos que não possuem ciclos. Dados dois vértices quaisquer, existe exatamente um caminho entre eles. Um percurso em extensão visita cada nó começando pelo menor nível e move-se para os níveis mais altos, nível após nível, visitando cada nó da esquerda para a direita. Depois que um nó é visitado, seus filhos, se houver algum, são colocados no final da fila e o nó do início da fila é visitado. Assim, os nós do nível **n+1** serão visitados somente depois de ter visitados todos os nós do nível **n**. Computa-se a menor distância para todos os vértices alcançáveis. A **busca em profundidade** (*Depth-First Search* ou DFS) realiza uma busca que progride através da expansão do primeiro nó filho e se aprofunda cada vez mais, até que o último nível seja encontrado, ou seja, um nó que não possui filhos (nó folha). Então a busca retrocede e começa no próximo nó.

2.2.3 Grafos com atributos e transformações de grafos

Além de vértices e arestas, grafos podem conter textos e atributos para armazenar informações adicionais. Por exemplo, se um grafo representa uma rede de estradas entre cidades, as arestas podem conter o comprimento de cada segmento da estrada.

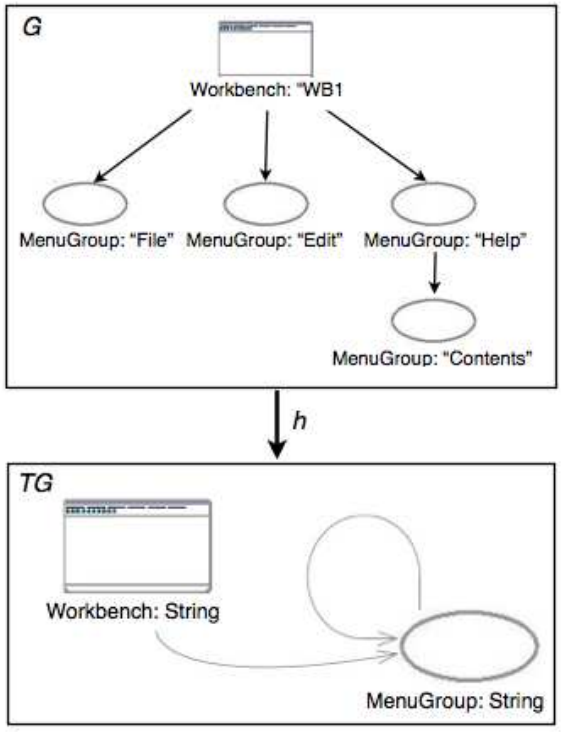
Uma gramática de grafos [ROZENBERG, 1997] generaliza as gramáticas de Chomsky de strings para grafos: eles especificam um sistema em termos de estados, descritos por grafos e mudanças de estados, descritas por regras tendo grafos do lado esquerdo e direito [RIBEIRO, 2012]. Uma gramática de grafos pode ser definida [RIBEIRO, 1997] identificando o grafo que representa a gramática, descrevendo as regras usando grafos, de forma que do lado esquerdo L de uma regra r : L → R em um grafo G, esta regra leve a substituição de L em G pelo lado direito R da regra;

A identificação do grafo define a sintaxe da linguagem, que é a estrutura dos grafos utilizados para construir a especificação e como estes grafos podem ser modificados. As regras definem a semântica da linguagem, ou seja, como as modificações descritas pelas regras são efetuadas. Grafos nos quais os vértices e arestas podem ser associados com atributos de algum tipo de dado são sempre chamados de grafos com atributos. Estes grafos com atributos geralmente consistem de duas partes: a parte gráfica e a parte dos dados. Podem ser usadas especificações algébricas para definir tipos de dados e álgebras para descrever os valores que podem ser usados como atributos.

Grafos são estruturas consistindo de um conjunto de vértices e um conjunto de arestas que conectam estes vértices. Um morfismo total entre os grafos é um mapeamento de vértices e arestas que é compatível com os fontes e destinos das arestas, intuitivamente o morfismo total de um grafo *G1* para um grafo *G2* significa que todos os itens (vértices e arestas) de *G1* podem ser encontrados em *G2* (mas distintos vértices e arestas de *G1* não são necessariamente distintos em *G2*). Se houver um grafo *TG*, que representa todos os possíveis tipos necessários para descrever um sistema, um morfismo total *h* de qualquer grafo *G* para *TG* terá um tipo associado para cada item de *G*. Esta tripla <*G*, *h*, *TG*> é chamada de grafo tipado e *TG* é chamado de grafo tipo (isto é, os vértices de *TG* descrevem todos os possíveis tipos de vértices de um sistema e as arestas de *TG* descrevem os possíveis relacionamentos entre estes vértices) [EHRIG, 1999].

Podemos associar atributos aos vértices ou arestas, ou seja, cada vértice ou aresta pode ter um conjunto de atributos que são valores de tipos de dados. Na Figura 2.10, é apresentado um grafo tipo com atributos. O grafo *TG* é um grafo tipo, tendo dois tipos de vértices, *Workbench* e *MenuGroup*, e dois tipos de arestas. Cada vértice de *TG* tem um atributo do tipo String. O grafo *G* é tipado sobre *TG*, isto é, todos os itens (vértices e arestas) de *G* devem ser mapeados para tipos de *TG* (indicado pela seta h).

Figura 2.10: Notação gráfica da gramática do exemplo.



A seguinte definição descreve grafos cujos vértices podem ter valores de algum tipo atribuídos.

Definição 1 (*Grafo com atributos e morfismo de grafo com atributos)*. *Um grafo G é uma tupla (vertG, edgeG, sourceG, targetG), onde vertG é um conjunto de vértices, edgeG é um conjunto de arestas e sourceG e targetG: edgeG → vertG são funções totais, definindo a fonte e destino de cada aresta, respectivamente.*

Dada uma especificação SPEC, um ***grafo com atributos*** *é uma tupla AG = (G, A, AttrG, valG, elemG), onde G é um grafo, A é uma SPEC-Algebra, AttrG é um conjunto e valG: AttrG → U(A), elemG: AttrG → vertG são funções totais. Vértices pertencendo ao AttrG são chamados* ***vértices com atributos***.

Dados dois grafos G = (vertG, edgeG, sourceG, targetG) e H = (vertH, edgeH, sourceH, targetH), um morfismo (parcial) de grafos f : G ↛ H é uma tupla (f_V : vertG ↛ vertH, F_E : edgeG ↛ edgeH) tal que f comuta com funções de origem e destino, isto é

$$\forall e \in dom(f\_E).f\_V(sourceG(e)) = sourceH(f\_E(e)) \text{ e}$$

$$\forall e \in dom(f\_E).f\_V(targetG(e)) = targetH(f\_E(e))$$

Um ***morfismo*** *(parcial)* ***de grafos com atributos*** *g entre grafos com atributos AG e AH é uma tripla g = (g_{Graph}, g_{Alg}, g_{Attr}) consistindo de um morfismo de grafo g_{Graph} = (g_V, g_E), uma álgebra homomorfismo g_{Alg} e uma função parcial g_{Attr} entre os componentes correspondentes que são compatíveis com a atribuição, isto é*

$$\forall a \in dom(g_{Attr}).g_{Alg}(valG(a)) = valH(g_{Attr}(a)) \text{ e}$$

$$g\_V(elemG(a)) = elem(H(g_{Attr}(a))$$

Um morfismo de grafo com atributos g é chamado total ou injetivo se todos os componentes são totais ou injetivos, respectivamente.

O papel do grafo tipo (*type graph*) é definir os tipos de vértice e arestas do grafo de instância. Assim, é adequado que a parte do grafo tipo que descreve os elementos de dados consista em nomes de tipos.

Definição 2 (Grafo tipo com atributos, tipos de grafo com atributos). *Dada uma especificação SPEC, um grafo tipo com atributos é um grafo com atributos AT = (T, A, AttrT, valT, elemT) na qual todos carrier sets de A são conjuntos singletons.*

Um ***grafo com atributos tipado*** *é uma tupla AG^{AT} = (AG, tAG, AT), onde AG é um grafo atribuído, chamado grafo instância, AT é um grafo tipo atribuído e tAG : AG → AT é um morfismo de grafo com atributos total chamado morfismo com atributos tipado.*

Um ***morfismo de grafo com atributos tipado*** *entre grafos AG^{AT} e AH^{AT} com grafo tipo com atributos AT é um morfismo grafo com atributos g entre AG e AH tal que tAG ≥ fAH ∘g (isto é, g só pode mapear entre elementos do mesmo tipo).*

Uma regra consiste em um lado esquerdo descrevendo itens que devem estar presentes num estado para habilitar a aplicação da regra e do lado direito, expressando os itens que estarão presentes após a aplicação da regra. Os possíveis atributos, no lado esquerdo e direito, serão variáveis e as possíveis relações entre estas variáveis serão expressas por equações associadas a cada regra. Quando aplicada uma regra, todas as

suas equações deverão ser satisfeitas pela atribuição de valores às variáveis escolhidas. Além disso, será exigido que as regras não colapsem vértices ou arestas (são injetivas) e não excluam vértices.

Definição 3 (Regra com atributos). *Dada uma especificação SPEC = (SIG, Eqns). Uma regra com atributos com* [EHRIG, 2008] *sobre a SPEC com tipo AT é a tripla attRule = (α, X, reluEqns) onde*

- X é um conjunto de variáveis sobre os tipos de SPEC;

- α: $AL^{AT} \nrightarrow AR^{AT}$ é um morfismo de grafo com atributos injetivo sobre a especificação (SIG, Ø) com AL = (L. $T_{OP}(X)$, AttrL, valL, elemL) e AR = (R, $T_{OP}(X)$, AttrR, valR, elemR), no qual α_V : vertL $\rightarrowtail$ vertR é uma função total sobre o conjunto de vértices e o componente álgebra é a identidade sobre o termo álgebra $T_{OP}(X)$;

- ruleEqns é o conjunto de equações usando termos de $T_{OP}(X)$:

Uma gramática de grafo com atributos em relação a algumas especificações de tipos de dados SPEC é composta de um **grafo tipo com atributos** (*attributed type graph*), um **grafo inicial** (*initial graph*) e um **conjunto de regras** (*set of rules*).

Definição 4 (Gramática de grafo com atributos). *Dada uma especificação SPEC e uma SPEC-álgebra A, uma* ***gramática de grafo com atributos (tipada)*** *é uma tupla AGG = (AT, AG0, R), tal que AT (o tipo da gramática) é um grafo tipo com atributos sobre SPEC, AG0 (o grafo inicial da gramática) é um grafo com atributos tipado sobre AT usando a álgebra A e R é o conjunto de regras sobre SPEC com tipo AT*.

O comportamento do sistema especificado por uma gramática de grafos é dado por aplicações sucessivas das regras da gramática no grafo inicial. As aplicações são feitas encontrando correspondências do lado esquerdo da regra no grafo que representa o estado atual do sistema. As correspondências das regras atribuídas devem levar em conta não só a parte gráfica, mas devem garantir que todas as equações da regra são satisfeitas pela atribuição de valores para as variáveis da regra.

2.2.4 Exemplo de Grafos com atributos

Para um melhor entendimento, foi criado um exemplo com um conjunto pequeno de possibilidades para criação de um modelo de UI. O sistema consiste dos seguintes elementos:

Workbench: bancada principal e pode ser de quatro tipos (MDI, MP, *Web*, Mob) distintos, deve ter um nome, uma sigla, um ícone opcional e um idioma;

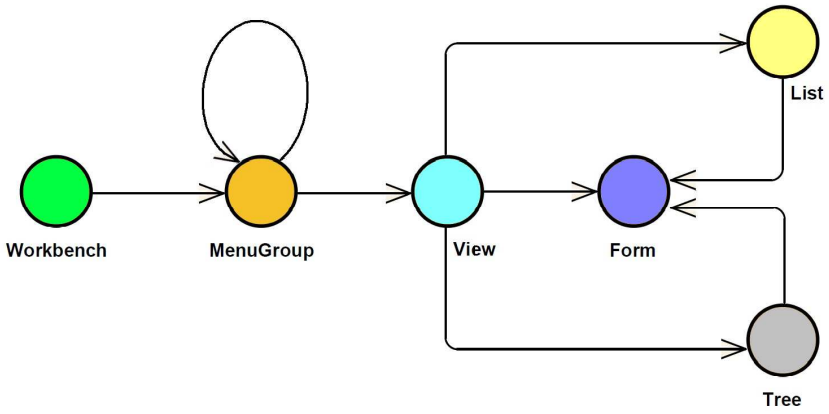
MenuGroup: identifica uma opção de *menu*, pode estar ligada em um *Workbench* ou a outro *MenuGroup*, deve ter um nome e uma ordem;

View: identifica um elemento de UI onde estarão as funcionalidades necessárias ao sistema. Uma *View* sempre estará ligada a um *MenuGroup*. Uma *View* deve ter como atributos o nome, a ordem e um ícone. Também pode ser composta por uma combinação de *List*, *Form* ou *Tree*, seguindo as seguintes regras: a) Uma *List*, ou um *Form* ou um *Tree*. b) Se tiver um *List* pode ainda ter um *Form*. c) Se tiver um *Tree* pode ter um *List* ou um *Form*.

Um sistema terá somente um *Workbench* e pode ter diversos *MenuGroups* e diversas *Views*, tendo pelo menos um *MenuGroup* e uma *View*. Com estas propriedades

definidas pode ser criada a gramática para a modelagem de um sistema. Na Figura 2.11 temos a notação gráfica da gramática.

Figura 2.11: Notação gráfica da gramática do exemplo.



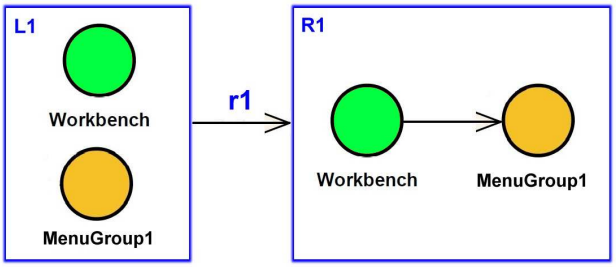
Quando for instanciado para criar um sistema, deve ser atribuídos valores aos atributos, por exemplo, no caso do *Workbench* conforme a Figura 2.12.

Figura 2.12: Atributos do *Workbench*.

```
workbench.Name = Business Structure
workbench.Tipo = MDI
workbench.Sigla = BS
workbench.Idioma = EN
```

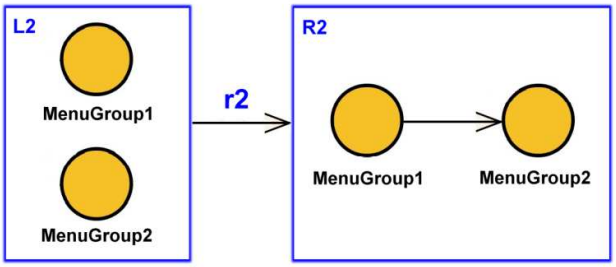
As regras formam a gramática de grafos da modelagem do sistema e assim definem o comportamento da construção da interface do sistema. Na Figura 2.13 a regra r1 transforma a construção de uma opção do *Workbench* com um *MenuGroup*.

Figura 2.13: Regra r1: Adição de *MenuGroup* no *Workbench*.



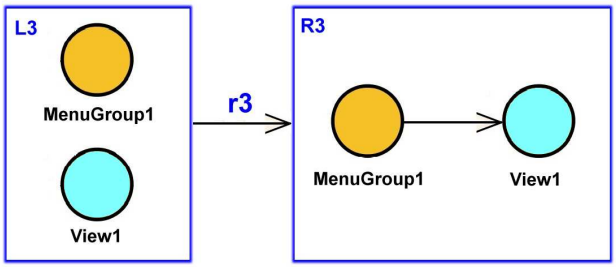
Na Figura 2.14 a regra r2 transforma a construção de uma opção *MenuGroup* encadeada em outro *MenuGroup*.

Figura 2.14: Regra r2: Adição de *MenuGroup2* ao *MenuGroup1*.



Na Figura 2.15 a regra r3 transforma a construção de uma *View* ligada em um *MenuGroup*.

Figura 2.15: Regra r3: Adição da *View1* ao *MenuGroup1*.



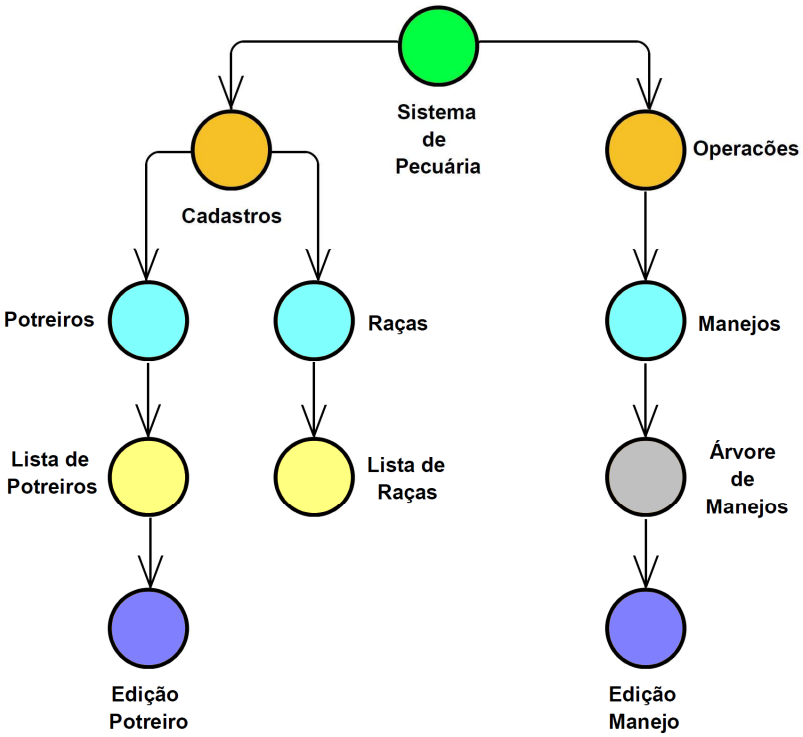
Seguindo o mesmo processo são construídas outras seis regras que definem a gramática e na tabela 2.1 estão apresentadas todas as regras.

Tabela 2.1: Regras de Criação

Regra	*Descrição*
r1	Adição *MenuGroup* ao *Workbench*
r2	Adição *MenuGroup* a um *MenuGroup*
r3	Adição de *View* a um *MenuGroup*
r4	Adição de *List* a uma *View*
r5	Adição de *Form* a uma *View*
r6	Adição de *Tree* a uma *View*
r7	Adição de *Form* a um *List*
r8	Adição de *List* a uma *Tree*
r9	Adição de *Form* a uma *Tree*

Utilizando as regras é possível criar o modelo de um sistema como apresentado na Figura 2.16, onde temos a estrutura de um sistema hipotético.

Figura 2.16: Exemplo de um modelo de sistema hipotético.



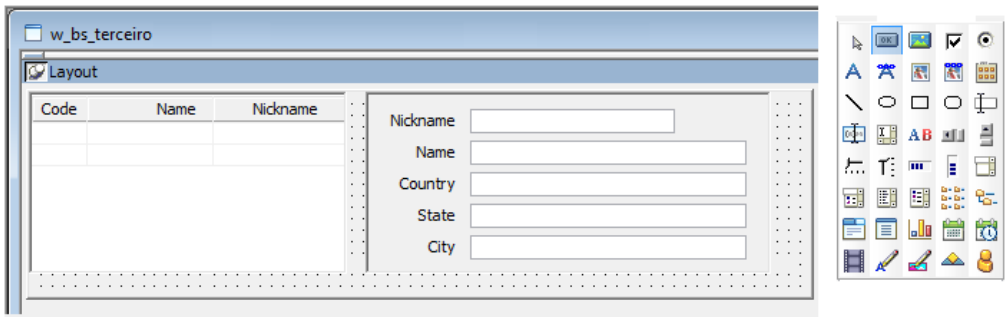
3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados os trabalhos relacionados com o objetivo de estabelecer comparativos e ao final, junto com as conclusões, será apresentada uma tabela comparando os diversos trabalhos.

3.1 Construção de Interfaces de Usuário com "Drag and Drop"

Diversas ferramentas permitem a construção da interface do usuário utilizando controles de interface que podem ser arrastados e soltos em uma janela para construir a interface. Estas ferramentas facilitam bastante o trabalho de construção, mas o processo é totalmente artesanal, pois cada controle e seus relacionamentos devem ser desenvolvidos para compor uma interface do usuário.

Figura 3.1: Construção de Interfaces do Usuário com "*Drag and Drop*".



Na Figura 3.1 temos um exemplo de uma interface do usuário construída com uma ferramenta "*Drag and Drop*". Cada controle da interface deve ser selecionado na caixa de ferramentas e arrastado para a interface e dessa forma ela é construída. Praticamente todas as ferramentas *Rapid Application Development* (RAD) têm meios para a construção de interfaces do usuário dessa forma, pode ser citado como exemplos dessas ferramentas, o *Powerbuilder* da *Sybase* [POWERBUILDER, 2012], o *Visual Studio* da *Microsoft* [VS, 2012], o *Javafx Scenebuilder* da *Oracle* [JAVAFX, 2012] entre outras. É importante destacar que é bastante difícil manter o mesmo padrão para as *interfaces*, pois cada uma é criada manualmente e não há como automatizar a geração dessas UIs e estas *interfaces* são dependentes da plataforma para a qual estão sendo desenvolvidas. A

vantagem desta abordagem é que o projetista está totalmente livre para criar a interface, podendo colocar controles da forma que desejar.

3.2 Especificação de Interfaces de Usuário por Modelos UML

Outra abordagem para construir interfaces do usuário é pela geração a partir de modelos UML, onde a interface é criada como um modelo UML [UML, 2012] e, por transformação deste modelo, a interface concreta é construída. Esta abordagem usa a *Model Driven Architecture* (MDA) [OMG, 2003] [GUTTMAN, 2007] como suporte para a geração.

O trabalho de Rosado da Cruz [ROSADO DA CRUZ, 2010] cria um metamodelo para facilitar a geração dos modelos de interface do usuário (UIM) a partir de um modelo de casos de uso (UCM) e um modelo de domínio (DM). Ambos o UCM e o DM estão embasados em metamodelos estendidos da UML. Partindo do UIM, por transformação de modelo *model-to-code*, gera as UIs concretas. A premissa deste trabalho é que, para sistemas de informação, grande parte das funcionalidades e a estrutura de uma UI estão bastante relacionadas com a estrutura e funcionalidades descritas no modelo de domínio e casos de uso. O modelo de casos de uso é utilizado para definir as funcionalidades e também quais entidades do modelo de domínio estão disponíveis para cada tipo de ator, uma vez que as entidades estão ligadas aos casos de uso (via *tagged-values*) e, para cada caso de uso, existe o ator que pode executá-lo. Os casos de uso estão divididos entre independentes e dependentes. Os casos de uso independentes serão transformados em pontos de entrada da aplicação, como opções de *menu* e podem ser iniciados pelo ator, os casos de uso dependentes somente podem ser iniciados pelo caso de uso dos quais eles dependem. O modelo de domínio é utilizado para definir a estrutura e também contem extensões para propriedades e operações, de modo que possa gerar o UIM sem a necessidade do UCM. As operações possíveis são as operações de CRUD.

Outro trabalho é o MERLIM [MRACK, 2009] que gera código fonte a partir de um conjunto de modelos UML. O modelo de domínio que é o modelo principal e na prática são classes Java. Este modelo provê as informações necessárias para a estruturação das interfaces do usuário. Outros modelos utilizados são o modelo de apresentação, o modelo de tarefas e o modelo de diálogos. O modelo de apresentação é implementado com o uso de *Java Annotations*, o modelo de tarefas são métodos de classes onde podem existir pré-condições. O modelo de diálogo também é implementado com *Java Annotations* e fazem a ligação entre o modelo de apresentação e o modelo de tarefas.

O método XIS [SILVA, 2007] é outro trabalho que gera sistemas interativos partindo de um modelo de domínio, um modelo de casos de uso, um modelo de atores e um modelo de interface do usuário. As entidades do modelo de domínio são ligadas com os casos de uso. Uma das desvantagens do método XIS é a inflexibilidade das UIs geradas, pois, segundo o autor, elas são fixas e não permitem extensões.

Outro trabalho [ELKOUTBI, 2006] gera a UI a partir de cenários de casos de uso, modelo de estrutura e modelo de comportamento de UI, mas existe a necessidade de informações relacionadas à UI como os campos de entrada e saída.

Uma vantagem desta abordagem por modelos UML é que as interfaces do usuário podem ser geradas para múltiplas plataformas, pois o modelo é independente, temos

também como vantagem a garantia de padronização da UI, pois ela será construída por geração. Por outro lado, o esforço para criar um modelo de UI pode ser maior do que a abordagem *Drag and Drop*.

3.3 Abordagem baseada em metamodelo e padrões

Nesta abordagem, além dos modelos UML também são utilizados padrões para gerar as UIs, onde os modelos UML são ligados com os padrões identificados. O trabalho de Molina [MOLINA, 2003] utiliza um modelo conceitual para especificar um sistema de informação, identifica as interfaces do usuário necessárias, relaciona com abstrações padronizadas e, por fim, usa geração de código para criar as *interfaces* concretas. No modelo conceitual são descritas as estruturas e comportamento do sistema utilizando modelos UML. Uma vez identificadas as funcionalidades necessárias ao sistema, elas são associadas com interfaces do usuário. Utilizando uma linguagem de padrões, chamada *Just-UI*, que é composta por três níveis e dezesseis padrões, as funcionalidades são associadas a estes padrões. Esta estrutura é utilizada como base para a geração concreta das interfaces do usuário. O primeiro nível permite especificar a hierarquia de ações dependendo do tipo de usuário. No segundo nível são especificadas as unidades de interação que podem ser um serviço, uma instância que é um formulário, uma população que representa uma lista ou uma janela do tipo mestre e detalhe. Uma janela mestre e detalhe é uma construção típica de UI usada para casos com pedido e itens de pedido. No terceiro nível os padrões elementares são especificados em complemento às unidades de interação, e é composto por onze padrões. Em cada um desses níveis existe uma representação gráfica para a modelagem e assim os modelos podem ser construídos. Por fim, utilizando estes modelos, um gerador mapeia os padrões em código para a construção da aplicação concreta. Seguindo a mesma abordagem, em continuidade aos padrões identificados no trabalho anterior, outro trabalho [PASTOR, 2006] reestrutura os padrões e cria um processo baseado em MDA para a geração de código das UIs de um sistema de informação.

Outro trabalho que pode ser citado aqui é o ZOOM [JIA, 2007], que modela um sistema construindo um modelo gráfico que é traduzido para uma linguagem chamada ZOOM, que é traduzida em uma aplicação executável. O modelo estrutural contém as classes e um modelo de máquina de estados modela o comportamento. Um mecanismo conecta o modelo estrutural com o modelo de UI usando componentes pré-definidos.

Uma vantagem adicional desta abordagem com metamodelo em relação à abordagem por modelos UML é que os modelos são criados embasados neste metamodelo e desta forma existe uma maior padronização nos modelos criados. Um dos trabalhos desta abordagem [MOLINA, 2003] aprimora mais associando padrões ao modelo.

3.4 Interfaces do Usuário com Transformações de Grafos

Nesta abordagem o sistema de informação tem suas UIs modeladas por grafos e a geração pode ser feita por transformações de grafos. O trabalho de Limbourg [LIMBOURG, 2005], chamado TOMATO, utiliza grafos como base para modelagem e geração das UIs. Neste trabalho é discutido que um dos grandes problemas de gerar UIs

é problema de mapeamento identificado por Puerta and Eisenstein [PUERTA, 1999], onde existe uma grande dificuldade em relacionar os elementos abstratos do modelo com os elementos concretos da UI. A metodologia TOMATO (*formal meThOdology for MApping user interface specificaTiOn models*) é composta de um ciclo de desenvolvimento e uma linguagem (TOMATO-L) que suporta o desenvolvimento transformacional das UIs. Cada UI é representada com um conjunto de modelos que podem ser analisados, editados e processados. Cada modelo é armazenado dentro de um repositório em uma linguagem de especificação de UI baseada na teoria dos grafos. Esta representação pode ser submetida a regras de produção que progressivamente transformam conceitos abstratos em conceitos concretos para finalmente criar a descrição final da UI e com esta descrição, usando uma ferramenta de renderização produzir o código fonte. Na metodologia TOMATO os mapeamentos são feitos de forma gráfica, onde as funções do sistema são mapeadas com os domínios. A especificação inicial é detalhada em níveis menores. Partindo desta especificação, uma sequência de transformações é executada, algumas usando uma abordagem manual e outras automáticas.

Os conceitos abstratos são formalizados com estruturas de grafos com as abstrações necessárias para construir uma UI em uma abordagem dirigida por modelos. Estas abstrações são categorizadas em três classes. A primeira classe é um modelo de componentes necessários para construir as *views* de uma UI e este modelo contém um modelo funcional (*task model*), um modelo de domínio (*domain model*), um modelo de apresentação (*presentation model*), um modelo de diálogo (*dialog model*) e, por fim, um modelo de contexto (*context model*). A segunda classe mapeia os relacionamentos entre os modelos de componentes. A terceira classe é o conhecimento que inserido na UI via uma gramática de grafos por um conjunto de regras de produção. Estas regras vão possibilitar a transformação dos grafos representando as UIs. A técnica de transformação utilizada é *Single PushOut approach* (SPO). A sintaxe concreta da UI é expressa em TOMATO-L que tem uma sintaxe gráfica e uma linguagem textual do tipo XML. Para as transformações foi utilizado o ambiente AGG [EHRIG, 1999] que foi escolhido por ter uma linguagem para especificar a gramática de grafos e um interpretador que permite definir as transformações de grafos. Segundo o próprio autor, um grande esforço de abstração será necessário pelas pessoas que forem responsáveis por incorporar o conhecimento de projeto usando a metodologia TOMATO.

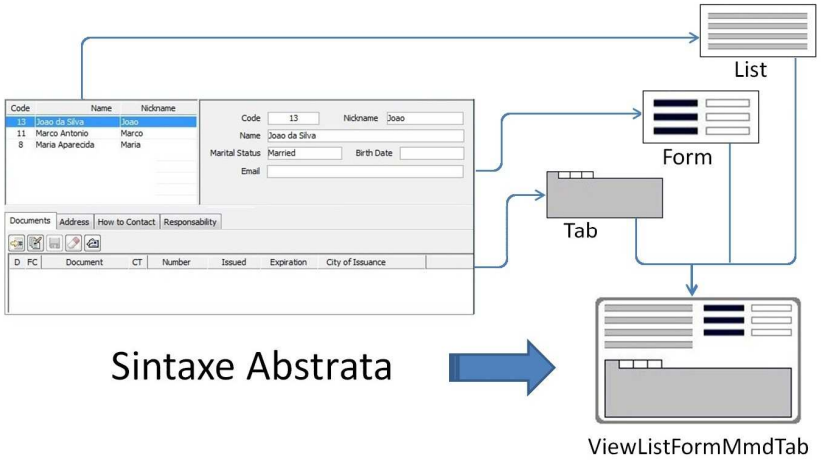
4 PROPOSTA E VALIDAÇÃO DO METAMODELO

Neste capítulo, é descrito como foram criados os padrões e como eles foram utilizados para a criação do metamodelo proposto. Também é discutido como o metamodelo foi validado. Inicialmente foram analisados diversos sistemas de informação para identificar os padrões de interface existentes. Estes padrões foram normalizados e para cada padrão foi criado um elemento visual. A partir destes elementos visuais e dos relacionamentos possíveis entre eles foi criado o metamodelo como um grafo. Partindo deste metamodelo foram modelados alguns sistemas para validar o metamodelo.

4.1 Padrões de UIs identificados

Inicialmente, foram identificados os padrões da bancada de trabalho, que é a área completa do sistema. Ela é apresentada quando o sistema é executado e aqui é chamado de *Workbench*. Em continuidade foram identificados os diversos tipos de UIs, onde são implementadas as funcionalidades do sistema e aqui são chamadas de *Views*. Para cada *View* foram identificados os tipos básicos de apresentação e manipulação de dados. Também foram analisados os padrões para invocar as *Views*.

Figura 4.1: Criação de um Padrão.



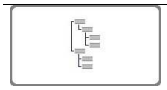
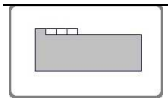
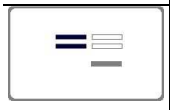
Na Figura 4.1 está representado o processo utilizado para identificar e definir os elementos visuais da UI. Nesta figura, pode ser visto que uma parte da interface contém uma lista, que está representada por um elemento primitivo *List*, a interface também contém um formulário que permite a visualização e edição dos dados do registro

selecionado na lista. Este formulário é representado por um elemento primitivo *Form*. Na interface também há diversas abas que contém outras informações sobre o registro selecionado na lista. Esta aba está representada por um elemento primitivo *Tab*.

Juntando estes elementos primitivos na *View* foi criada a sua sintaxe abstrata e nomeada como uma *View List Form Master multi detail Tab*, ou de forma resumida, *ViewListFormMmdTab*. Uma *View* é composta de elementos primitivos, ou seja, elas podem conter formulários (*Form*), listas (*List*), árvores (*Tree*), abas (*Tab*) e filtros (*Filter*). Esta estratégia apresentada anteriormente foi utilizada para a criação de cada um dos padrões identificados. Inicialmente a interface foi decomposta em cada tipo de elemento primitivo de apresentação de dados. Após foi identificado como estes elementos primitivos se relacionam e então foi criada uma *View* para representá-los. Como resultado foi construído um conjunto de *Views* padronizando todos os tipos de UIs encontrados e cada uma dessas *Views* foi nomeada com a regra identificada anteriormente, de acordo com os elementos primitivos e a solução que ela implementa.

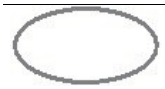
Para cada elemento primitivo foram levantados os padrões de aparência e de comportamento. Também foi identificado, para cada *View*, sua composição com os elementos primitivos e os seus padrões de aparência e comportamento e também o relacionamento entre os elementos primitivos dentro desta *View*. Para o *Workbench*, para os *Menus* e para os elementos primitivos também foram criadas representações e todos estes elementos foram sumarizados. Nos Apêndice A, B e C estão apresentadas detalhadamente a análise dos sistemas utilizados para identificar os elementos e também a relação completa de todos os elementos de UI identificados. Nas Tabelas 4.1 a 4.6 estão apresentados todos os padrões que foram criados neste trabalho. Eles foram desenvolvidos após a normalização das UIs dos sistemas de informação analisados. Este conjunto de elementos foi a base para a criação do metamodelo. Na Tabela 4.1 são apresentados os elementos primitivos.

Tabela 4.1: Elementos primitivos

Icone	*Nome*	*Descrição*
	*Form*	Representa um formulário.
	*List*	Representa uma lista.
	*Tree*	Representa uma árvore.
	*Tab*	Representa um conjunto de abas.
	*Filter*	Representa um filtro.

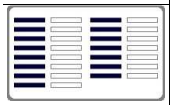
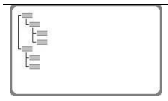
Na Tabela 4.2 são apresentados os elementos que identificam o *Workbench* e a estrutura de *menu*.

Tabela 4.2: *Workbench* e Estrutura de *Menu*

Icone	*Nome*	*Descrição*
	*Workbench*	Identifica a interface inicial do sistema.
	*MenuGroup*	Identifica um grupo de opções de *menu.*
	*MenuAction*	Identifica uma ação realizada no *menu.*
	*MenuFlag*	Identifica uma propriedade do sistema, ligada ou desligada no próprio *menu.*

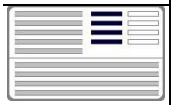
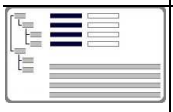
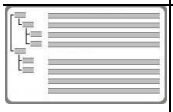
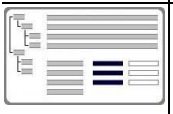
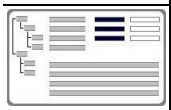
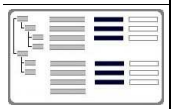
Na Tabela 4.3 são apresentados os padrões de *Views* simples que foram identificados.

Tabela 4.3: *Single Views*

Icone	*Nome*	*Descrição*
	*ViewVoid*	Identifica uma *View* básica.
	*ViewForm*	Representa uma *View* que tem por objetivo apresentar um formulário.
	*ViewList*	Representa uma *View* que tem por objetivo apresentar uma lista.
	*ViewListForm*	É uma *View* que tem uma lista e para cada registro da lista selecionado e apresenta um formulário.
	*ViewTree*	Representa uma *View* que tem por objetivo apresentar uma árvore.
	*ViewTreeForm*	É uma *View* que tem uma árvore e para cada nodo da árvore selecionado e apresenta um formulário
	*ViewTreeList*	É uma *View* que tem uma árvore e para cada nodo da árvore selecionado e apresenta uma lista.
	*ViewTreeListForm*	É uma *View* com uma árvore e para cada nodo da árvore selecionado, apresenta uma lista e para cada registro da lista selecionado apresenta o formulário.

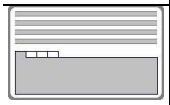
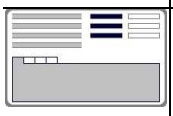
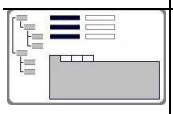
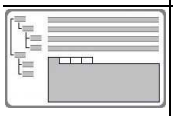
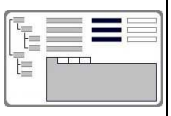
Na Tabela 4.4 são apresentados os padrões de *Views* compostas, que implementam soluções do tipo mestre e detalhe.

Tabela 4.4: Composite Views

Icone	*Nome*	*Descrição*
	*ViewListMdList*	*View* com o mestre em uma lista e o detalhe também numa lista.
	*ViewListMd ListForm*	*View* com o mestre em uma lista e o detalhe também em uma lista. Para cada registro do detalhe tem um formulário.
	*ViewListFormMd List*	*View* com o mestre em uma lista e cada registro da lista tendo um formulário. O detalhe é uma lista.
	*ViewListFormMd ListForm*	*View* com o mestre em uma lista e cada registro da lista tendo um formulário. O detalhe é uma lista com cada registro da lista tendo um formulário.
	*ViewTreeFormMd List*	*View* com o mestre em uma árvore e cada nodo da árvore tendo um formulário. O detalhe é uma lista.
	*ViewTreeListMd List*	*View* com o mestre em uma árvore e cada nodo da árvore tendo uma lista. O detalhe é uma lista.
	*ViewTreeListMd ListForm*	*View* com o mestre em uma árvore e cada nodo da árvore tendo uma lista. O detalhe é uma lista. Para cada registro do detalhe tem um formulário.
	*ViewTreeListForm MdList*	*View* com o mestre em uma árvore e cada nodo da árvore tendo uma lista e cada registro da lista com um formulário. O detalhe é uma lista.
	*ViewTreeListForm MdListForm*	*View* com o mestre em uma árvore e cada nodo da árvore tendo uma lista e cada registro da lista com um formulário. O detalhe é uma lista e formulário.

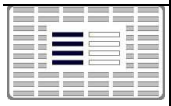
Na Tabela 4.5 são apresentados os padrões de *Views* compostas, que implementam soluções do tipo mestre e múltiplos detalhes.

Tabela 4.5: Multi Composite *Views*

Icone	*Nome*	*Descrição*
	*ViewListMmdTab*	*View* com uma lista e para cada registro da lista tem diversas abas relacionadas.
	*ViewListForm MmdTab*	*View* com uma lista e um formulário para cada registro da lista tem diversas abas relacionadas.
	*ViewTreeForm MmdTab*	*View* com uma árvore e um formulário para cada nodo da árvore tem diversas abas relacionadas.
	*ViewTreeList MmdTab*	*View* com uma árvore e uma lista onde para cada nodo da árvore tem uma lista e para cada registro da lista tem diversas abas relacionadas.
	*ViewTreeListForm MmdTab*	*View* com uma árvore e uma lista onde para cada nodo da árvore tem uma lista e um formulário e para cada formulário tem diversas abas relacionadas.

Na Tabela 4.6 são apresentados os padrões de *Views* com referência cruzada.

Tabela 4.6: Crosstab Views

Icone	*Nome*	*Descrição*
	*ViewCrosstab*	*View* com uma referência cruzada para apresentar dados.
	*ViewCrosstabForm*	*View* com uma referência cruzada para apresentar dados e também editar em um formulário.
	*ViewCrosstabList*	*View* com uma referência cruzada para apresentar dados e também editar em uma Lista.

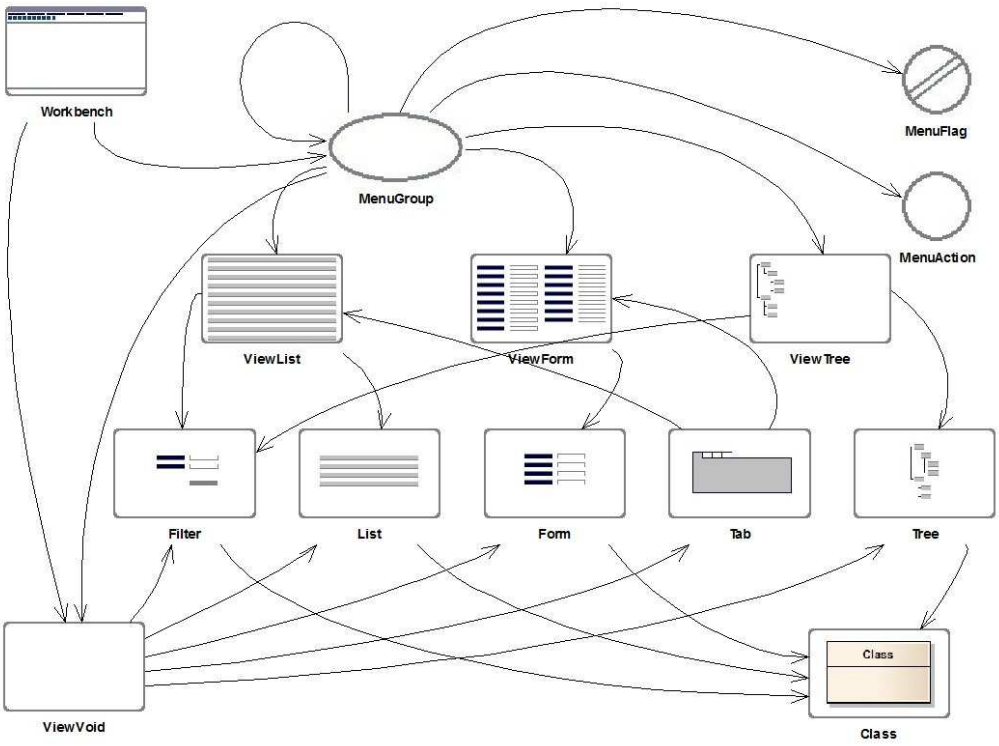
Como está definida a aparência e o comportamento de cada elemento, não é necessário especificar detalhes sobre eles quando da implementação do mesmo, e mais importante, este comportamento e esta aparência estarão sempre padronizados ao serem utilizados como sintaxe abstrata para uma transformação concreta, gerando código fonte, o que deve ser desenvolvido em continuidade a este trabalho. Estes padrões contemplam praticamente todas as características de uma boa interface, exceto a memória prospectiva, a repetição simplificada e comentários de outras pessoas. Estas características podem facilmente ser implementas posteriormente em um sistema de informação, caso seja necessário. Um próximo passo é a padronização dos elementos de dados, que será objeto de um trabalho futuro.

4.2 Definição do Metamodelo

Considerando que cada elemento de interface do usuário pode ser considerado como um vértice e que o relacionamento entre os elementos podem ser considerados como arestas, foi criado o metamodelo como uma estrutura de grafos orientados. Assim, cada nó representa um elemento possível da modelagem e as arestas são as relações possíveis entre estes elementos. As restrições são definidas nas arestas e os atributos são definidos tanto nas arestas como em cada nó, dependendo do tipo de atributo. Como mencionado anteriormente, a escolha de uma estrutura de grafos foi feita por ser uma forma natural de representar estes tipos de os elementos. Esta mesma abordagem já foi feita em outros trabalhos [BIERMANN, 2008] que utilizam modelos de grafos e transformações. Usando grafos podemos utilizar transformações de grafos [EHRIG, 2006] e para isto existe uma vasta teoria que pode ser aplicada para as transformações de modelo.

A Figura 4.2 apresenta uma visão de uma parte do metamodelo. Nela podem ser vistos os relacionamentos entre o *Workbench*, as opções de *Menus* e algumas das *Single Views* e também os seus relacionamentos com os elementos primitivos. Seguindo a mesma estratégia foi criado o metamodelo completo do grafo com o relacionamento entre todos os vértices.

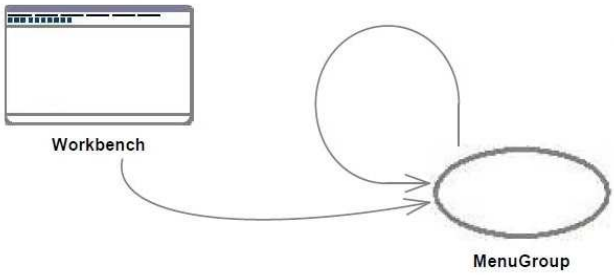
Figura 4.2: Parte do Metamodelo.



Para um melhor entendimento do metamodelo, um exemplo é apresentado para mostrar os elementos e seus relacionamentos. Na Figura 4.3 é apresentado o relacionamento entre o *Workbench* e o *MenuGroup*. Este relacionamento define no metamodelo a possibilidade de construir as opções de *menu* que o sistema de

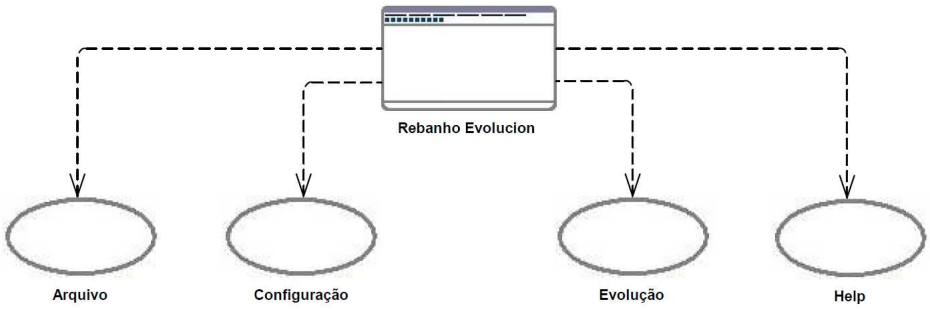
informação modelado irá contemplar. É possível identificar que um *Workbench* pode conter diversos *MenuGroups* e um *MenuGroup* pode ter vários níveis.

Figura 4.3: Relacionamento do *Workbench* e *MenuGroup*.



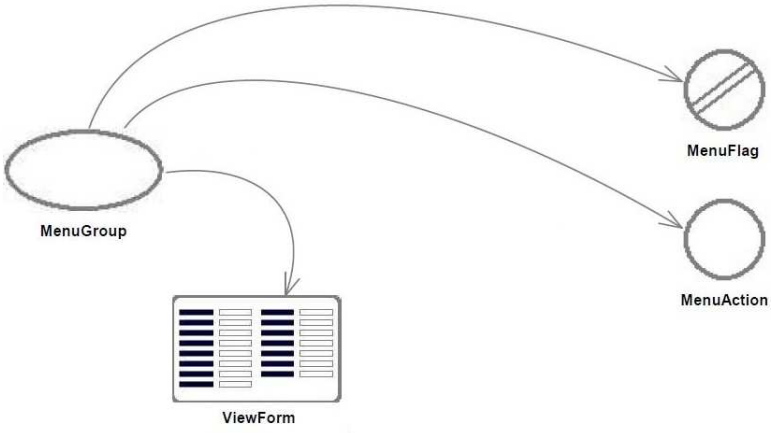
Um exemplo de modelo para o *Workbench* e *MenuGroup* é apresentado na Figura 4.4, onde é possível identificar as opções disponíveis do menu do sistema.

Figura 4.4: Modelagem usando o *Workbench* e *MenuGroup*.



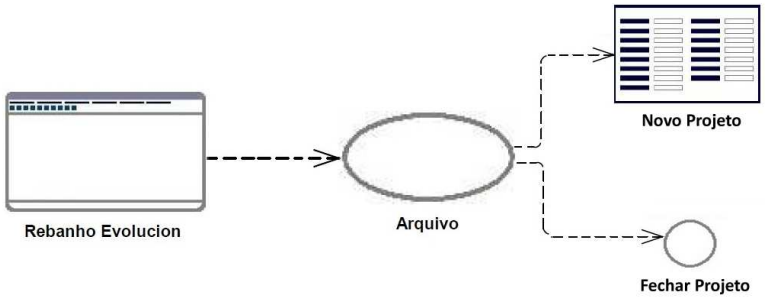
No metamodelo está definido que um *MenuGroup* pode estar conectado com um *MenuFlag*, *MenuAction* ou uma *View*, que pode ser qualquer uma das *Views* identificadas na Tabela 4.3 até a Tabela 4.6. Do ponto de vista de modelagem do sistema, estas conexões definem as opções que executam alguma operação no sistema. Um *MenuFlag* é utilizado para ligar ou desligar propriedades, um *MenuAction* é uma opção onde não há necessidade de uma UI e uma *View* implementa uma UI propriamente. A Figura 4.5 apresenta a parte do metamodelo que define as conexões do *MenuGroup* com um *MenuFlag*, *MenuAction* e uma *ViewForm*.

Figura 4.5: Relacionamento *MenuGroup* com *MenuFlag*, *MenuAction* e *ViewForm*.



Um exemplo de modelo para o *MenuGroup* e uma opção com *MenuAction* e outra como *ViewForm* é apresentado na Figura 4.6, onde é possível identificar duas funcionalidades do sistema.

Figura 4.6: Modelagem usando *MenuGroup* com *MenuAction* e *ViewForm.*



Na Figura 4.7 é apresentado um exemplo concreto da modelagem que foi exemplificada acima. É possível identificar que a opção de arquivo tem as opções Novo Projeto e Fechar Projeto. A opção Fechar Projeto não tem interação com o usuário, é simplesmente uma *Action* que faz o fechamento do projeto em uso. A opção Novo projeto tem um formulário para serem informados os dados iniciais do Projeto. A modelagem é representada como uma *ViewForm* e na Figura 4.8 é apresentado a UI concreta do exemplo.

Figura 4.7: *Menu* concreto de um sistema de informação.

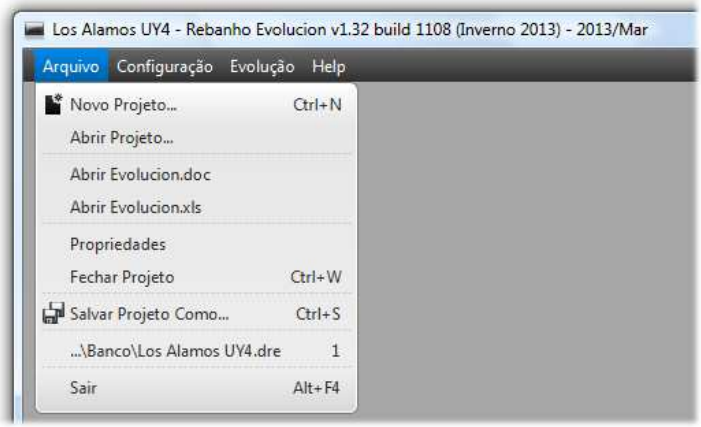


Figura 4.8: Uma *ViewForm* concreta de um sistema de informação.

4.2.1 A Estrutura do Metamodelo

O metamodelo define como pode ser estruturado um modelo de um sistema de informação. A estrutura parte de um *Workbench* que pode ser ligado a grupos de *menu* (*MenuGroup*). A esses podemos ligar flags que identificam atributos que podem estar ligados ou não (*MenuFlag*), ações que podem ser executadas diretamente no *Menu* (*MenuAction*) e também às diversas *Views* que é onde são implementadas as funcionalidades do sistema. O Apêndice D contém a descrição completa do metamodelo.

4.3 Validação do Metamodelo

Para validar o metamodelo foram construídos modelos para um exemplo hipotético e para três sistemas reais. No sistema hipotético foi criada uma definição e modelado o sistema proposto. Para os sistemas reais, foi feita uma engenharia reversa para avaliar a possibilidade de representar suas *interfaces* através de um modelo embasado no metamodelo.

4.3.1 Exemplo de um sistema hipotético

Para modelar um exemplo de um sistema de informação foi criada a definição hipotética. Esta definição é para um sistema de matrículas de uma universidade, onde temos alguns conceitos como departamento, curso, disciplina, professor, aluno, turma, matrícula e histórico.

Algumas regras que definem o sistema são:

- Um departamento é responsável por diversos cursos;
- Um curso tem diversas disciplinas;
- Um professor pode lecionar diversas disciplinas;
- Um aluno está inscrito somente em um curso;
- Um aluno pode se matricular em diversas turmas;

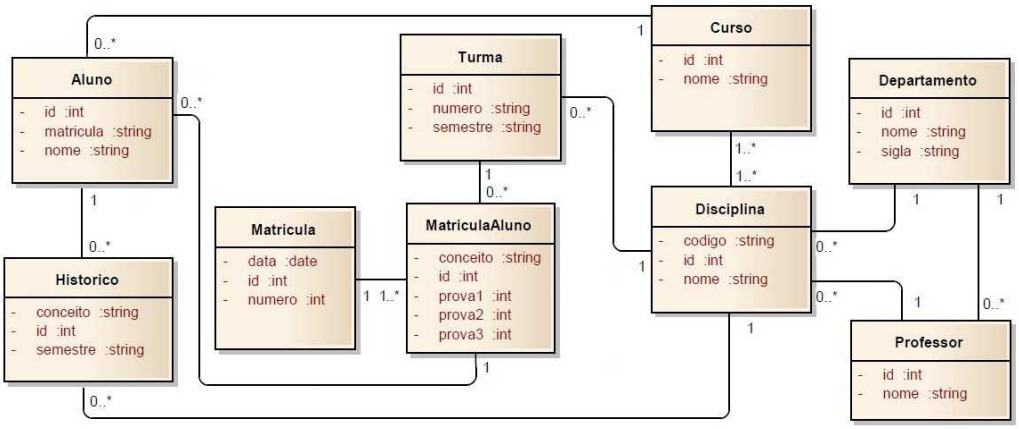
- Uma turma é para uma disciplina;
- Uma turma tem um único professor;
- Uma turma está vinculada para um semestre;
- Um aluno só pode se matricular em turmas do semestre corrente;
- Um aluno é avaliado por três provas em cada turma;
- Um aluno só pode se matricular em disciplinas não cursadas ou cursadas sem aproveitamento;
- O código da disciplina é a sigla do departamento mais um número e um complemento;

Alguns procedimentos necessários para o desenvolvimento do sistema:

- Na matrícula o aluno seleciona turmas do semestre corrente;
- No final de semestre é feito o fechamento, onde é conferido se todas as turmas estão fechadas (resultado de cada aluno preenchido entre aprovado ou reprovado), atualizado o histórico e zerada a matrícula;

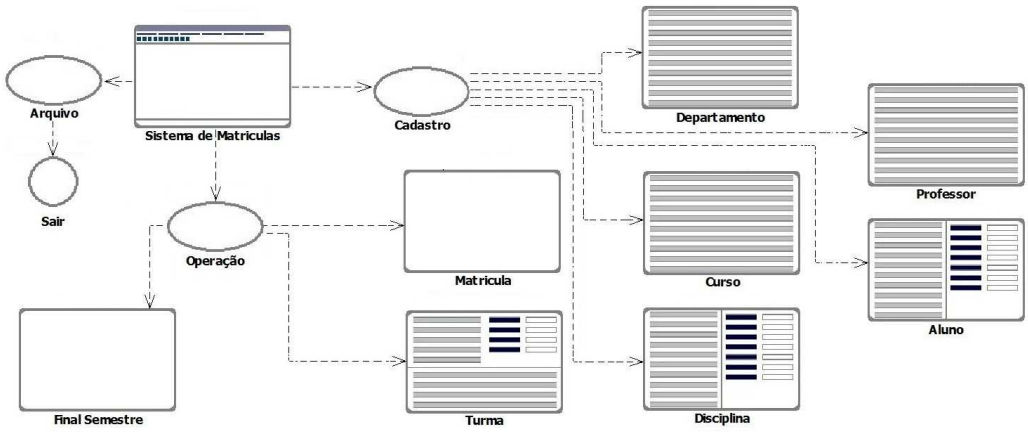
A Figura 4.9 apresenta o modelo UML com as classes referentes à persistência dos dados necessários ao sistema e também a representação de seus relacionamentos. No contexto deste trabalho, este modelo de classes é utilizado apenas para exemplificar os dados que serão manipulados nas interfaces do usuário e persistidos em uma base de dados.

Figura 4.9: Modelo Classes do Sistema de Matrículas.



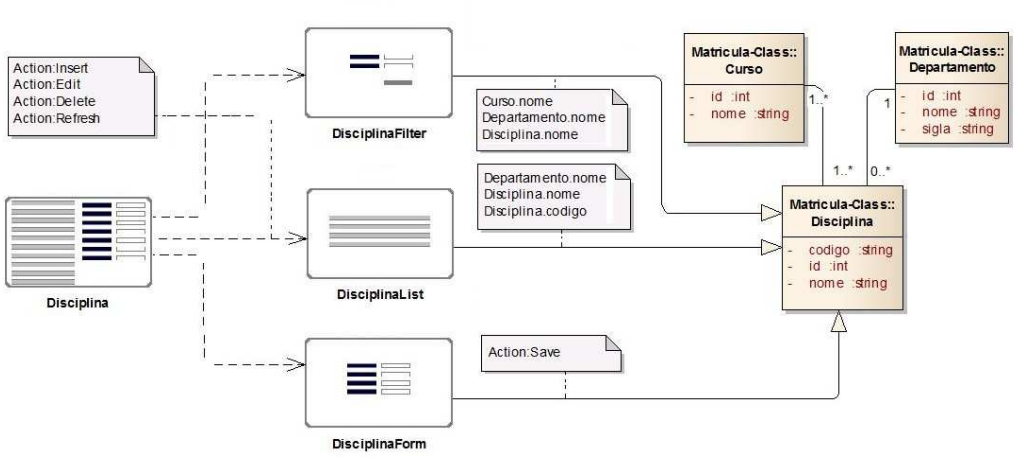
Na Figura 4.10 é apresentado o exemplo da modelagem deste sistema de matrícula construído com base no metamodelo.

Figura 4.10: Modelo do Sistema de Matrículas.



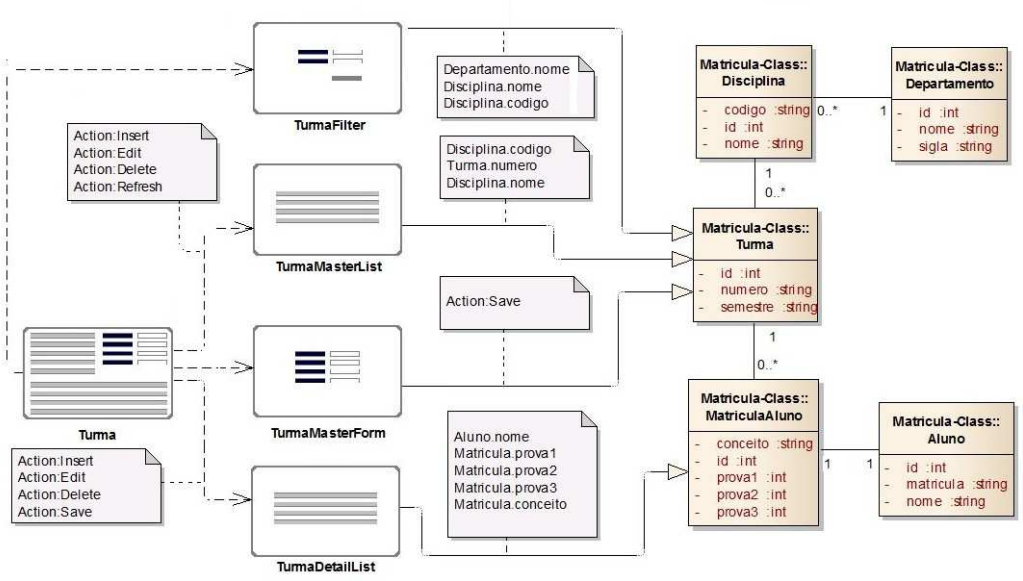
Cada uma destas representações de *Views* deve ser detalhada, na Figura 4.11 é apresentado o detalhamento para Disciplina. A *View* está composta de três padrões básicos, um *Filter*, um *List* e um *Form.* O *Filter* restringe os dados da lista de disciplinas e ele tem uma ligação com a classe que fornece os dados. Quando ele é executado se aplica aos registros do *List*. O *Form* se aplica a um determinado registro do *List.*

Figura 4.11: Modelo detalhado para Disciplina.



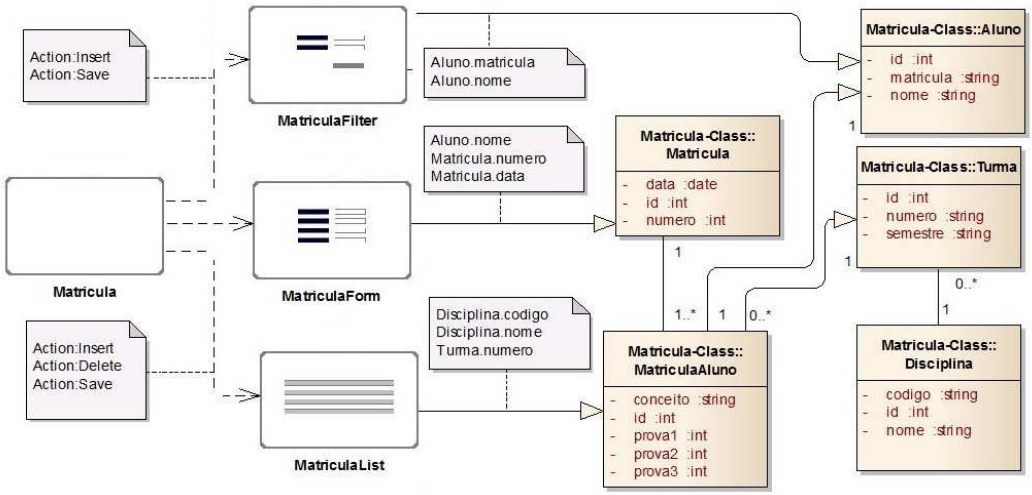
A Figura 4.12 mostra o detalhamento para Turma, que está modelada como um mestre e detalhe. O mestre possui uma opção de filtro e mostra os dados numa lista. A edição dos dados do mestre é através de um *Form.* Para cada registro de Turma, selecionada na lista, são mostrados os Alunos desta turma. A edição dos registros de alunos da Turma é feito na própria lista.

Figura 4.12: Modelo detalhado para Turma.



A funcionalidade Matricula é uma funcionalidade que não tem um padrão que se ajuste integralmente, mas podemos usar uma *ViewVoid* e compor com os padrões primitivos, o que é apresentado na Figura 4.13. Este procedimento é a solução para as UIs que não têm um padrão específico por não ocorrerem em número suficiente a ponto de ser criado um padrão.

Figura 4.13: Modelo detalhado para Matrícula.

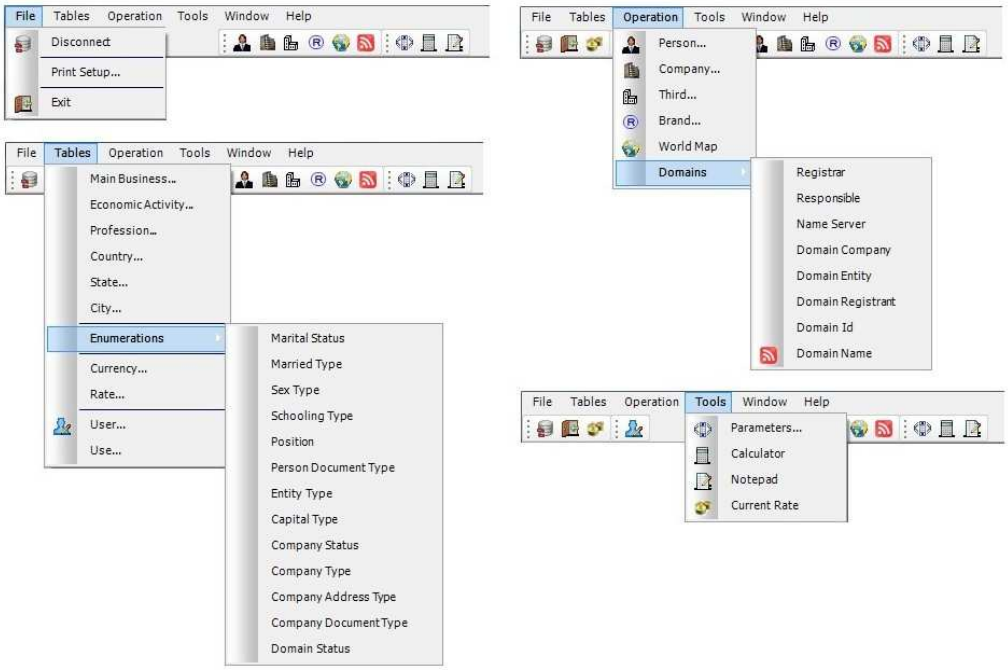


4.3.2 Exemplo de um sistema Desktop

Como estudo de caso em sistemas reais, um sistema de informação no padrão *Graphics User Interface* (GUI) foi analisado. O *Workbench* é do tipo *Multi Document Interface* (MDI), onde podemos ter diversas *Views* abertas simultaneamente. Este

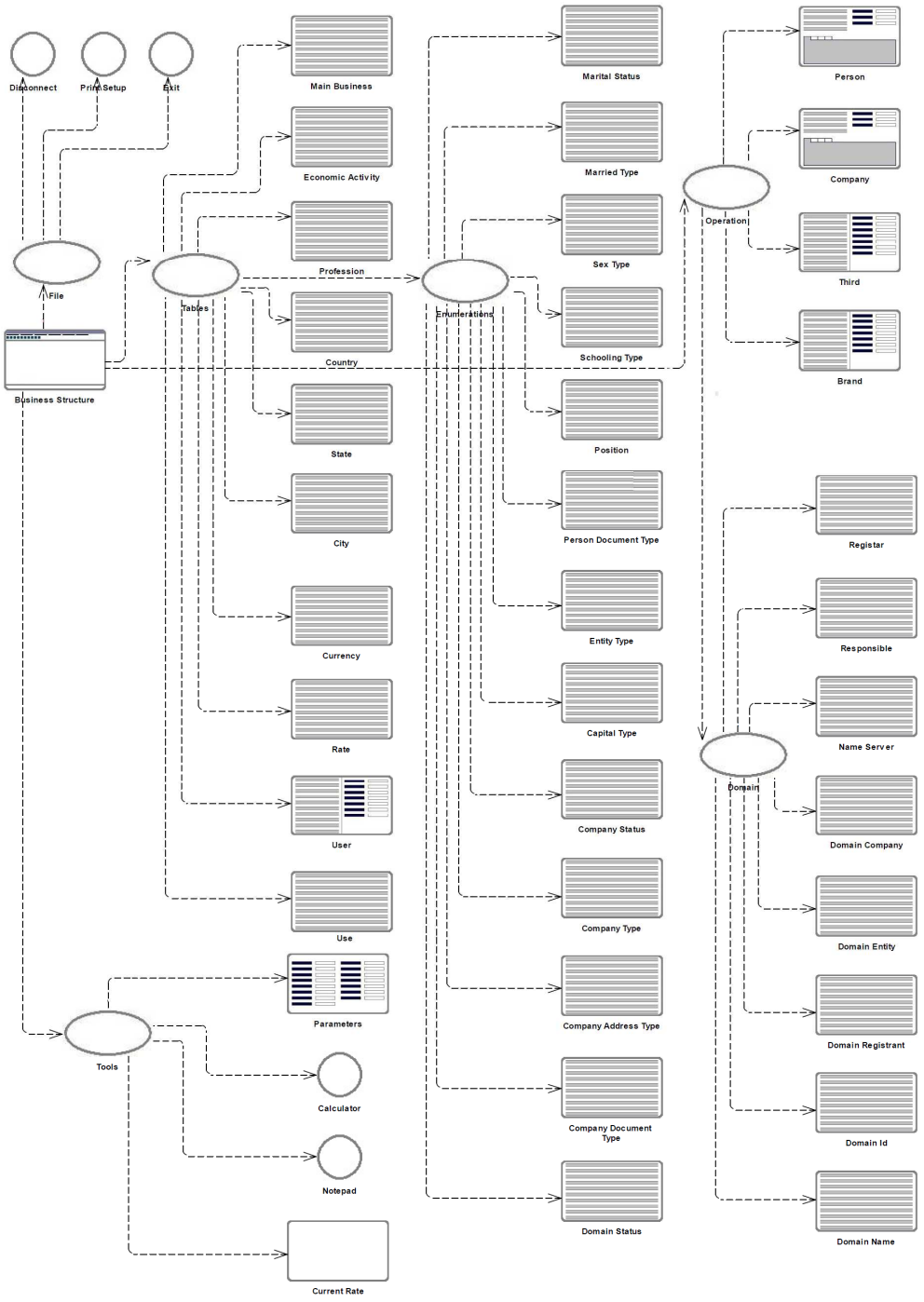
sistema contém diversas funcionalidades que foram implementadas em UIs. A navegação é feita através de *menus*. Ele é composto por diversas UIs bastante simples e algumas mais complexas. A Figura 4.14 mostra alguns dos principais *menus* de navegação existentes neste sistema.

Figura 4.14: *Menus* de um sistema de informação MDI.



Este sistema foi modelado com o metamodelo proposto e a Figura 4.15 apresenta o resultado desta modelagem, veja que está sendo apresentado o modelo completo e o objetivo não é permitir ler o texto, mas sim ver o modelo como um todo. Apenas uma *View* não pôde ser representada integralmente, pois nenhum dos padrões identificados permitia a sua representação. Numa situação real, a *View* poderia ser modificada para ser utilizado algum dos padrões disponíveis.

Figura 4.15: Modelo do sistema GUI.



4.3.3 Exemplo de um sistema Web

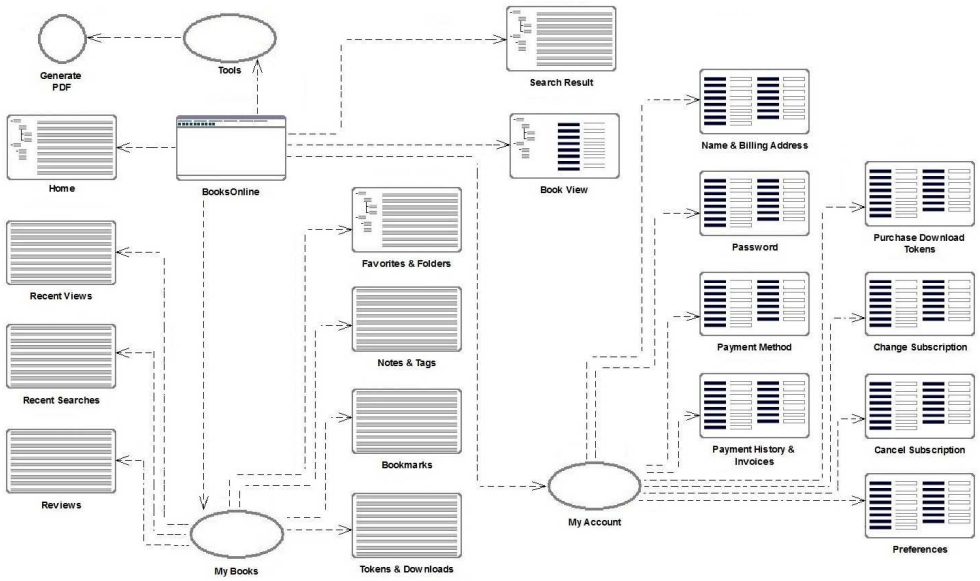
Neste estudo de caso, um sistema de informação no padrão *Web User Interface* (WUI) foi analisado. Este sistema foi escolhido ao acaso para realmente validar o

metamodelo. O *Workbench* é do tipo *web*. Este sistema é parte de um serviço de assinatura de livros online, onde o usuário pode pesquisar e consultar livros. A Figura 4.16 mostra uma UI, chamada de *My Books*, onde há uma lista das pastas do usuário e para cada pasta são mostrados os livros que estão nesta pasta.

Figura 4.16: Uma UI do Sistema Books Online.

Analisando todas as UIs deste sistema foi possível modelar completamente o sistema, tendo assim, um modelo em um nível bastante abstrato, mas com toda a informação sobre a estrutura de UIs do sistema analisado. Na Figura 4.17 é apresentado o modelo criado a partir deste sistema.

Figura 4.17: Modelo do sistema WUI.



4.3.4 Exemplo de um sistema Mobile

Neste estudo de caso foi analisada a parte de um sistema de informação coorporativo que tem a entrada de dados feita em um dispositivo móvel. Este "mini" sistema de informação é executado por um peão (empregado de fazenda responsável pela lida no campo) quando está em campo lidando com o gado. O sistema contém uma base de dados local que de tempos em tempos é sincronizado com o sistema coorporativo. Dessa forma o peão substitui a caderneta (bloco onde ele anotava as operações em campo) por um dispositivo móvel. As operações que o peão executa em campo são: contagem de animais, movimentação de animais entre potreiros, transferência entre lotes de animais, reclassificação de lotes, informações sobre um determinado potreiro e lançamento de mortes. Na Figura 4.18 é apresentado o *menu* do sistema no celular.

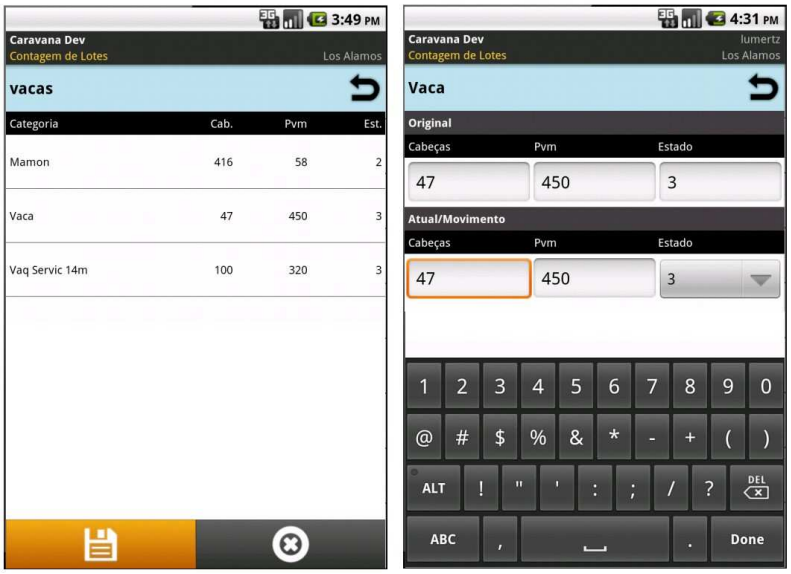
Figura 4.18: *Menu* do sistema no celular.

Simulando o uso do sistema, suponha que o peão selecionou a operação de contagem. Neste caso ele deve selecionar um setor da estância e na sequência um lote de animais, como apresentado na Figura 4.19.

Figura 4.19: Opção contagem, seleção do setor e do lote.

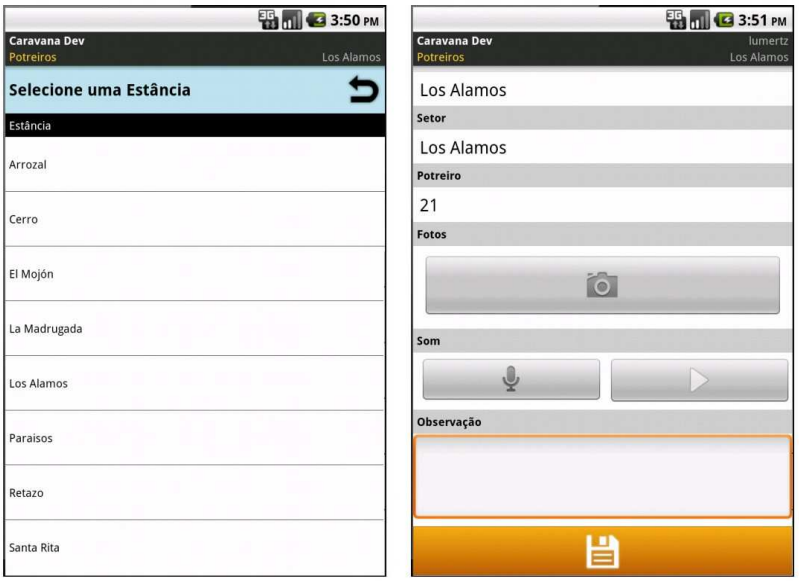
Após selecionar o lote, são mostradas todas as categorias de animais do lote e para cada categoria o peão de informar a quantidade de cabeças, o peso médio e o estado dos animais, como apresentado na Figura 4.20.

Figura 4.20: Opção contagem, informando cabeças, peso e estado.



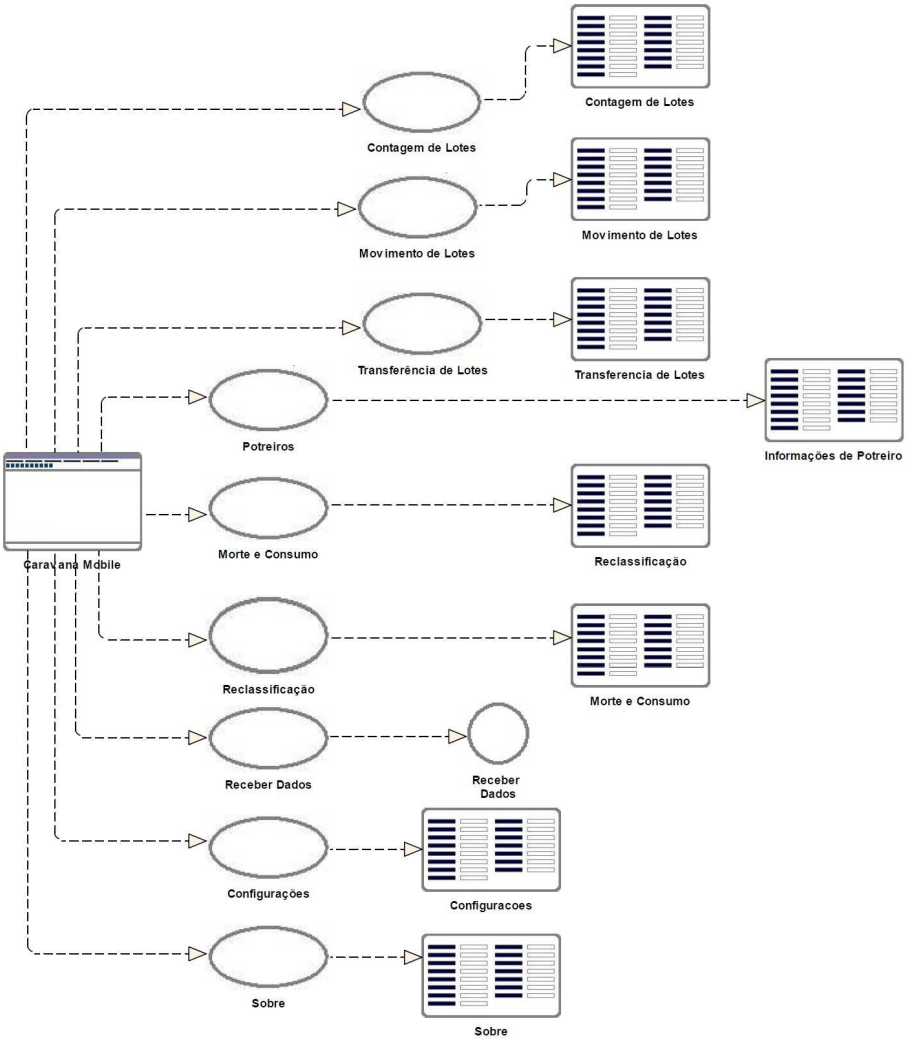
Outra opção disponível para o peão no sistema é adicionar informações sobre um determinado potreiro, pois quando ele está andando no campo pode identificar alguma situação que seja necessário algum procedimento, como conserto de cercas, semeadura, entre outros. Neste sistema ele pode adicionar uma gravação de som, fotos ou ainda escrever um comentário. Para isto ele faz a seleção da estância, do setor e do potreiro e lança as informações, como apresentado na Figura 4.21.

Figura 4.21: Opção potreiros, lançando informações.



Com a análise de todas as UIs deste sistema foi possível construir um modelo que contemple a totalidade dessas UIs. Na Figura 4.22 está apresentado o modelo completo no primeiro nível.

Figura 4.22: Modelo do sistema Caravana Mobile.



Com estes estudos de caso, pode-se concluir que o metamodelo proposto supre as necessidades de modelagem destes tipos de sistemas de informação, o que é feito com bastante simplicidade utilizando uma ferramenta de modelagem. A ferramenta utilizada foi o *Enterprise Architect* [EA, 2012], desenvolvida pela *Sparx Systems*. Esta escolha foi feita pelo conhecimento da ferramenta e também por ela permitir a extensão com o uso de *plug-ins*. Com este metamodelo é possível construir uma analisador que valide as modelagens de sistemas de informação e também aplicar técnicas de *Model Driven Development* (MDD) para gerar código das UIs seguindo os padrões previamente definidos.

Com os exemplos de modelos desenvolvidos anteriormente pode ser visto que uma representação visual, como a apresentada, auxilia no entendimento do sistema como um todo, permitindo ter uma boa ideia de tamanho inclusive. É importante salientar que usar uma linguagem visual aumenta a clareza dos modelos e facilita o entendimento.

5 ANÁLISES E EVOLUÇÃO DAS UIS DOS SISTEMAS

A teoria dos grafos fornece um ferramental importante para analisar grafos e neste trabalho são utilizadas algumas destas abordagens para exemplificar a análise dos modelos de sistemas criados a partir do metamodelo.

5.1 Ferramentas de análise de grafos

Para analisar os grafos podemos utilizar ferramentas que estão disponíveis na comunidade como projetos *open source*. Neste trabalho serão apresentadas três delas. Uma delas será utilizada para demonstrar as análises que podem ser feitas sobre o modelo de interface de um sistema modelado com o metamodelo proposto.

QuickGraph [QUICKGRAPH, 2013] é uma biblioteca contendo estruturas de grafos genéricas e algoritmos desenvolvidos em *.Net*. Estes algoritmos resolvem problemas clássicos da teoria dos grafos. As informações dos grafos podem ser serializadas através de *Xml*.

Outra biblioteca é a ***Yet Another Graph-Search Based Planning Library*** (YAGSBPL) [YAGSBPL, 2013]. Esta é uma biblioteca baseada em templates *C++* para pesquisa em grafos. Ela é bastante rápida e eficiente, tendo sido projetada para grafos de media e grande escala para pesquisa de caminhos.

Gephi [GEPHI, 2013] é uma plataforma interativa, para visualização e exploração de todos os tipos de redes, sistema complexo e grafo hierárquico e dinâmico. O *Gephi* possibilita a exploração e o entendimento visual de grafos. O usuário pode interagir com a representação, manipular estruturas, formas e cores para que sejam reveladas propriedades dos grafos. Tem como objetivo o auxílio na análise de dados e descoberta de padrões. Esta ferramenta será usada neste trabalho para a análise dos modelos de interface criados. Ela permite que os grafos sejam importados através de arquivos com a relação de nodos e arestas.

5.2 Ferramentas de análise de gramáticas de grafos

Existem diversas ferramentas de gramáticas de grafos e aqui são apresentadas as principais:

Attributed Graph Grammar (AGG) [RUNGE, 2012] [AGG, 2013] é uma ferramenta tradicional de transformação de grafos, que provê uma linguagem gráfica para definir transformações modelo para modelo. As transformações de grafo são definidas sobre os elementos do metamodelo e visualizadas com um *layout* genérico,

chamado sintaxe abstrata, onde os nodos são visualizados como retângulos e as arestas como setas direcionadas. *AGG* é uma linguagem visual baseada em regras suportando a abordagem algébrica para transformações de grafos. *AGG* ajuda na especificação e prototipação de aplicações que tenham dados complexos estruturados como grafos. *AGG* pode ser usado (implicitamente no "código") como um *engine* de propósito geral para transformação de grafos em aplicações Java empregando métodos de transformação de grafo.

Fujaba Tool Suite (Fujaba) [FUJABA, 2013] é uma ferramenta *open source* que provê suporte aos desenvolvedores para desenvolvimento de *software* baseado em modelos. Ela é fácil de usar, formal, gráfica e tem uma linguagem baseada em transformações de grafos. As transformações de modelo para modelo são especificadas em *plug-ins* com *Triple Graph Grammars* (TGG). Estes *plug-in*, chamados de *MoTE* e *MoRTEn* permitem especificar e executar as transformações entre os modelos. Usando uma especificação declarativa as regras de transformação são executadas incrementalmente, o que permite a sincronização dos modelos e também *round-trip engineering.*

Graph Rewrite Generator (GrGen) [GRGEN, 2013] é uma ferramenta *open source* que oferece uma linguagem declarativa para modelar grafos e transformações sobre grafos.

VIsual Automated model TRAnsformations (VIATRA) [VIATRA, 2013] é um *framework* para suportar transformações de modelo permitindo especificar, executar e validar as transformações. Ele é composto de um espaço de modelo que é usado para uniformizar as representação dos modelos e metamodelos, uma linguagem de transformação com facilidades declarativas e imperativas baseadas em transformação de grafos e máquinas de estado abstratas. Por fim, um *engine* para executar as transformações.

Groove [GROOVE, 2013] é um projeto centrado no uso de grafos simples para modelagem de sistemas orientados a objeto e transformações de grafos como base para transformações de modelo. A ferramenta inclui um editor para criar as regras de grafos, um simulador para computar as transformações de grafos, um gerador para explorar os espaços de estado e um *model checker.*

Foi feito um trabalho de comparação entre três linguagens de transformações de modelos [OLSEN, 2009], a saber: *Concrete syntax-based graph transformation* (CGT) que é uma nova linguagem de transformação de modelos, *Attributed Graph Grammar* (AGG) que representa a tradicional linguagem de transformação de grafos e *Atlas Transformation Language* (ATL) que representa a linguagem de transformação de modelos. As conclusões foram as seguintes:

- Em termos de tamanho da solução, ou seja, o código necessário para resolver uma mesma transformação em *CGT* teve a solução expressa em meia página, *AGG* em uma página inteira e *ATL* em 4,5 páginas.
- A linguagem *CGT*, de acordo com Olsen et al. [OLSEN, 2009], parece ser mais intuitiva que a *AGG* e *ATL*, sendo mais fácil para que um modelador das transformações defina as regras de transformação.

Foi feita uma comparação entre as ferramentas *AGG* e *EMF Tiger* (*Eclipse Modeling Framework*) [BIERMANN, 2009] na qual são verificadas as facilidades de ambas as ferramentas e avaliadas suas habilidades para criar um modelo. A conclusão

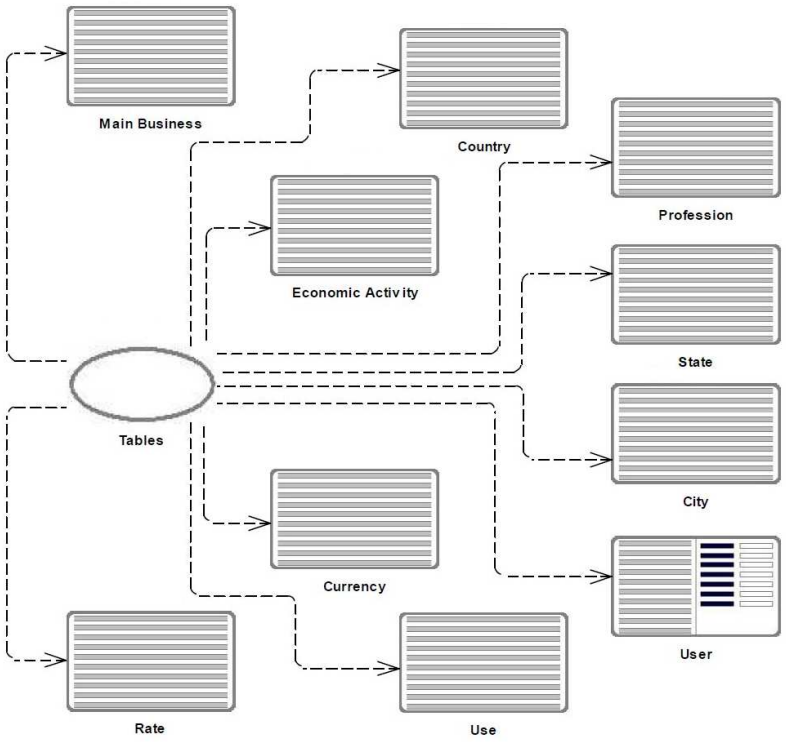
foi que embora as características de ambas as ferramentas pareçam bastante semelhantes à primeira vista (modelagem visual de regras de transformação, metamodelo, aplicação de regras não determinísticas), a diferença chave dos conceitos levam a seguinte recomendação para quando for usar uma das ferramentas: a *EMF Tiger* têm habilidade para trabalhar com metamodelos *EMF* e o forte do *AGG* é na análise estática de propriedades dos sistemas de transformação de grafos.

Considerando as ferramentas identificadas acima e os trabalhos comparativos, a ferramenta *AGG* pode ser utilizada neste trabalho como ferramenta de transformação de grafos.

5.3 Melhorias do modelo

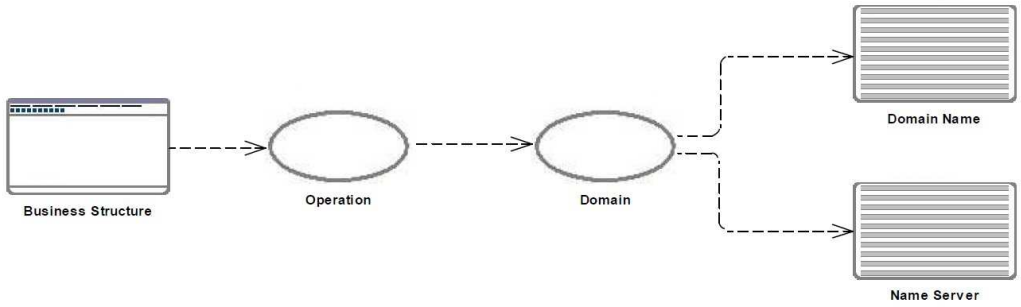
Por ter um modelo de UI de um sistema de informação em um grafo, podem ser feitas análises sobre estes grafos para propor melhorias na própria estrutura do sistema modelado, tendo assim um mecanismo formal de análise. Por exemplo, ao analisar o grau e a profundidade deste grafo, poderão ser sugeridas mudanças que ocasionam melhorias na interface. O grau de um vértice representa o número de arestas que incidem (*indegree*) em um determinado vértice ou que partem de um vértice (*outdegree*). Considerando o modelo de um sistema, como visto anteriormente, o grau de um vértice, com arestas partindo deste vértice, indica quantos itens de *menu* tem a opção representada pelo vértice. Considerando as arestas que partem (*outdegree*), indica quantas opções distintas que podem ser invocadas. Do ponto de vista das UIs, se tivermos um número muito grande de itens poderá indicar a necessidade de uma reestruturação nos *menus* do sistema. Na Figura 5.1 está exemplificada uma modelagem onde a opção de *menu Tables* tem 10 arestas partindo dele, o que nos indica grau 10.

Figura 5.1: Itens de *Menu* com grau 10.



Um caminho é uma sequência de vértices tal que, de cada um dos vértices, existe uma aresta para o vértice seguinte. A profundidade de um caminho é o número de suas arestas. Este conceito, em um modelo de sistema indica quantas opções de *menu* o usuário necessita selecionar para chegar em uma funcionalidade do sistema. Do ponto de vista de UIs, se tivermos um número muito grande de opções poderá indicar que há necessidade de uma reestruturação nos *menus* do sistema. Na Figura 5.2 está apresentado um exemplo da modelagem com profundidade 3.

Figura 5.2: Modelagem de *Menu* com profundidade 3.



Devido ao modelo ter sido construído com grafos, um analisador pode ser incorporado na ferramenta de modelagem para fazer análises que possibilitem uma melhoria do modelo. Com as informações de grau e profundidade, o analista que está desenvolvendo o modelo, pode validar se há necessidade de reformular a estrutura do sistema, pois como visto anteriormente, um grau muito grande indica muitas opções em um único item o que pode ser necessário uma reestruturação e se tivermos uma profundidade muito grande indica que muitas opções devem ser selecionadas até chegar na funcionalidade que o usuário deseja, talvez sendo necessário uma reestruturação. Um analisador como este não faz parte do escopo deste trabalho.

5.4 Consistência do modelo

Como o sistema é modelado como um grafo tipado, é possível analisar este grafo e validar as propriedades definidas para ele fazendo uma consistência propriamente dita. Com esta consistência é possível garantir que o modelo esteja sintaticamente correto. Abaixo algumas situações possíveis que podem ser consistidas e o que elas identificam:

Opção de *menu* sem itens: Se tivermos um vértice do tipo *MenuGroup* que não tenha vizinhança com vértices de *Views*, *MenuAction* ou *MenuFlag* ele será uma opção vazia.

***View* sem nome**: Se tivermos uma *View* onde o atributo *ViewName* estiver nulo indica que esta *View* não tem nome, ou seja, é uma opção no *menu* sem o nome de opção.

Composição de uma *View*: Cada *View* existente pode ser composta por elementos primitivos que no grafo são representados por vértices. Uma determinada *View* deve ter

ligação com determinados elementos primitivos. Para validar a consistência do modelo, no grafo, pode ser analisado se este vértice da *View* tem vizinhança com os elementos primitivos, por exemplo, um vértice do tipo *ViewListForm* deve ter ligação com um vértice do tipo *List* e um vértice do tipo *Form* e opcionalmente com um vértice do tipo *Filter*. Como esta validação tem outras 24 somente para *Views*.

Como o modelo é construí

5.5 Análises sobre os modelos

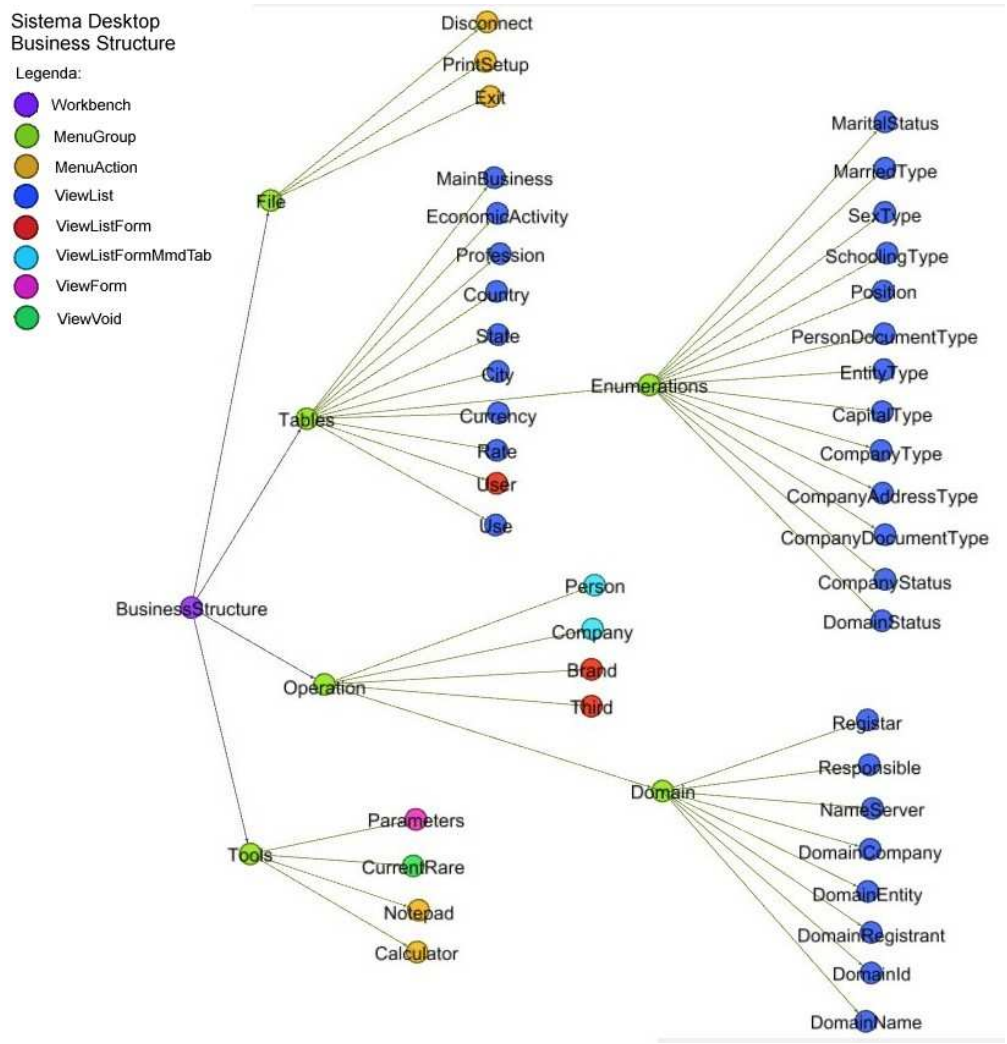
Como visto anteriormente, as UIs raramente têm o mesmo nível da especificação e modelagem que os dados e funcionalidades, ou seja, existe uma lacuna no que tange modelos de UI e dessa forma poucas análises são disponíveis sobre as UIs de um sistema de informação. Usando o metamodelo proposto neste trabalho, um bom conjunto de informações sobre as UIs do sistema modelado é criado e dessa forma é possível executar diversas análises. Um exemplo é a distribuição do tipo de padrão de UI utilizado o que pode ser utilizado para classificar o sistema. Esta distribuição é facilmente encontrada ao percorrer o grafo e identificar os vértices tipados como *Views* que definem as UIs do sistema. Se estas *Views* forem sumarizadas elas identificam a quantidade de IUs e cada tipo que o sistema contempla. Usando esta mesma estratégia pode ser sumarizada a quantidade de elementos primitivos utilizados no sistema. Como exemplo, na Tabela 5.1 é apresentada a quantidade de UIs do sistema GUI que foi usado anteriormente na validação do metamodelo. Este sistema é um exemplo de sistema que tem as UIs voltadas para uso em desktop e o seu objetivo é cadastrar pessoas, empresas, marcas e domínios e os relacionamentos entre elas, podendo inclusive guardar cópias de todos os documentos em pdf ou jpg.

Tabela 5.1: Estatística das UIs do Sistema GUI

Tipo UI	*Qtd*
MenuGroup	6
MenuAction	5
ViewForm	1
ViewList	30
ViewListForm	3
ViewListFormMmdTab	2
ViewVoid	1

Podem ser utilizadas ferramentas de análises de grafos. Exportando os dados do grafo de um modelo, por exemplo, o da Figura 4.15 que representa o sistema GUI analisado, e importando em uma ferramenta de análise de grafos, como o *Gephi*, pode ser visualizado o grafo que representa o sistema e fazer diversas análises.

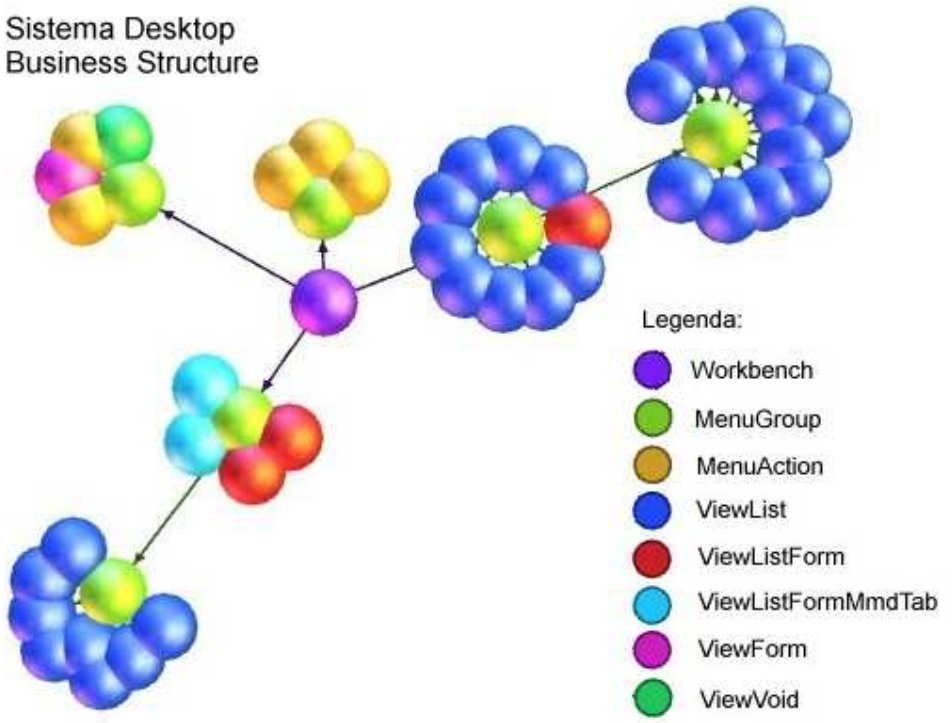
Figura 5.3: Grafo do modelo de um sistema GUI.



Na Figura 5.3 é apresentada a representação do grafo, onde cada tipo de *View* está categorizado por uma cor. Na Figura 5.4 é apresentado o mesmo grafo, porém com uma visualização por nodo onde pode ser identificada a densidade de cada grupo de opções do sistema. Para uma melhor análise, os padrões de UIs poderiam ter pesos de acordo com a complexidade de cada opção e neste grafo, cada nodo teria tamanho representado por este peso. No exemplo em questão, todos os nodos foram considerados com o mesmo peso. No modelo da Figura 5.4 é possível ver uma concentração do uso de *ViewList*, representado pelas esferas azuis e como o *ViewList* é uma construção CRUD bastante simples isto pode indicar que esta parte do sistema será bastante simples de implementar.

Usando o *Gephi*, também é possível analisar o grau, a largura e a profundidade do grafo. Considerando o grafo de um modelo de UIs, o grau identifica o número de opções abaixo que uma determinada opção contém. A profundidade identifica o número de opções necessárias para chegar a uma determinada UI. A largura identifica o número de opções num mesmo nível.

Figura 5.4: Grafo com densidade do modelo de um sistema GUI.



5.6 Evolução do metamodelo

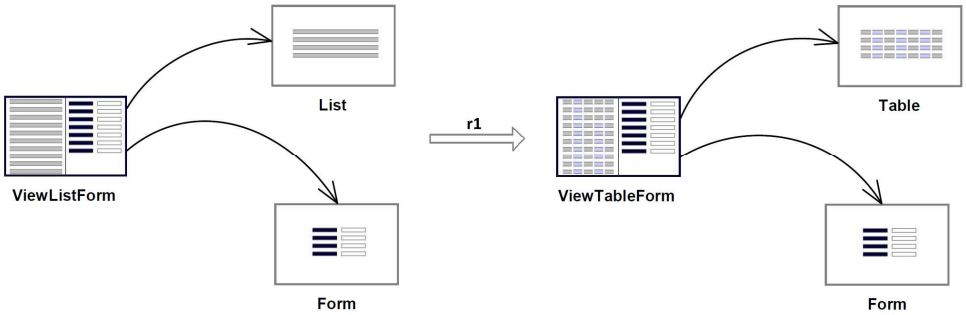
Com a atualização da tecnologia novos tipos de UIs podem ser necessários, ou também, por novos requisitos pode haver a necessidade de alteração nos padrões de UIs já existentes. Se os padrões suportados pelo metamodelo forem alterados, ou se tiver adição de novos padrões, os modelos anteriormente desenvolvidos não farão uso destas inovações. Dessa forma é importante que o metamodelo suporte um processo de evolução, onde cada instância de modelos que já tenham sido desenvolvidos possa ser atualizada. Como este metamodelo foi desenvolvido com grafos, podem ser definidas regras de transformação de grafos que contemplem esta evolução e serem aplicadas sobre as instâncias de modelos que já existam.

Para demonstrar esta capacidade, foi criado um exemplo onde houve a necessidade de melhorar um tipo de padrão de interface e é de interesse que todos os modelos passem a ter este novo padrão. No exemplo, o elemento primitivo *List* foi evoluído para o elemento hipotético *Table*, que para fins didáticos está sendo suposto que o elemento *Table* tem mais recursos do que um *List*. Dessa forma onde tiver um vértice do tipo *List* ele deve ser substituído por um *Table* e todas as *Views* que utilizam o *List* devem ser evoluídas também. Por exemplo, uma *ViewList* → *ViewTable*, uma *ViewListForm* → *ViewTableForm*, uma *ViewTreeList* → *ViewTreeTable* e assim sucessivamente para todas as *Views* que utilizam o *List*, onde → é a aplicação da regra. Na Figura 5.5 é

mostrada a exemplificação gráfica das mudanças de *List* para *Table* e na Figura 5.6 de *ViewListForm* para *ViewTableForm*.

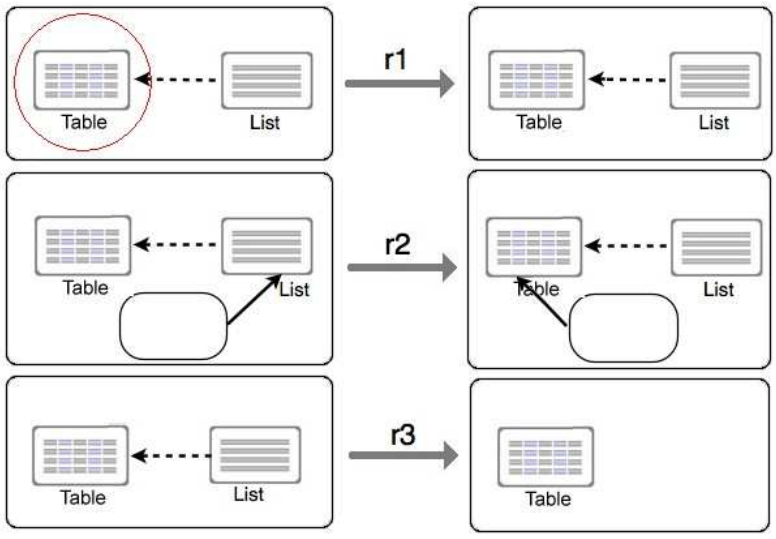
Figura 5.5: Regra r1: transformação de *List* em *Table*.

Figura 5.6: Aplicação dá regra r1: *ViewListForm* em *ViewTableForm*.



Utilizando uma ferramenta de transformação de grafos, devem ser criadas as regras de transformação, onde colocamos no lado esquerdo a ocorrência que deve ser encontrada e no lado direito a substituição. Na Figura 5.7 está exemplificado o detalhamento da regra. A regra *r1* pode ser aplicada para o vértice *List* onde o vértice *Table* ainda não tenha sido criado. A regra *r2* pode ser usada para substituir todas as arestas de vértices conectados ao vértice *List* para o vértice *Table*. Por fim, quando todas as arestas tiverem sido conectadas ao vértice *Table*, o vértice *List* pode ser removido.

Figura 5.7: Detalhamento da Regra.



Com este exemplo fica exemplificado que o um metamodelo construído com grafos possibilita que os modelos desenvolvidos sejam facilmente evoluídos.

6 CONCLUSÕES

Este trabalho apresentou a proposta de um metamodelo que permite modelar as interfaces do usuário para sistemas de informação. O objetivo do metamodelo é auxiliar na estruturação e desenvolvimento de um sistema para o nicho escolhido. Como pode ser percebido através dos exemplos e estudos de caso, existe uma grande clareza e simplicidade nos modelos, além de ter uma garantia da construção do modelo, devido a um metamodelo formal, que permite validações e análises sobre os modelos criados. Também este trabalho auxilia na padronização dos sistemas de informação, que será conseguida em grande parte do desenvolvimento das UIs, pois ele está centrado no reuso de padrões.

Como vantagem de usar um metamodelo pode ser citado que ele define as propriedades que o modelo pode ter, determinando restrições sobre os modelos a serem construídos. Este metamodelo em particular, por ter sido construído com grafos, tem a vantagem adicional de ter a evolução facilmente definida por transformações de grafos e esta característica não foi explorada em outros trabalhos relacionados.

6.1 Contribuições

As principais contribuições deste trabalho são as seguintes:

- Identificação de um conjunto de padrões de UIs existentes em sistemas de informação. Estes padrões podem ser utilizados como referência para o desenvolvimento de sistemas e no caso da identificação de algum novo padrão o conjunto pode ser estendido, de forma a ter documentado todos os padrões utilizados.
- O metamodelo permite o desenvolvimento de modelos por analistas em um ambiente de fabrica de *software*. Esta modelagem é feita com o uso de ferramentas CASE. Tais ferramentas têm poucos, ou nenhum, subsídios para a garantia sintática e semântica dos modelos construídos. Utilizando as análises e validações possíveis de serem feitas com o metamodelo será possível contribuir no preenchimento desta lacuna de modelagem das UIs.
- A exemplificação de como pode ser incorporado um processo de evolução de um metamodelo e aplicado aos modelos instanciados.

6.2 Comparação com os trabalhos relacionados

No Capítulo 3 foram apresentados diversos trabalhos relacionados e agora será apresentada uma comparação com este trabalho. Para isto foi desenvolvida uma tabela

relacionando diversas características e identificando qual trabalho dispõe desta característica. As características são:

UML: Trabalhos que de alguma forma usam UML. A maioria dos trabalhos usam UML para o modelo de domínio. Alguns usam ainda modelos de casos de uso e modelos de atores e criam um modelo de interface de usuário com UML.

Padrões: Trabalhos que usam padrões para associar à criação das UIs.

Geração de código: Trabalhos que produzem, por geração, o código concreto da UI para pelo menos uma plataforma.

Suporte à evolução: Trabalhos que tem suporte a evolução dos modelos.

Tabela 6.1: Comparação com trabalhos relacionados

Referência	*UML*	*Padrões*	*Geração*	*Evolução*
RAD				
Rosado da Cruz	X		X	
Mrack (MERLIN)	X		X	
Silva (XIS)	X		X	
Elkoutbi	X		X	
Molina	X	X	X	
Pastor	X	X	X	
Jia (Zoom)	X	X	X	
Limbourg (Tomato)	X			
Este Trabalho	X	X		X

Concluindo a avaliação deste trabalho com relação aos outros, é importante salientar que o uso de padrões é uma excelente estratégia, pois além de favorecer o reuso por construção, facilita o processo de geração de código. Praticamente todos os trabalhos utilizam a UML de alguma forma na representação dos modelos. Este trabalho teve preocupação em ter um processo evolutivo para os modelos, cujos exemplos estão na Seção 5.6.

6.3 Trabalhos futuros

Como identificado na Seção 6.3, este trabalho também têm a possibilidade de geração de código, utilizando transformadores específicos para as plataformas a serem utilizadas. Este trabalho abre caminho para novas pesquisas:

- O desenvolvimento de um analisador, que embasado no metamodelo, possa validar o modelo de um sistema. Este analisador pode ser desenvolvido como um plug-in para uma ferramenta de modelagem, permitindo que o modelo seja validado durante o seu desenvolvimento.

- Extensão do conjunto de elementos de UI para ser utilizado na modelagem, contemplando dessa forma outros tipos de sistemas.

- Desenvolvimento de transformadores de código para diferentes plataformas, utilizando o conceito de ilha de código para permitir *round trip*.

- Desenvolvimento de um mecanismo de evolução por transformação de grafos, que permita evoluir o metamodelo e ser aplicado em todas as instâncias de modelos de forma que estes modelos tenham contemplada a evolução do metamodelo.

- Refinamento do metamodelo incorporando a modelagem de outros controles de UI para os dados dentro dos elementos primitivos, onde serão pesquisadas as formas de apresentar estes dados.

- No refinamento dos elementos de dados pode ser criado um novo tipo que permita estabelecer uma ligação para uma UI criando dessa forma a navegação entre UIs, o que não foi contemplado neste trabalho.

- Existe uma fronteira entre a Engenharia de Software (ES) e Interação Humano Computador (IHC). Por um lado a ES define regras e procedimentos para a construção de UIs, por outro, a IHC tem um conjunto de normas que uma UI deve seguir. Usar o metamodelo proposto neste trabalho, estendido com elementos de dados para navegação entre as UIs e um gerador de código habilitado com *round trip*, e também um processo de evolução dos modelos pode possibilitar uma integração na fronteira entre estas duas importantes áreas, contemplando as propriedades de processo de construção de UI da ES e as propriedades das UIs pela IHC. Para a implementação do round trip uma boa alternativa para manter os modelos válidos, mesmo que os programadores adicionem códigos a UI é usar o conceito de ilhas de código, onde na geração são colocadas marcas (*tags*) onde o programador pode adicionar código complementar.

Como conclusão final, o metamodelo proposto, será utilizado para construção de ferramentas para auxiliar uma linha de produção de *software*, onde os sistemas desenvolvidos têm muitas das funcionalidades com aparências e comportamentos que se repetem e que podem ser implementadas com os padrões identificados neste trabalho, possibilitando assim, ganhos em qualidade e produtividade.

REFERÊNCIAS

AGG. **AGG Homepage**. <http://user.cs.tu-berlin.de/~gragra/agg/index.html>. Acesso em: Abril de 2013.

ANDROID. **Android Homepage**. <http://www.android.com>. Acesso em: Abril de 2014.

APPLE. **Mac OS X Human Interface Guidelines: User Experience**. Apple Inc. 2011.

APPLE-IOS. **Apple IOS Homepage**. <http://www.apple.com/ios>. Acesso em: Abril de 2014.

APPLE-OSX. **Apple OSX Homepage**. <http://www.apple.com/osx>. Acesso em: Abril de 2014.

BIERMANN, E.; ERMEL, C.; TAENTZER, G. Precise semantics of EMF model transformations by graph transformation. In: **Model driven engineering languages and systems**. [S.1.: s.n.], 2008. p. 53-67.

BIERMANN, E.; ERMEL, C.; TAENTZER, G. **Introduction to AGG and EMF Tiger by Modeling a Conference Scheduling System**. 2009.

BONDY, J.; MURTY, U. **Graph Theory**. Graduate Texts in Mathematics. Springer, 2006.

DA SILVA, P. P.; PATON, N. W. User interface modeling in UMLi. In: **IEEE Software**, [S.1.: s.n.], v.20.n.4, 2003. p. 62-69.

DIESTEL, R. **Graph Theory**. 2nd Ed. Springer. NewYork. 2000.

EA. **Enterprise Architect from Sparx Systems**. <http://sparxsystems.com.au>. Acesso em: Dezembro de 2012.

EHRIG, H.; ENGELS, G.; KREOWSKI, H-J.; e ROZEMBERG, G. **Handbook of Graph Grammars and Computing by Graph Transformation, Applications, Languages and Tools**. Vol. 2, World Scientific, Singapore, 1999.

EHRIG, H.; Ehrig, K.; PRANGE, U.; TAENTZER, G. **Fundamentals of Algebraic Graph Transformation**. EATCS Monographs in Theoretical Computer Science. Springer Verlag, 2006.

EHRIG, H; LAMBERS, L; TAENTZER, G. Sufficient criteria for applicability and non-applicability of rule sequences. **Journal of Electronic Communication of The European Association of Software Science and Technology**, [S.1.: s.n.], v. 10, 2008.

ELKOUTBI, M.; KHRISS, I.; KELLER, R. **Automated prototyping of user interfaces based on UML scenarios**. Journal of Automated Software Engineering 13(1), 5–40 2006.

EULER, I.; **The Königsberg Bridge problem**. Petersburg Academy. 1736.

FUJABA. **The Fujaba Project**. <site: http://www.fujaba.de/> Acesso em: Abril de 2013.

GALITZ, W. O. **The essential guide to user interface design:** an introduction to GUI design principles and techniques. [S.1.: s.n.], 2007.

GEPHI. **The Open Graph Viz Platform**. <http://gephi.org/>. Acesso em: Agosto de 2013.

GRGEN. **Graph Rewrite Generator**. <http://www.info.uni-karlsruhe.de/software/grgen> Acesso em: Abril de 2013.

GROOVE. **Simple Graphs for Modelling**. <http://groove.cs.utwente.nl/> Acesso em: Agosto de 2013.

GUTTMAN, M.; PARODI, J. **Real-life MDA: solving business problems with model driven architecture**, Morgan Kaufmann Publishers, Elsevier Inc. 2007.

HECKEL, R. Graph transformation in a nutshell. **Journal Electronic Notes in Theoretical Computer Science**, [S.1..: s.n.], v. 148, n. 1, 2006. p. 187-198.

JAVAFX. **Javafx Scenebuilder da Oracle**. <http://docs.oracle.com/javafx/>. Acesso em: Dezembro de 2012.

JIA, X.; STEELE, A.; QIN, L.; LIU, H.; JONES, C. Executable visual software modeling the ZOOM approach. **Software Quality Control**, [S.1.: s.n.], v. 15, n.1., 2007. p. 27–51.

JOHNSON, J. **Designing with the mind in mind:** simple guide to understanding user interface design rules. [S.1.]: Morgan Kaufmann. 2010.

LIMBOURG, Q.; VANDERDONCKT, J. **Transformational Development of User Interfaces with Graph Transformations**. Springer Netherlands. Computer-Aided Design of User Interfaces. Book Chapter. 107-120, 2005.

MOLINA, P. J.; MELIA, S; PASTOR, O. **User Interface Conceptual Patterns. Design, Specification, and Verification of Interactive Systems**. DSV-IS, pp. 159-172, 2002.

MOLINA, P. J. **Especificación de interfaz de usuario: De los requisitos a la generación automática**. Tesis Doctoral in Universidade Politecnica de Valencia, Departamento de Sistemas Informáticos y Computación. 2003.

MRACK, M. **Geração Automática e Assistida de Interfaces de Usuário**. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre, 2009.

MYERS, B. A.; ROSSON, M. B. **Survey on User Interface Programming**. In: Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, pp.95-202, 1992.

MYERS, B. A. User interface software tools. **ACM Transactions on Computer-Human Interaction**, [S.1.]: ACM, v. 2, n. 1, 1995. p. 64-103.

MYERS, B. A.; NICHOLS, J.; CHAU, D. H. Demonstrating the viability of automatically generated user interfaces. In: COMPUTER HUMAN INTERACTION - CHI, 2007. **Proceedings**..., [S.1.: s.n.], 2007. p. 1283-1292.

NETTO, P. O. **Grafos: Teoria, Modelos, Algoritmos**. 4. ed. São Paulo. Editora Blucher. 2006.

OLSEN, D.R.K.; SCOTT, R. The future of user interface design tools. In: CHI EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2005. **Proceedings**..., New York: ACM, 2005. p. 2134-2135.

OLSEN, G.; GRONMO, R.; MOLLER-PEDERSEN, B.; Comparison of three model transformation. In: EUROPEAN CONFERENCE ON MODEL DRIVEN ARCHITECTURE FOUNDATIONS AND APPLICATIONS, 2009. **Proceedings**..., [S.1.: s.n.], 2009. p. 2-17.

OMG: **Technical Guide to Model Driven Architecture**. In: The MDA Guide v1.0.1, June 12, 2003. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>. Acesso em: Março de 2012.

PASTOR, O. Generating user interfaces from conceptual models a model transformation based approach. In: COMPUTER-AIDED DESIGN OF USER INTERFACES - CADUI, 2006. **Proceedings**..., [S.1.: s.n.], 2006. p. 1-14.

PASTOR, O. et al. Conceptual user interface patterns for object-oriented conceptual models. In: IFIP CONFERENCE ON HUMAN-COMPUTER INTERACTION. **Proceedings**..., [S.1.: s.n.], 2007. p. 36-50.

PETRASCH, R. Model based user interface development with HCI patterns: variatio delectat. INTERNATIONAL WORKSHOP ON PATTERN-DRIVEN ENGINEERING OF INTERACTIVE COMPUTING SYSTEMS, 2010. **Proceedings**... New York: ACM, 2010. p, 10-11.

POWERBUILDER. **The Rapid Enterprise Application Development Tool from Sybase**. <http://www.sybase.com/powerbuilder12>. Acesso em: Dezembro de 2012.

PUERTA, A.; EISENSTEIN, J. Towards a general computational framework for model-based interface development systems model-based interfaces. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES - IUI, 1999. **Proceedings**... New York: ACM, 1999. p. 171-178.

QUICKGRAPH. **QuickGraph**. <http://quickgraph.codeplex.com>. Acesso em: Agosto de 2013.

RIBEIRO, L.; KORFF, M. **Métodos formais para Especificação: Gramática de Grafos**. Pelotas: UFPel, 1997.

RIBEIRO, L.; DA COSTA, S.A.; FOSS, L. Specification patterns for properties over reachable states of graph grammars. SIMPOSIO BRASILEIRO DE METODOS FORMAIS. 2012. **Anais**... [S.1.: s.n.], 2012. p. 83-98.

ROSADO DA CRUZ, A. M.; FARIA, J. P. **A Metamodel-Based Approach for Automatic User Interface Generation**. In: Model Driven Engineering Languages and Systems. p. 256-270, 2010.

ROZENBERG, G. Handbook of graph grammars and computing by graph transformations. In: **Graph Grammars Workshops**, [S.1.: s.n.], 1997. Volume 1.

RUNGE, O.; ERMEL, C.; TAENTZER, G. **AGG 2.0 – New Features for Specifying and Analyzing Algebraic Graph Transformations**. In: Applications of Graph Transformations with Industrial Relevance Lecture Notes in Computer Science, Volume 7233, pp 81-88, 2012.

SCHLINGBAUM, E.; ELWERT, T. Automatic user interface generation from declarative models. In: COMPUTER AIDED DESIGN OF USER INTERFACES - CADUIT, 1996. **Proceedings**..., Namur: s.n. 1996. p. 3-18.

SCHLINGBAUM, E. **Individual User Interfaces and Model-based User Interface Software Tools**. GIT-GVU-96-28, 1997.

SCHMIDT, D.C. Model-driven engineering. **IEEE Computer**, [S.1.]: IEEE, v.39, n. 2, 2006. p. 25–31.

SILVA, A.R. et al. XIS - UML profile for eXtreme modeling interactive systems. In: INTERNATIONAL WORKSHOP ON MODEL-BASED METHODOLOGIES FOR PERVASIVE AND EMBEDDED SOFTWARE - MOMPES, 2007. **Proceedings**... Los Alamitos, IEEE Computer Society, 2007.

TIDWELL, J. **Designing Interfaces**. O'Reilly. 2011.

UML. **Introduction to OMG's Unified Modeling Language**. <http://www.omg.org/gettingstarted/what_is_uml.htm>. Acesso em: Dezembro de 2012.

VS. **Visual Studio from Microsoft**. State-of-the-art toolset for enterprise development. <http://www.microsoft.com/visualstudio/eng/products/visual-studio-overview>. Acesso em: Dezembro de 2012.

VIATRA. **VIsual Automated model TRAnsformations**. <http://www.eclipse.org/viatra2> Acesso em: Abril de 2013.

WINDOWS8. **Windows 8 from Microsoft**. <http://windows.microsoft.com/pt-br/windows-8> Acesso em: Abril de 2014.

YAGSBPL. **Yet Another Graph-Search Based Planning Library**. <http://code.google.com/p/yagsbpl>. Acesso em: Agosto de 2013.

APÊNDICE A ANÁLISE DAS UIS DE SISTEMAS DE SOFTWARE

Para o desenvolvimento deste trabalho foi selecionado um conjunto de sistemas de informação reais para diversas áreas de negócio como hotéis, fábricas, fazendas, financeiro, contabilidade, entre outros, totalizando 19 sistemas. Eles foram analisados completamente e foram selecionados devido à facilidade de acesso aos mesmos. Acreditamos que mesmo a amostra não tendo sido feita de forma exaustiva, ela permite uma boa análise. A análise dos sistemas foi feita do ponto de vista da navegação, dos comportamentos e das aparências das UIs. Os sistemas que foram analisados têm *interfaces* para *web* e para *desktop* e totalizam 4 para *web* e 15 para *desktop*. Estas UIs serão chamadas de *Web User Interfaces* (WUI) e *Graphics User Interfaces* (GUI), respectivamente. Nesta análise, foi possível concluir que existem diversos padrões nestas UIs. Outros sistemas, pesquisados principalmente na Internet, foram analisados parcialmente com o objetivo de validar a aparência dos padrões de UIs encontrados na análise e, dessa forma, validar a amostra.

Sistemas Analisados

Os sistemas analisados são reais e automatizam diferentes áreas de negócio de diferentes empresas. Estes sistemas foram desenvolvidos em diferentes tecnologias, como *Powerbuilder*, Visual Basic, *Cold Fusion* e *Java* e para diferentes plataformas como *Desktop* e *Web*. Na tabela A.1 estão sumarizados todos os sistemas analisados que rodam em ambiente *web*. Estes sistemas tem a característica de possuírem os acessos via navegador *web* e manter os dados em uma base de dados centralizada.

Na tabela A.2 estão sumarizados todos os sistemas analisados que rodam em ambiente *desktop*. Estes sistemas são instalados para cada usuário e possuem bases de dados centralizadas onde os dados são armazenados.

Tabela A.1: Sistemas *Web* Analisados.

Nome	*Área*	*Tecnologia*
Reserva Online	Hotel	php
Application	Atividades de pessoal	Cold Fusion
Material System	Estoque de materiais	Cold Fusion
Vendas	Vendas	Cold Fusion

Tabela A.2: Sistemas *Desktop* Analisados.

Nome	*Área*	*Tecnologia*
Mercury	Fábrica de calçados	Java
JExport	Faturamento	Java
JCF	Financeiro	Java
Controle Financeiro	Financeiro	Powerbuilder
Global	Pedidos de calçados	Powerbuilder
Fox	Pedidos de calçados	Powerbuilder
Gas	Controle de acessórios	Powerbuilder
Business Structure	Administração	Powerbuilder
Export	Faturamento	Powerbuilder
Financial	Financeira	Powerbuilder
Sistema de Pecuária	Pecuária	Powerbuilder
Trading System	Financeira	Powerbuilder
Control System	Alocação de pessoas	Powerbuilder
Customer Invoice	Faturamento	Powerbuilder
Contabilidade	Contabilidade	Visual Basic

Bancada de Trabalho

A bancada de trabalho do usuário é a partir de onde o sistema é iniciado e foram identificados os tipos existentes nos sistemas em análise. Esta bancada de trabalho, que é a área completa do sistema e apresentada quando ele é carregado e é a partir de onde o usuário realmente executa as funcionalidades do sistema de *software*. A bancada de trabalho será chamada de *Workbench* e os tipos encontrados foram:

- Painéis Móveis (*Movable Panels*);
- MDI (*Multi Document Interface*);

- *Web-based applications*

A Figura A.1 apresenta um exemplo de *Workbench* com Painéis Móveis (note que o objetivo da imagem é mostrar o *Workbench* e não seu conteúdo propriamente dito). A Figura A.2 apresenta o *Workbench* com MDI. Ambos são dos sistemas reais que estão sendo investigados.

Figura A.1: Exemplo de *Workbench* com Painéis Móveis.

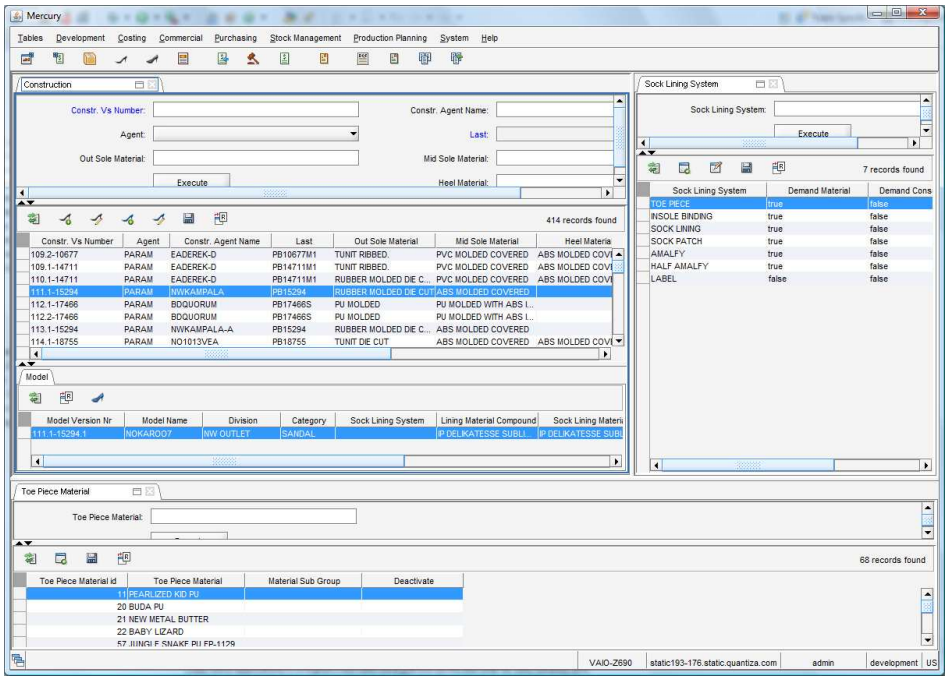
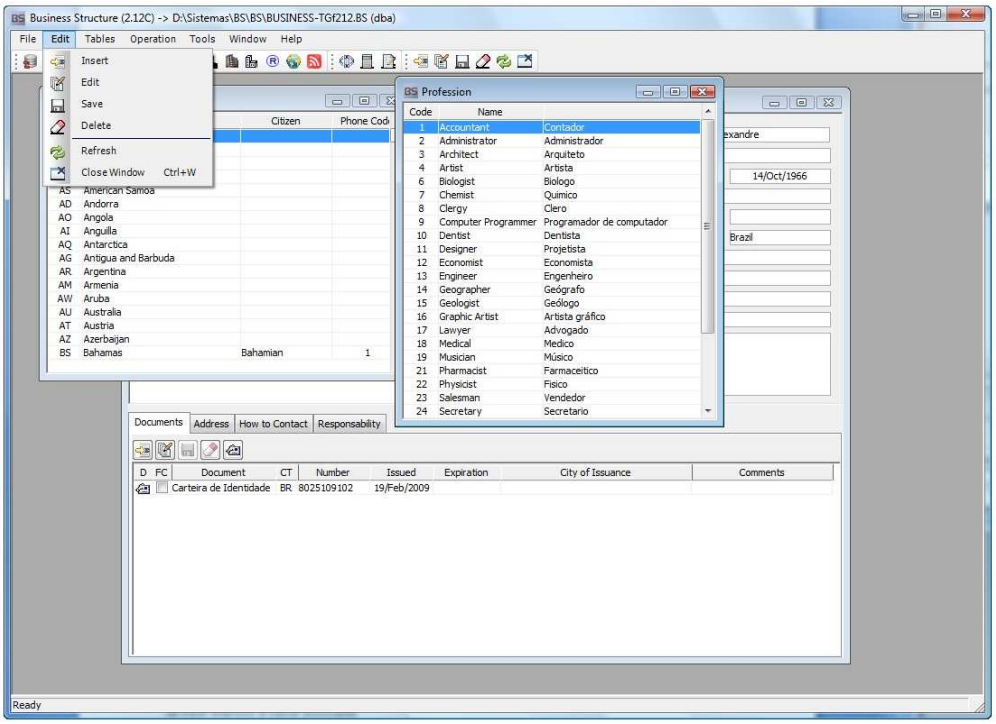
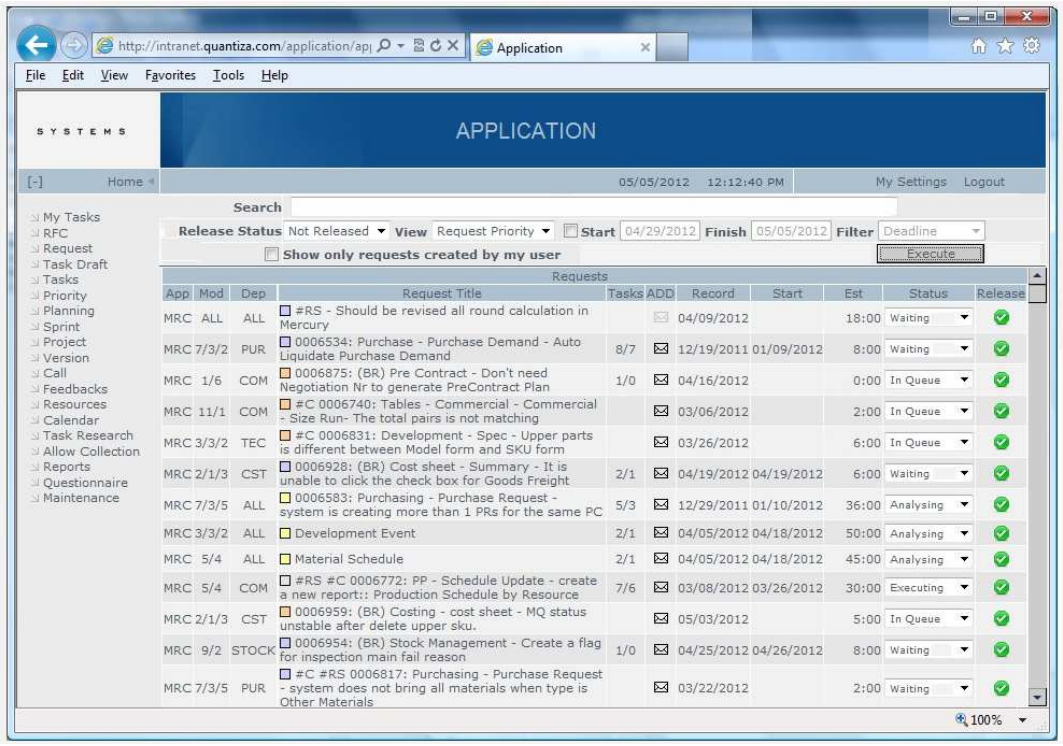


Figura A.2: Exemplo de *Workbench* com MDI.



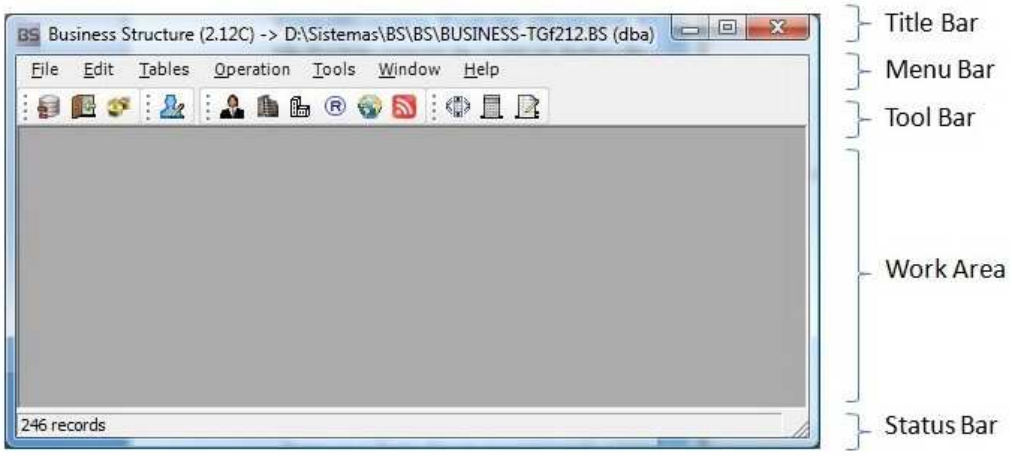
A Figura A.3 apresenta o *Workbench* de um sistema *Web*.

Figura A.3: Exemplo de *Workbench* com *Web*.



Tanto num *Workbench* com *Movable Panels*, como em um MDI, podem ser identificadas algumas áreas padrão: uma área de título (*Title Bar*), uma área de *menus* (*Menu Bar*), uma barra de ferramentas (*Tool Bar*), uma área de trabalho (*Work Area*) e uma área de mensagens (*Status Bar*). A Figura A.4 apresenta onde estão estas áreas dentro do *Workbench* (note que a imagem tem como objetivo apenas mostrar as áreas e não o conteúdo do texto).

Figura A.4: Áreas de um *Workbench*.



Após isto foram identificados os diversos tipos de UIs, onde são implementadas as funcionalidades do *software*, que foram chamadas de *Views*. Para cada *View* foram identificados os tipos básicos de apresentação e manipulação de dados. Também foram analisados os padrões para invocar as *Views*. No padrão MDI as *Views* aparecem sobrepostas e fica totalmente visível a que está com foco. Em um *Workbench* com *Movable Panels* as *Views* ficam em subdivisões da *Work Area*.

Um padrão de *Workbench* é composto por: uma *Title Bar* que é a área de título do sistema e normalmente contém informações como o ícone do sistema, o título, e pode conter outras informações adicionais; uma *Menu Bar* é a área dos *menus* de navegação do sistema; uma *Tool Bar* contém os ícones de atalho relativos a determinadas opções do *menu*; uma *Work Area* é a região onde ficam as interfaces do usuário que contém as funcionalidades, aqui chamadas de *Views*; e uma *Status Bar* é uma área normalmente usada para dar algumas notificações sobre o uso do sistema. Através da *Menu Bar* ou da *Tool Bar* o usuário pode selecionar funcionalidades que serão apresentadas através de *Views*.

Views dos Sistemas

As *Views* são as interfaces de usuário que contém as funcionalidades do *software*. Normalmente um sistema tem diversas destas *Views* e elas têm aparências e comportamentos diferentes. Na Figura A.5 é apresentado um exemplo de uma *View* onde existe um filtro que permite selecionar informações a serem consultadas e uma lista que mostra o resultado do filtro.

Figura A.5: Exemplo de *View* com filtro e lista.

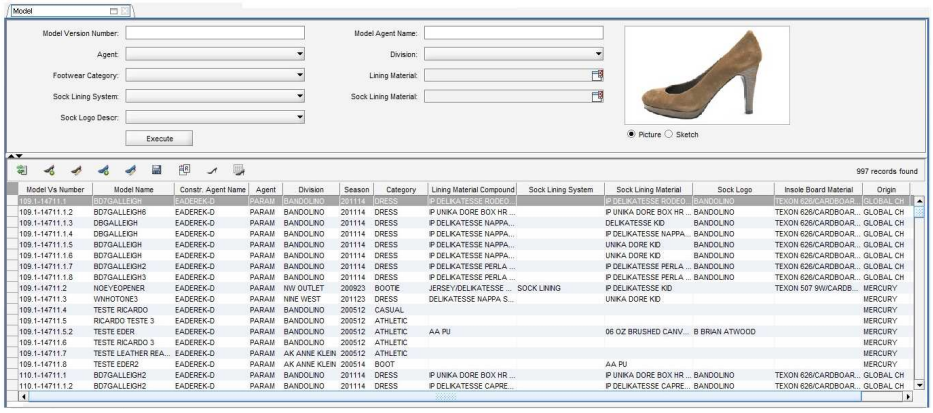
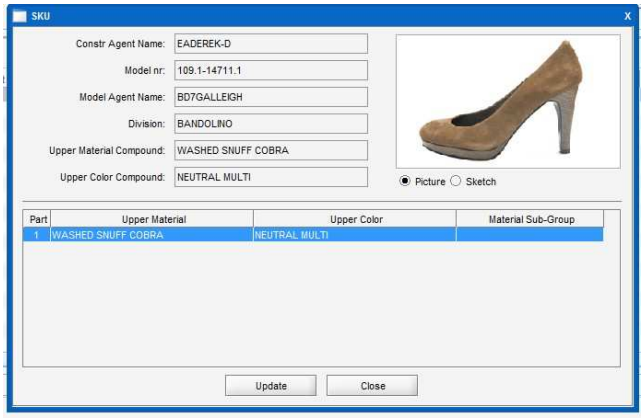
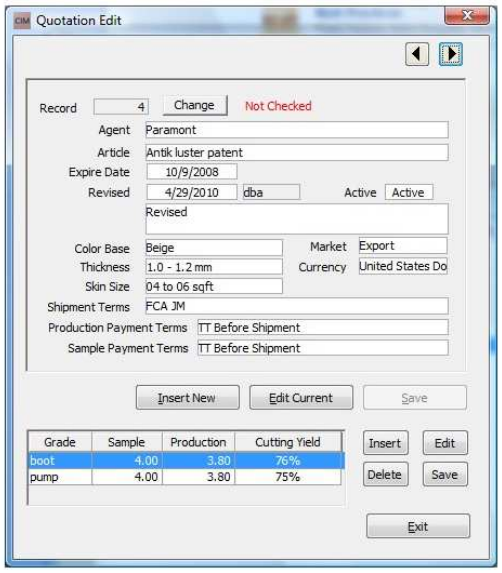


Figura A.6: Exemplo de *View* com formulário e lista.



A Figura A.6 contém um formulário onde podem ser editados alguns dados e uma lista mostrando informações vinculadas aos dados deste formulário. A Figura A.7 mostra também um formulário e uma lista, mas tem diversas funcionalidades associadas ao formulário e também à lista.

Figura A.7: Outro exemplo de *View* com formulário e lista.



Na Figura A.8 é apresentada uma *View* que tem quatro formulários, três listas e uma área com ações disponíveis para a *View*. Esta é uma *View* bastante complexa e que tem diversos comportamentos. Este é um exemplo que não é muito comum nos sistemas investigados e ele é bastante complexo. A grande maioria tem comportamentos mais simples. Na Figura A.9 é apresentada outra *View* que tem uma árvore e uma lista associada a cada elemento da árvore.

Figura A.8: Exemplo de *View* com diversos formulários e listas.

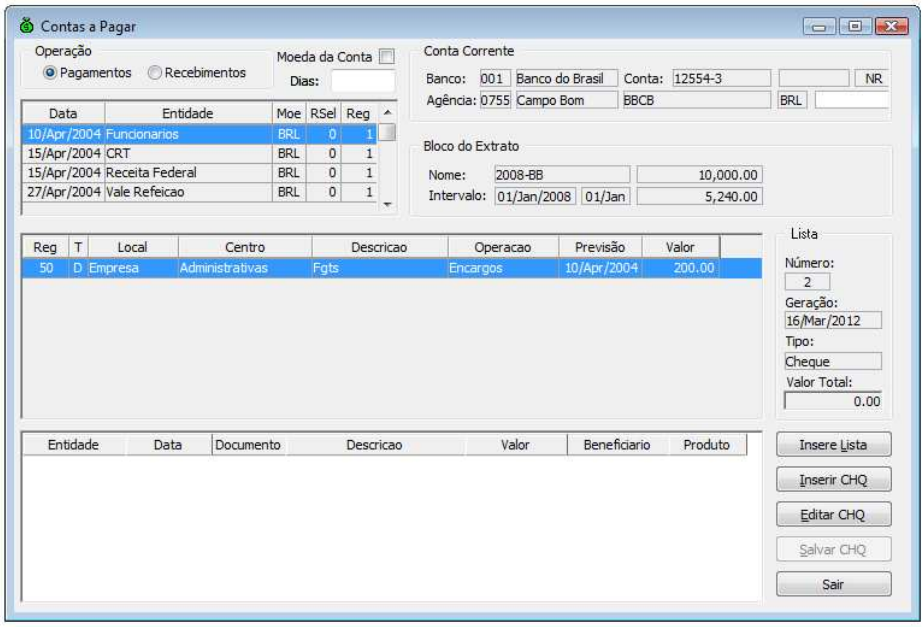
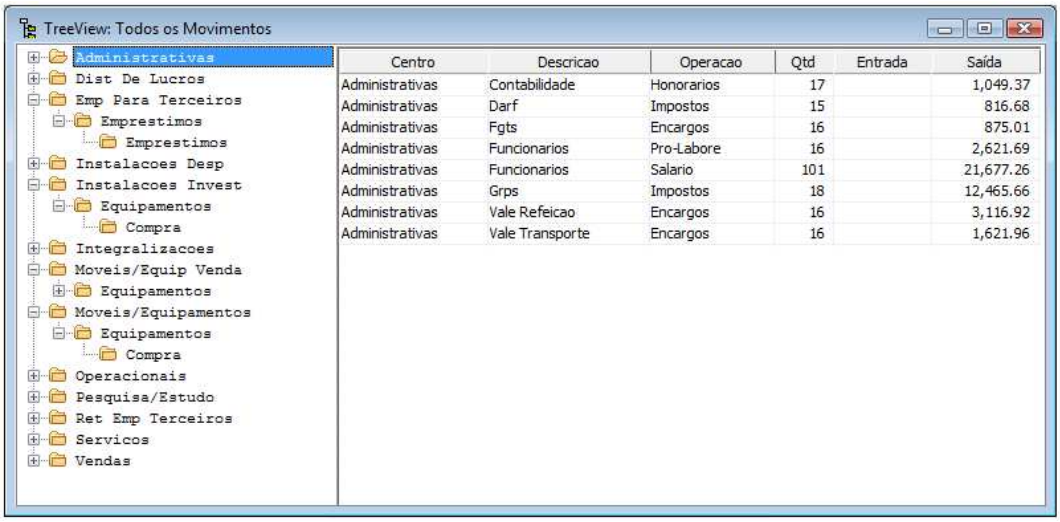
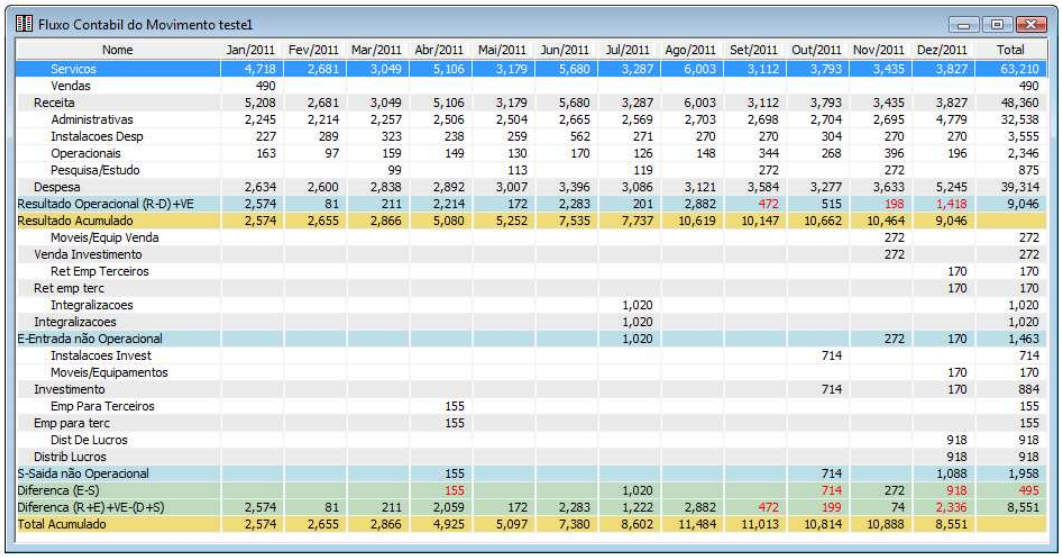


Figura A.9: Exemplo de *View* com árvore e lista.



A Figura A.10 mostra uma *View* que tem informações com referência cruzada. Eeste exemplo é um fluxo de caixa com visão contábil. Note que as colunas contêm informações do mesmo tipo, no caso são numéricas.

Figura A.10: Exemplo de *View* com referência cruzada.



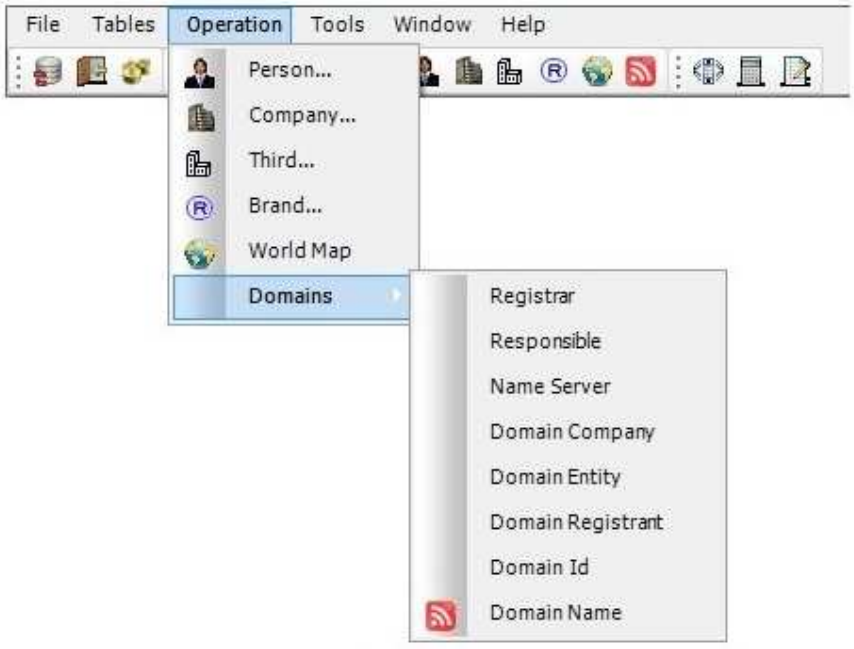
Fluxo Contabil do Movimento teste1

Nome	Jan/2011	Fev/2011	Mar/2011	Abr/2011	Mai/2011	Jun/2011	Jul/2011	Ago/2011	Set/2011	Out/2011	Nov/2011	Dez/2011	Total
Servicos	4,718	2,681	3,049	5,106	3,179	5,680	3,287	6,003	3,112	3,793	3,435	3,827	63,210
Vendas	490												490
Receita	5,208	2,681	3,049	5,106	3,179	5,680	3,287	6,003	3,112	3,793	3,435	3,827	48,360
Administrativas	2,245	2,214	2,257	2,506	2,504	2,665	2,569	2,703	2,698	2,704	2,695	4,779	32,538
Instalacoes Desp	227	289	323	238	259	562	271	270	270	304	270	270	3,555
Operacionais	163	97	159	149	130	170	126	148	344	268	396	196	2,346
Pesquisa/Estudo			99		113		119		272		272		875
Despesa	2,634	2,600	2,838	2,892	3,007	3,396	3,086	3,121	3,584	3,277	3,633	5,245	39,314
Resultado Operacional (R-D)+VE	2,574	81	211	2,214	172	2,283	201	2,882	472	515	198	1,418	9,046
Resultado Acumulado	2,574	2,655	2,866	5,080	5,252	7,535	7,737	10,619	10,147	10,662	10,464	9,046	
Moveis/Equip Venda											272		272
Venda Investimento											272		272
Ret Emp Terceiros												170	170
Ret emp terc												170	170
Integralizacoes							1,020						1,020
Integralizacoes							1,020						1,020
E-Entrada não Operacional							1,020				272	170	1,463
Instalacoes Invest										714			714
Moveis/Equipamentos												170	170
Investimento										714		170	884
Emp Para Terceiros				155									155
Emp para terc				155									155
Dist De Lucros												918	918
Distrib Lucros												918	918
S-Saida não Operacional				155						714		1,088	1,958
Diferenca (E-S)				155			1,020			714	272	918	495
Diferenca (R+E)+VE-(D+S)	2,574	81	211	2,059	172	2,283	1,222	2,882	472	199	74	2,336	8,551
Total Acumulado	2,574	2,655	2,866	4,925	5,097	7,380	8,602	11,484	11,013	10,814	10,888	8,551	

Menus e Tool Bar

Através da *Menu Bar* ou da *Tool Bar* o usuário pode selecionar funcionalidades que serão apresentadas através de *Views*. Normalmente temos o *menu* do sistema e o *menu* de cada *View* e também há variações onde fica o *menu* da *View*, que pode ficar integrado ao *menu* do sistema ou junto à *View*. A mesma característica se aplica para a *Tool Bar*. Na Figura A.11 é apresentado um exemplo de *menu* de um sistema.

Figura A.11: Exemplo de *Menu* de um sistema.



APÊNDICE B NORMALIZAÇÃO DOS PADRÕES

Com os sistemas analisados, o próximo passo foi fazer uma normalização dos elementos de interface do usuário identificados. Para cada elemento foi definido a aparência e o comportamento. Para esta normalização dividimos as interfaces do usuário nos seguintes grupos:

- *Workbench*;
- *Views*;
- Componentes básicos de interfaces do usuário;

Workbench

Conforme já descrito anteriormente o *Workbench* é a área completa do sistema e apresentada quando ele é carregado. O *Workbench* com Movable Panels e MDI têm características muito semelhantes, tendo a diferença na forma como as *Views* são apresentadas, ou seja, a diferença está *Work Area*. No *Workbench* para sistemas *web* não há um padrão fixo, mas eles também seguem algumas características que podem ser padronizadas. Tanto em um *Workbench* com Movable Panels, como em um MDI, podem ser identificadas algumas áreas padrão:

- Área de título (*Title Bar*);
- Área de *menus* (*Menu Bar*);
- Barra de ferramentas (*Tool Bar*);
- Área de trabalho (*Work Area*);
- Área de mensagens (*Status Bar*).

Title Bar

A barra de títulos é mostrada no topo da *Workbench*. Normalmente ela é usada para identificar o sistema, como pode ser visto na Figura B.1. Ela é composta por: um ícone do sistema; o nome do sistema e outras informações como usuário, versão; um botão de minimizar; um botão de maximizar; e um botão de fechar o sistema;

Figura B.1: Exemplo de Title Bar.

Os comportamentos básicos vinculados a Title Bar são:

- Quando o usuário clicar no botão fechar deve ser ativado um evento de close para cada *View* aberta, fazendo com que verifique se há ocorrência de dados não salvos. Se essa situação ocorrer, uma mensagem de alerta será exibida para o usuário questionando por salvar os dados.
- Um duplo clique na *Title Bar* maximiza o *Workbench* e se estiver maximizado ele volta ao tamanho anterior.

Menu Bar

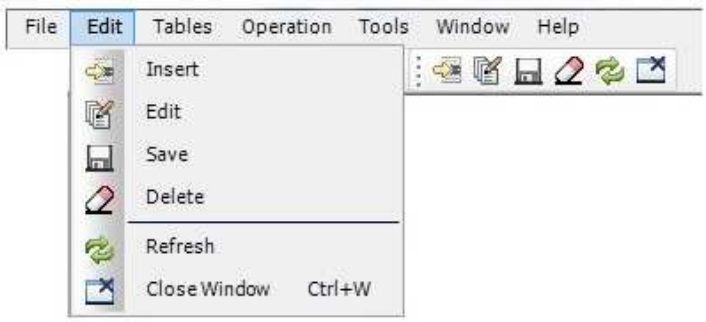
A *Menu Bar* é uma estrutura hierárquica que permite a navegação entre as funcionalidades do sistema, de forma que o usuário possa escolher qual *View* deseja utilizar. A *Menu Bar* é composta por itens que podem ser de agrupação (*Menu Group*), flags de liga e desliga (*Menu Flag*), ações que executam algum procedimento específico (*Menu Action*) ou ainda para ativar uma determinada *View*.

As aparências de uma *Menu Bar* são:

- Os itens de *menu* devem ser agrupados por funcionalidades semelhantes, sendo estes grupos podem ser separados por uma linha.
- Conceito de (...) indica que nesta opção de *menu* o usuário vai precisar inserir informações adicionais para completar o comando.
- Quanto temos um item de *menu* que tem mais sub níveis, ele deve ter uma seta para identificar que este *menu* tem mais opções de *menu*.
- Conceito de marcador para itens de *menu* que sejam *flag* de liga e desliga.

Cada item de *menu* é composto por um nome, um ícone podendo ser composto por duas imagens, uma para quando esta habilitado e outra para desabilitado, uma sequência de teclas de atalho e uma letra de acesso rápido. Também pode ter um texto de micro-help que será apresentado na área de mensagens quando o item estiver selecionado. Como características, um item de *menu* pode estar visível, habilitado ou com marcação de checado. As teclas de atalho são combinações com as teclas *alt*, *ctrl*, *shift* e uma letra. Na Figura B.2 é apresentado um exemplo de uma *Menu Bar*.

Figura B.2: Exemplo de *Menu Bar*.



Tool Bar

A *Tool Bar* é uma barra composta por itens selecionados da *Menu Bar*. Ela é útil por propiciar aos usuários acesso imediato às ações mais utilizadas. Normalmente o usuário pode selecionar que itens de menu ele deseja colocar nesta *Tool Bar*. Uma *Tool Bar* utiliza os dados definidos para cada item de menu. Na Figura B.3 é apresentado um exemplo de uma *Tool Bar*.

Figura B.3: Exemplo de Tool Bar.

Work Area

É a área do *Workbench* onde as *Views* são exibidas. Para *Workbench* do tipo MDI as *Views* são apresentadas sobrepostas, estando totalmente visível aquela que estiver com foco. Uma *View* pode estar maximizada dentro da *Work Area*. Na Figura B.4 é apresentada uma *Work Area* de *Workbench* to tipo MDI. Para *Workbench* do tipo Movable Panels a *Work Area* é dividida em painéis (*Panels*) e as *Views* ficam dentro destes *Panels*. Pode ter mais de uma *View* em cada *Panel*. Na Figura B.5 é apresentada uma *Work Area* de um *Workbench* do tipo *Movable Panels*.

Figura B.4: Exemplo de *Work Area* de *Workbench* MDI.

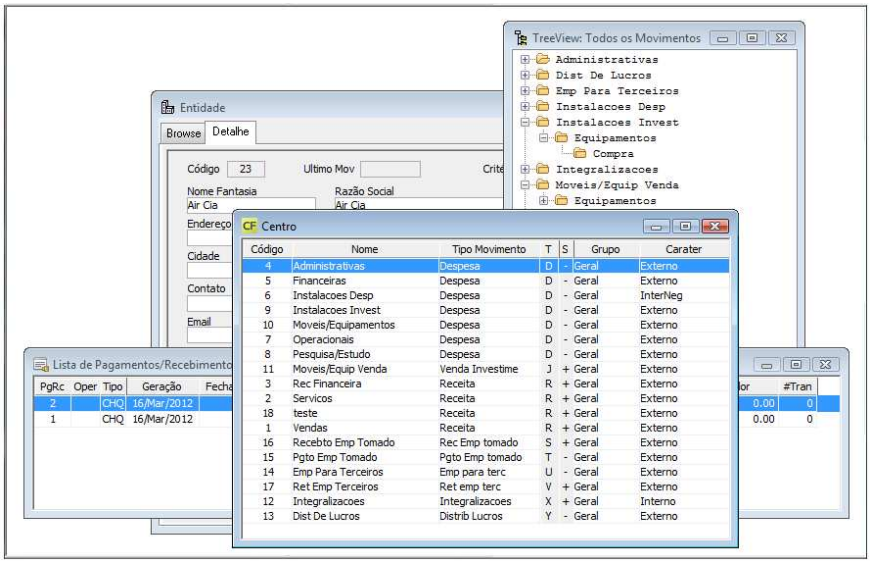
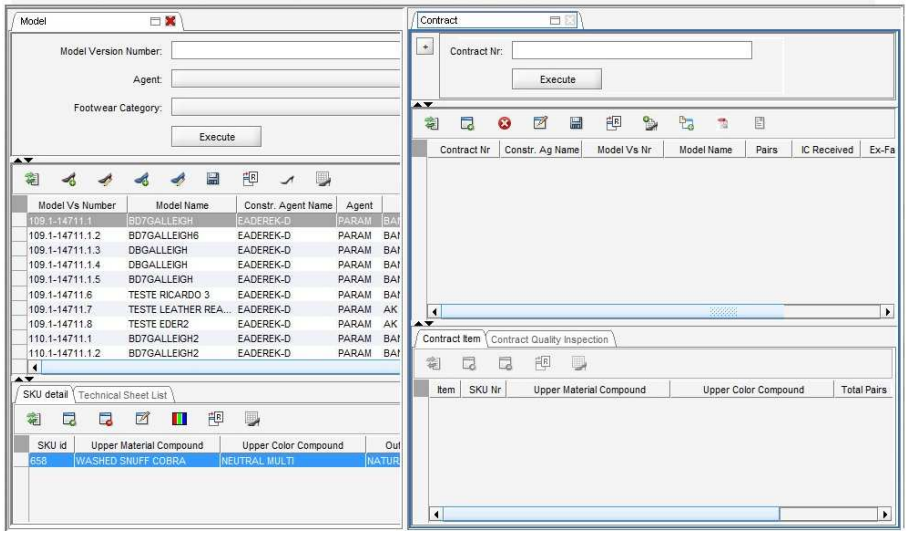


Figura B.5: Exemplo de *Work Area* de *Workbench Movable Panels*.



Status Bar

A *Status Bar* é uma área especial, na parte inferior do *Workbench*, que mostra informações sobre o estado atual do que está sendo executado e outras informações contextuais, como por exemplo, o banco no qual o usuário está conectado. Na Figura B.6 é apresentado o exemplo de uma *Status Bar*.

Figura B.6: Exemplo de *Status Bar* de *Workbench Movable Panels*.

Views

As *Views* foram divididas segundo a forma como os dados são apresentados e foram agrupadas como segue:

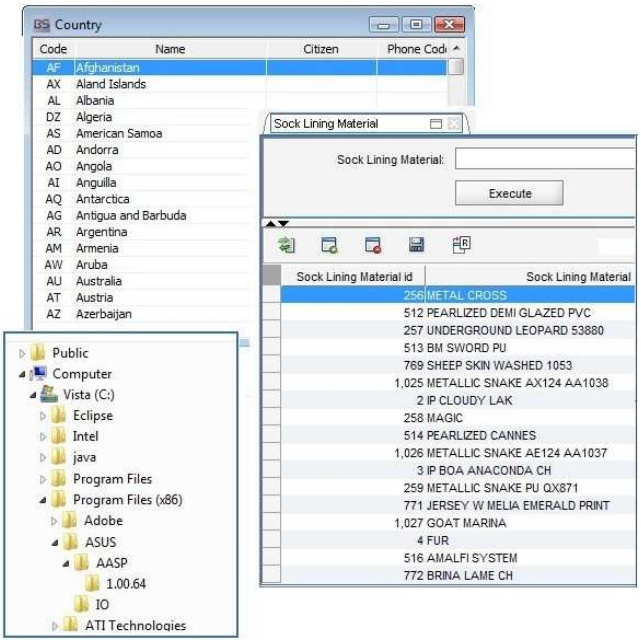
- *Single Views*: apresentam grupos de dados sem relacionamentos.
- *Composite Views*: apresentam dados relacionados no conceito de mestre e detalhe (*Master Detail*).
- *Multi Composite Views*: apresentam dados relacionados no conceito de um mestre e vários detalhes (*Master Multi Detail*).
- *Crosstab Views*: apresentam dados com referência cruzada.

Single Views

Estas *Views* são utilizadas para mostrar e manipular dados sem relacionamentos através de formulários (*Form*), listas (*List*) ou árvores (*Tree*). Podem conter operações básicas sobre estes dados como criação (*Create*), leitura (*Read*), atualização (*Update*)

ou exclusão (*Delete*), também conhecidas como operações de CRUD. Podemos ter as *Single Views* com uma lista ou árvore e um formulário para editar os dados. Elas podem ou não conter opção de filtrar os dados. Na Figura B.7 é apresentado alguns exemplos de *Single Views.*

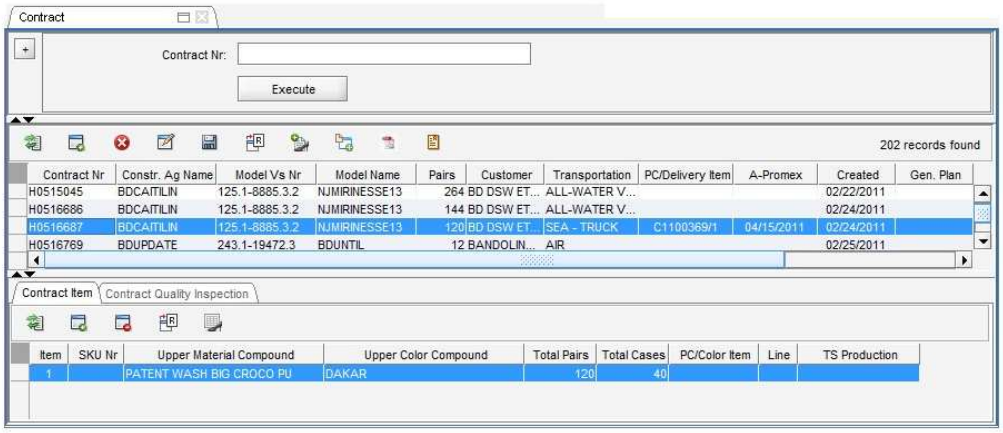
Figura B.7: Exemplos de *Single Views.*



Composite Views

Estas *Views* são utilizadas para mostrar e manipular dados relacionados, conhecidos como mestre e detalhe (*Master Detail*). Normalmente o mestre está como um formulário (*Form*) e os detalhes como lista (*List*), mas pode também ter o mestre como uma lista (*List*) ou uma árvore (*Tree*). Podem conter as operações básicas CRUD sobre os dados do mestre ou sobre os dados do detalhe. Um exemplo típico de mestre e detalhe é um pedido e seus itens de uma compra. Elas podem ou não conter opção de filtrar os dados do mestre. Na Figura B.8 é apresentado um exemplo de uma *Composite View* onde o mestre e o detalhe estão em listas.

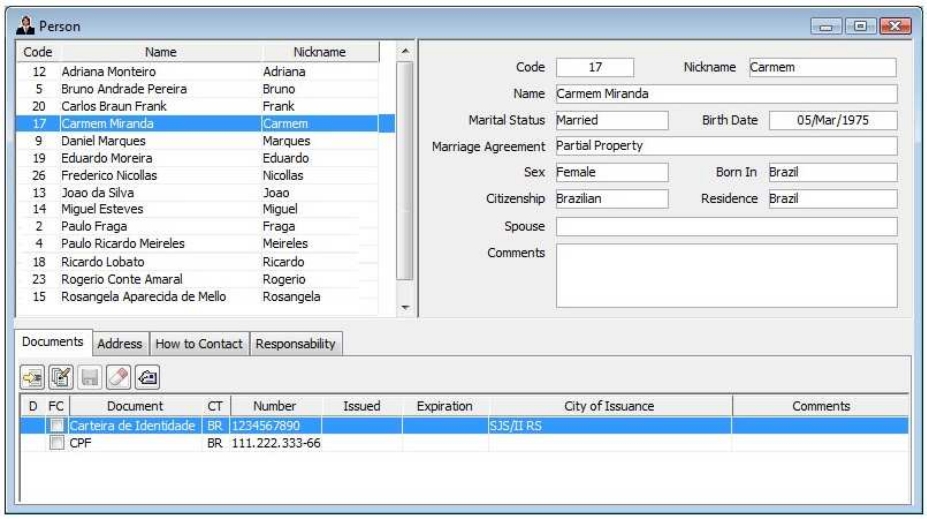
Figura B.8: Exemplos de *Composite View*.



Multi Composite Views

As *Views* deste padrão são utilizadas para mostrar e manipular dados relacionados, conhecidos como mestre e múltiplos detalhes (*Master Multi Detail*). Normalmente o mestre está com um formulário (*Form*) e os detalhes separados em abas (*Tabs*), mas pode também ter o mestre como uma lista (*List*) ou uma árvore (*Tree*). Podem conter operações básicas CRUD sobre os dados do mestre ou de cada detalhe. Elas podem ou não conter opção de filtrar os dados do mestre. Dentro de cada aba podemos ter listas, formulários ou árvores. Na Figura B.9 é apresentado um exemplo de uma *Multi Composite View*.

Figura B.9: Exemplo de Multi Composite View.



Crosstab Views

Uma *Crosstab View* permite ver os dados como uma referência cruzada. Um exemplo típico de uma referência cruzada é um fluxo de caixa, onde nas colunas temos o período, normalmente meses e nas linhas temos contas ou descrições. Ela pode ou não conter opção de filtrar os dados antes de construir a referência cruzada. Um elemento, identificado por uma linha e uma coluna, é chamado de célula e podemos mostrar alguma informação adicional de uma célula através de uma lista ou de um formulário em uma *View modal*. A Figura B.10 apresenta um exemplo de uma *Crosstab View*.

Figura B.10: Exemplos de uma Crosstab View.


Fluxo Caixa do Movimento Botao Todos Movimentos

Nome	Total-G	Jan/2011	Fev/2011	Mar/2011	Abr/2011	Mai/2011	Jun/2011	Jul/2011	Ago/2011	Set/2011	Out/2011	Nov/2011	Dez/2011	Total
SALDO INICIAL		0												
Integralizacoes	1,020							1,020						1,020
Moveis/Equip Venda	272											272		272
Ret Emp Terceiros	170												170	170
Servicos	63,210	4,718	2,681	3,049	5,106	3,179	5,680	3,287	6,003	3,112	3,793	3,435	3,827	47,870
Vendas	490	490												490
ENTRADAS	65,163	5,208	2,681	3,049	5,106	3,179	5,680	4,308	6,003	3,112	3,793	3,707	3,997	49,823
Administrativas	44,245	2,245	2,214	2,257	2,506	2,504	2,665	2,569	2,703	2,698	2,704	2,695	4,779	32,538
Dist De Lucros	918												918	918
Emp Para Terceiros	155				155									155
Instalacoes Desp	5,016	227	289	323	238	259	562	271	270	270	304	270	270	3,555
Instalacoes Invest	714										714			714
Moveis/Equipamentos	170												170	170
Operacionais	3,287	163	97	159	149	130	170	126	148	344	268	396	196	2,346
Pesquisa/Estudo	1,147			99		113		119		272		272		875
SAIDAS	55,653	2,634	2,600	2,838	3,047	3,007	3,396	3,086	3,121	3,584	3,991	3,633	6,333	41,272
DIFERENCA	9,511	2,574	81	211	2,059	172	2,283	1,222	2,882	472	199	74	2,336	8,551
ACUMULADO		2,574	2,655	2,866	4,925	5,097	7,380	8,602	11,484	11,013	10,814	10,888	8,551	

Componentes básicos de interfaces do usuário

São elementos da interface que são utilizados para formar componentes mais complexos, como as *Views*. Os elementos básicos identificados neste estudo são:

- Ações (*Actions*);
- Formulário (*Form*);
- Listas (*List*);
- Árvores (*Tree*);
- Abas (*Tab*).

Actions

Diversas ações ocorrem de forma semelhante em todos os sistemas analisados e aqui elas foram normalizadas.

Inserir (Insert)

Está ação está relacionada com a inclusão de um novo registro de dados para a classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de incluir ou adicionar e teclas comuns de atalho são o *ctrl+I* e *ins*.

Editar (Edit)

Está ação está relacionada com a passagem de um registro selecionado para o modo de edição, onde seus valores podem ser alterados. O registro deve ser da classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de alterar e teclas comuns de atalho são clique simples ou clique duplo sobre um formulário.

Deletar (Delete)

Está ação permite a exclusão de um registro já existente e que esteja selecionado na classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de remover, excluir ou apagar. Teclas comuns de atalho são o *del* e o *ctrl+D*.

Salvar (Save)

Após um registro ter sido inserido ou alterado, esta ação permite fazer a persistência deste registro para a classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de gravar. Teclas comuns de atalho são *ctrl+S* e o *ctrl+W*.

Atualizar (Refresh)

A funcionalidade está relacionada com uma nova leitura dos dados e apresentação. Os dados apresentados devem ser da classe ou classes que estiverem vinculadas ao contexto. Tecla de atalho normalmente utilizada é o *F5*.

Sair (Exit)

Esta ação encerra uma operação em curso, normalmente fecha uma *View*. Também chamado de fechar ou close.

Copiar (Copy)

Esta ação dá a possibilidade de copiar dados selecionados para uma área de transferência. Tecla de atalho utilizada é *ctrl+C*.

Cortar (Cut)

Esta ação dá a possibilidade de copiar dados selecionados para uma área de transferência, a diferença do copiar é que os dados são removidos da origem. Tecla de atalho utilizada é *ctrl+X*.

Colar (Paste)

Esta ação permite trazer dados da área de transferência para a edição corrente. Tecla de atalho utilizada é *ctrl+V*.

Form

Este componente é utilizado para edição ou apresentação de informações. Normalmente para um determinado registro. Um *Form* sempre será apresentado dentro de uma *View*. Na Figura B.11 é apresentado um exemplo de *Form*.

Figura B.11: Exemplo de um Form.

Um comportamento padrão entre os campos de um *Form* são as teclas *tab* que passa para o próximo campo e o *shift-tab* que volta para o campo anterior.

List

Este componente possibilita a apresentação dos dados para visualização ou edição através de uma lista. Utilizado normalmente para apresentação ou edição de vários registros. Um cursor identificará a linha corrente. As setas direcionais para cima e para baixo permitem navegar pelas linhas, as teclas *page up* e *page down* permitem navegar pela página, a tecla *ctrl+home* vai para o primeiro registro e a tecla *ctrl+end* vai para o último registro.

Normalmente um clique no cabeçalho é utilizado para classificar em ordem crescente e um novo clique inverte a ordem. Também *ctrl+clique* permite selecionar mais de uma coluna para classificar. A Figura B.12 apresenta um exemplo de *List*.

Figura B.12: Exemplo de um List.

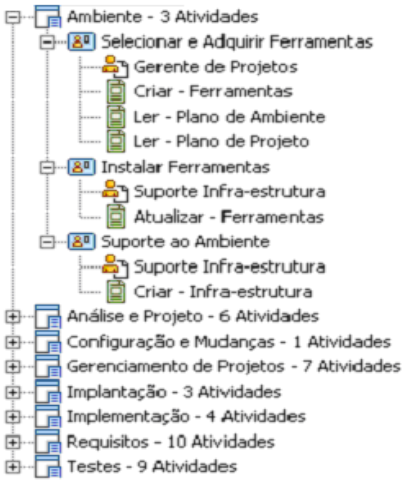
A/I	Article	Raw Material	Color Base	Grade	Sample	Production	Cutting Yield
A	Burgundy pump	Bovines Sides	Black	boot	3.50	3.00	75%
A	Burgundy pump	Bovines Sides	Black	pump	3.50	3.00	75%
A	Atacama pump	Cow Sides	Colors	boot	3.00	2.20	75%
A	Atacama pump	Cow Sides	Colors	pump	3.00	2.20	75%
A	Brotini pump	Calf Hides	Black	boot	3.50	3.40	75%
A	Brotini pump	Calf Hides	Black	pump	3.50	3.40	75%
A	Antik luster patent	Cow Sides	Beige	boot	4.00	3.80	76%
A	Antik luster patent	Cow Sides	Beige	pump	4.00	3.80	75%

Tree

A intenção deste componente é permitir a exibição de dados de forma hierárquica, distribuída em níveis. Ao ser inicializado a estrutura pode ser definida como aberta ou fechada. Se estiver aberta ela tem a seguinte navegação pelo teclado: tecla *home* vai

para o primeiro elemento da hierarquia, *end* vai para o último elemento da hierarquia, *page up* vai uma página acima, *page down* vai uma página abaixo. A seta direcional para direita expande a exibição dos elementos e se já estiver expandido vai para um nível abaixo. A seta direcional para esquerda contrai a exibição dos elementos do nível e se já tiver contraída vai para um nível acima. As setas direcionais para cima e para baixo navegam na estrutura hierárquica que está aberta. Os itens da estrutura podem ser selecionados com o mouse. A Figura B.13 mostra um exemplo de *Tree*.

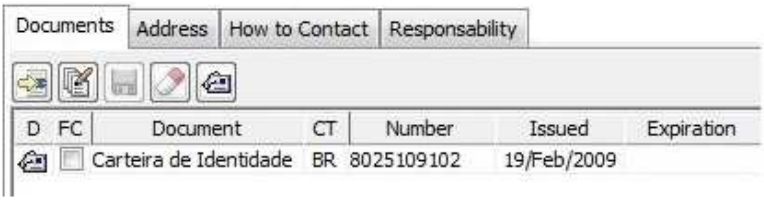
Figura B.13: Exemplo de um *Tree*.



Tab

Este componente oferece uma organização dos dados em abas. Dentro de uma aba é possível compor com outros componentes como *Form*, *List* ou *Tree*. A Figura B.14 mostra um exemplo de *Tab*.

Figura B.14: Exemplo de um Tab.

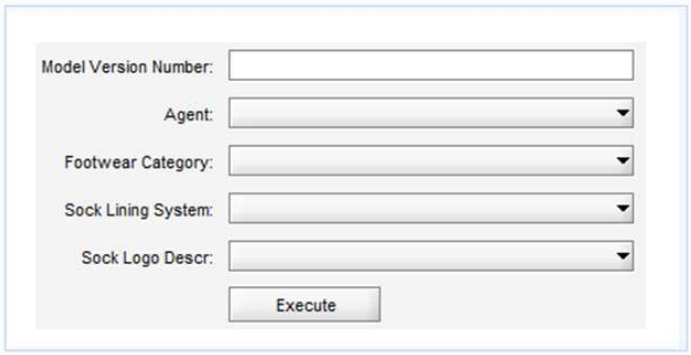


Filter

A intenção deste componente é oferecer ao usuário uma área com campos para consultas que irão filtrar os dados de acordo com um determinado critério. O critério depende do contexto onde estiver. Este componente é formado por um *Form*, e assim, herda seus comportamentos. Ele tem uma ação para executar a seleção dos dados. Pode

opcionalmente ter outra ação para limpar os dados do filtro de uma eventual seleção. Na Figura B.15 é apresentado um exemplo de *Filter*.

Figura B.15: Exemplo de um Filter.



APÊNDICE C PADRÕES DE ELEMENTOS DE UI

Aqui está descrito detalhadamente os padrões de interface identificados nos sistemas de *software* analisados. Estes padrões foram identificados, classificados e foi criado um elemento gráfico para identificá-los, visando facilitar o processo de criação do metamodelo e dos modelos.

Padrões Primitivos (Primitive Patterns)

Os padrões primitivos são aqueles utilizados para construir os outros.

Actions Patterns

São os padrões que definem o comportamento que está disponível para os elementos de interface do usuário. Todos os atributos desta *action* são padronizados e eles são *Icon*, *ShortCut*, *MenuHelp* e *ToolTips*.

Nome	**Inserir (Insert)**
Descrição	Está ação está relacionada com a inclusão de um novo registro de dados para a classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de incluir ou adicionar e teclas comuns de atalho são o *ctrl+I* e *ins*.

Nome	**Editar (Edit)**
Descrição	Está ação está relacionada com a passagem de um registro selecionado para o modo de edição, onde seus valores podem ser alterados. O registro deve ser da classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de alterar e teclas comuns de atalho são clique simples ou clique duplo sobre um formulário.

Nome	**Deletar (Delete)**

Descrição	Está ação permite a exclusão de um registro já existente e que esteja selecionado na classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de remover, excluir ou apagar. Teclas comuns de atalho são o *del* e o *ctrl+D.*

Nome	**Salvar (Save)**
Descrição	Após um registro ter sido inserido ou alterado, esta ação permite fazer a persistência deste registro para a classe ou classes que estiverem vinculadas ao contexto onde ele se encontra. Também chamado de gravar. Teclas comuns de atalho são *ctrl+S* e o *ctrl+W.*

Nome	**Atualizar (Refresh)**
Descrição	A funcionalidade está relacionada com uma nova leitura dos dados e apresentação. Os dados apresentados devem ser da classe ou classes que estiverem vinculadas ao contexto. Tecla de atalho utilizada é o *F5*.

Nome	**Copiar (Copy)**
Descrição	Esta ação dá a possibilidade de copiar dados selecionados para uma área de transferência. Tecla de atalho utilizada é *ctrl+C.*

Nome	**Cortar (Cut)**
Descrição	Esta ação dá a possibilidade de copiar dados selecionados para uma área de transferência, a diferença do copiar é que os dados são removidos da origem. Tecla de atalho utilizada é *ctrl+X*.

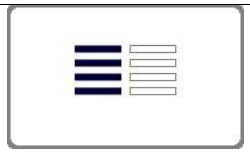
Nome	**Colar (Paste)**
Descrição	Esta ação permite trazer dados da área de transferência para a edição corrente. Tecla de atalho utilizada é *ctrl+V*

Nome	**Sair (Exit)**
Descrição	Esta ação encerra uma operação em curso, normalmente fecha

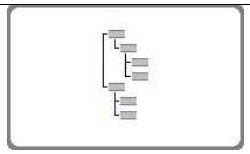
	uma View. Também chamado de *fechar* ou *close*.

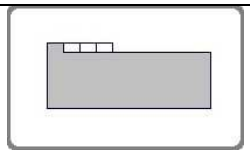
View Patterns

São aqueles que são utilizados para apresentar ou editar dados.

Representação	
Nome	**Form**
Descrição	Este padrão representa os formulários que são utilizados para apresentar ou editar dados em uma View. Ele pode ser composto de vários elementos de dados e estes elementos podem ter diferentes tipos de dados. A navegação básica nos dados é feita pela tecla *tab* que avança para o próximo campo ou *shift tab* que volta para o anterior.
Atributos	Deve conter uma lista de campos com LabelName e FieldName. **Actions**: {Edit, Save} estas são as actions possíveis.

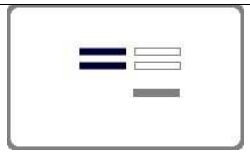
Representação	
Nome	**List**
Descrição	Este padrão representa as listas que são utilizadas para apresentar ou editar dados em uma View. Ele pode ser composto de vários elementos de dados e estes elementos podem ter diferentes tipos de dados. A navegação básica nas linhas pode ser feita por setas para cima e para baixo, por *page up* ou *page down* que navega por páginas acima ou abaixo e *ctrl home* e *ctrl end* para ir ao início ou para o fim da lista. Também pode ser usado o mouse para clicar em linhas especificas.
Atributos	Deve conter uma lista de campos com LabelName e FieldName. **Actions**: {Insert, Edit, Delete, Save, Refresh} estas são as actions possíveis.

Representação	
Nome	**Tree**
Descrição	Quando há necessidade de apresentar os dados em estruturas de árvore podemos utilizar este padrão. Numa árvore os dados podem ser apresentados ou editados em uma View. Cada item de uma árvore pode ter dois ícones associados, um para quando ele estiver fechado e um para quando ele estiver aberto. Ao ser inicializada a estrutura pode ser definida como aberta ou fechada. Se estiver aberta ela tem a seguinte navegação pelo teclado: tecla *home* vai para o primeiro elemento da hierarquia, *end* vai para o último elemento da hierarquia, *page up* vai uma página acima, *page down* vai uma página abaixo. A seta direcional para direita expande a exibição dos elementos e se já estiver expandido vai para um nível abaixo. A seta direcional para esquerda contrai a exibição dos elementos do nível e se já tiver contraída vai para um nível acima. As setas direcionais para cima e para baixo navegam na estrutura hierárquica que está aberta. Os itens da estrutura também podem ser selecionados com o mouse.
Atributos	Deve conter uma lista hierárquica de campos com PictureName e FieldName. **Actions**: {Insert, Edit, Delete, Save, Refresh} estas são as actions possíveis.

Representação	
Nome	**Tab**
Descrição	O padrão Tab pode ser utilizado para apresentar dados separados por abas. Dentro de cada aba é possível ter formulários, listas ou árvores. Um componente *Tab* pode ser colocado dentro de uma View.
Atributos	TabNumber: Número de abas que conterá no componente.

Search Patterns

São os padrões utilizados para consultar dados.

Representação	
Nome	**Filter**
Descrição	Este padrão representa as construções utilizadas para selecionar um conjunto de dados. Normalmente as informações são definidas num formulário e estas informações são utilizadas para executar uma consulta em uma base de dados. Ele tem uma Action para executar a seleção dos dados. Pode opcionalmente ter outra Action para limpar os dados do filtro de uma eventual seleção.
Atributos	Deve conter uma lista de campos com LabelName e FieldName.

Padrões Básicos (Basic Patterns)

Os padrões básicos representam as UIs propriamente ditas, cada um destes padrões tem uma representação direta com uma UI que seja implementada em um sistema de informação.

Padrões de Workbench (Workbench Patterns)

Este padrão representa o início do sistema de informação.

Representação	
Nome	**Workbench**
Descrição	Este padrão representa o início do sistema de informação, ele deve ter um determinado tipo, como *Movable Panels*, MDI ou *Web*. Junto com ele devem ser definidos os atributos de conexão com a base de dados, nome do sistema, ícone associado ao sistema. Também deve ser definido se as Actions de uma View serão incorporadas ao Menu do sistema ou se cada View terá sua própria estrutura de menu vinculada à View.
Atributos	**Type**: Deve ter um determinado tipo, como Movable Panels, MDI ou *Web*. **SystemName**: Deve ter o nome do sistema.

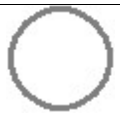
	Icon: Icone associado ao sistema.

Padrões de Menu (Menu Patterns)

Estas representações de Menu permitem construir a estrutura de um menu para um sistema de informação. Para compor um menu uma estrutura de atributos de menu é necessária, esta estrutura chamaremos M**enuStructure** e ela é composta como segue:

Order: A ordem que aparece no junto com outros itens de mesmo nível no menu.
SeparatorAfter: Yes/No indicando se terá uma linha de separação após no nome na lista do menu.
Icon: Icone associado ao menu action.
ShortCut: Tecla de atalho para o menu action.
MenuHelp: Help que será mostrado na linha de status do *Workbench* quando selecionado no menu.
ToolTips: Dica que será mostrada quando tiver uma ação de mouseOver sobre a opção do menu ou toolbar.

Representação	
Nome	**Menu Group**
Descrição	Este padrão representa os itens que são agrupadores de menus, eles são utilizados para criar a árvore de menus. O item raiz é o próprio menu. Para cada item de grupo deve ser definido o nome e opcionalmente uma tecla de acesso rápido.
Atributos	**Name**: Deve ter o nome do grupo de menu. **Order**: A ordem que aparece no junto com outros itens de mesmo nível no menu. **SeparatorAfter**: Yes/No indicando se terá uma linha de separação após no nome na lista do menu.

Representação	
Nome	**Menu Action**
Descrição	Muitas vezes é necessário uma opção de menu que executa

	diretamente um procedimento sem a necessidade de carregar uma interface de usuário específica. Este padrão foi criado para representar esta necessidade. Para um Menu Action deve ser definido o nome e atribuído o procedimento a ser executado, que pode ser via um determinado método de uma classe. Opcionalmente pode ser definido um atalho, uma tecla de acesso rápido, um texto para ser exibido como micro help, um ícone para habilitado e um para desabilitado. Também pode ser informado se ele aparecerá na Tool Bar. Cada View ligada ao menu deverá ter os mesmos atributos de um Menu Action.
Atributos	**Name**: Deve ter o nome do menu action. M**enuStructure**: A estrutura para compor o menu.

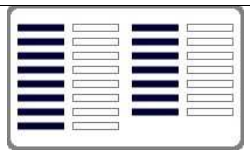
Representação	
Nome	**Menu Flag**
Descrição	Também é necessário uma opção de menu que permita ligar ou desligar determinadas características do sistema. Este padrão foi criado para representar esta necessidade. Para um *Menu Flag* deve ser definido o nome e opcionalmente pode ser definido um atalho, um texto para ser exibido como micro help, um ícone para ligado e um para desligado. Também pode ser informado se ele aparecerá na Tool Bar.
Atributos	**Name**: Deve ter o nome do menu flag. M**enuStructure**: A estrutura para compor o menu.

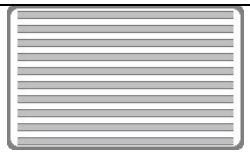
Single View Patterns

Estas Views são utilizadas para mostrar e manipular dados sem relacionamentos através de formulários, listas ou árvores. Todas as Views que forem ligadas diretamente ao menu devem ter os atributos de menu, conforme descrito para Menu Action. Todas as Views devem ter um nome e podem ter um ícone associado.

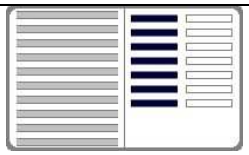
Representação	
Nome	**View Void**

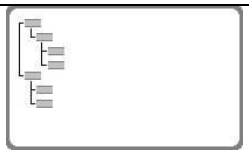
Descrição	Este padrão de View representa as interfaces de usuário que não há um padrão pré-definido para ser utilizado e, neste caso, pode ser construído com os padrões primitivos. Ele sempre será composto de pelo menos uma View.
Atributos	**Name**: Deve ter o nome da view. M**enuStructure**: A estrutura para compor o menu. **Actions**: {Exit} esta é a action possível.

Representação	
Nome	**View Form**
Descrição	Um padrão *View Form* é um padrão que representa uma interface de usuário que permite apresentar ou editar dados de um registro. Normalmente ele estará vinculado a uma classe que representa estes dados. Um *View Form* terá como Actions o Exit e pode ter o Save quando ela permitir editar dados. Ele será composto de uma View e um Form.
Atributos	**Name**: Deve ter o nome da view. M**enuStructure**: A estrutura para compor o menu. **Actions**: {Exit} esta é a action possível.

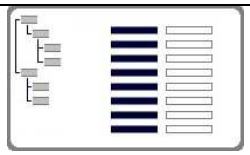
Representação	
Nome	**View List**
Descrição	O padrão *View List* é utilizado para representar uma interface de usuário que pode apresentar ou editar dados em uma lista. Normalmente ele estará vinculado a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na lista. Um *View List* pode ter as Actions de CRUD e mais o Exit e opcionalmente pode ter outras Actions. Ele será composto de uma View, uma List e opcionalmente um Filter.
Atributos	**Name**: Deve ter o nome da view. M**enuStructure**: A estrutura para compor o menu.

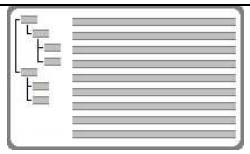
	Actions: {Exit} esta é a action possível.

Representação	
Nome	**View List Form**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar dados em uma lista e editar em um formulário. O formulário pode ser aberto na mesma View ou numa nova ViewForm de forma modal. Normalmente ele estará vinculado a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na lista. Um View List Form pode ter as Actions de CRUD e mais o Exit para a List, opcionalmente pode ter outras Actions. Para o Form, se for modal, terá as Actions de save e exit. Ele será composto de uma View, uma List e opcionalmente um Filter e um Form. Quando uma linha é selecionada os dados associados ao registro da linha são apresentados no formulário. Quando a Action Insert é ativada um formulário em branco é associado permitindo a inclusão de um novo registro.
Atributos	**Name**: Deve ter o nome da view. M**enuStructure**: A estrutura para compor o menu. **Actions**: {Exit} esta é a action possível.

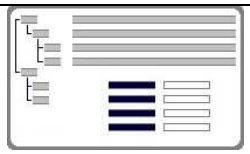
Representação	
Nome	**View Tree**
Descrição	O padrão *View Tree* é utilizado para representar uma interface de usuário que pode apresentar ou editar dados em uma árvore. Normalmente ele estará vinculado a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na árvore. Um *View Tree* pode ter as *Actions* de CRUD e mais o *Exit*, opcionalmente pode ter outras *Actions*. Ele será composto de uma *View*, uma *Tree* e opcionalmente um *Filter*.
Atributos	**Name**: Deve ter o nome da view. M**enuStructure**: A estrutura para compor o *menu*.

	Actions: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree Form**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar dados em uma árvore e editar em um formulário. O formulário pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. Normalmente ele estará vinculado a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na árvore. Um *View Tree Form* pode ter as *Actions* de CRUD e mais o *Exit* para a *Tree*, opcionalmente pode ter outras *Actions*. Para o *Form*, se for *modal*, terá as *Actions* de *Save* e *Exit*. Ele será composto de uma *View*, uma *Tree*, um *Form* e opcionalmente um *Filter*. Quando um item da árvore é selecionado os dados associados a este item são apresentados no formulário. Quando a *Action Insert* é ativada um item é criado e um formulário em branco é associado permitindo a inclusão de um novo registro.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar dados em uma árvore e os dados relacionados a um item da árvore listados numa Lista que permite edição. Tanto a *Tree* como a *List* estão vinculados a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na árvore. Um *View Tree List* pode ter as *Actions* de CRUD e mais o *Exit* para a *Tree*, opcionalmente pode ter outras *Actions*. Este padrão é composto de uma *View*, uma *Tree*, uma *List* e opcionalmente um *Filter*. Quando um item da árvore é selecionado os dados associados a este item são apresentados na lista e podem ser editados. Quando a *Action Insert* é

	ativada um item é criado e uma é inserida na lista permitindo a inclusão de um novo registro.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

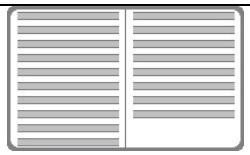
Representação	
Nome	**View Tree List Form**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar dados em uma árvore, mostrar os relacionados em uma lista e editar os detalhes em um formulário. O formulário pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. Tanto a *Tree* como a *List* e o *Form* estão vinculados a uma classe que representa estes dados. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão mostrados na árvore. Um *View Tree List Form* pode ter as *Actions* de CRUD e mais o *Exit* para a *Tree*, opcionalmente pode ter outras *Actions*. Para o *Form*, se for *modal*, terá as *Actions* de *Save* e *Exit*. A *List* terá as *Actions* de *Insert* e *Delete*. Este padrão é composto de uma *View*, uma *Tree*, um *List* e um *Form* e opcionalmente um *Filter*. Quando um item da árvore é selecionado os dados associados a este item são apresentados na lista. Quando uma linha da lista é selecionada os dados são apresentados no formulário. Quando a *Action Insert* é ativada um item é criado e um formulário em branco é associado permitindo a inclusão de um novo registro.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

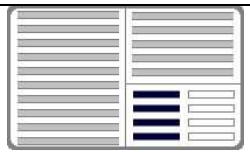
Composite View Patterns

Estas *Views* são utilizadas para mostrar e manipular dados relacionados, conhecidos como mestre e detalhe (*Master Detail*). Podem conter as operações básicas CRUD sobre os dados do mestre ou sobre os dados do detalhe. Um exemplo típico de mestre e detalhe é um pedido e seus itens de uma compra.

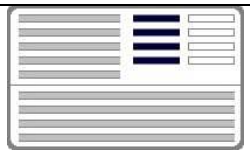
View List Master Detail

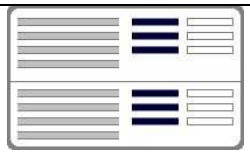
Nestes padrões, o *Master* estará numa lista e podemos ter algumas composições diferentes tanto para o *master* como para o *detail*.

Representação	
Nome	**View List Master Detail List**
Descrição	O padrão *View List Master Detail List* representa as *interfaces* de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma lista e o detalhe também em uma lista. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *List* para o mestre e opcionalmente um *Filter* e uma *List* para o detalhe. Tanto inserção ou edição do mestre quanto do detalhe ocorre nas listas.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View List Master Detail List Form**
Descrição	O padrão *View List Master Detail List Form* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma lista e o detalhe em uma lista complementada com um formulário. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as Actions de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *List* para o mestre e opcionalmente um *Filter* e uma *List* e um *Form* para o detalhe. A inserção ou edição do mestre ocorre na própria lista. A inserção do detalhe ocorre no formulário que pode ser aberto na mesma *View* ou em uma nova *ViewForm* de forma *modal*.
Atributos	**Name**: Deve ter o nome da *view*.

	M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

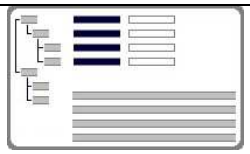
Representação	
Nome	**View List Form Master Detail List**
Descrição	O padrão *View List Form Master Detail List* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma lista complementado com o *Form* e o detalhe em uma lista. O mestre está vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e uma *List* para o detalhe. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou em uma nova *ViewForm* de forma *modal*. A inserção ou edição do detalhe ocorre na própria lista.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

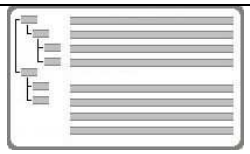
Representação	
Nome	**View List Form Master Detail List Form**
Descrição	O padrão *View List Form Master Detail List Form* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma lista complementada com o *Form* e o detalhe também em uma lista complementada com o *Form*. O mestre está vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e uma *List* e um *Form* para o

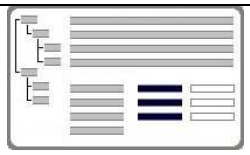
	detalhe. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção ou edição do detalhe também ocorre em um formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

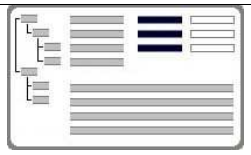
View Tree Master Detail

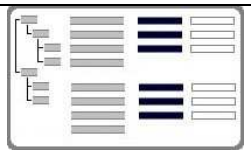
Nestes padrões, o *Master* estará numa árvore e podemos ter algumas composições diferentes tanto para o *master* como para o *detail*.

Representação	
Nome	**View Tree Form Master Detail List**
Descrição	O padrão *View Tree Form Master Detail List* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma árvore complementada com um formulário e o detalhe em uma lista. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *Tree* e um *Form* para o mestre e opcionalmente um *Filter* e uma *List* para o detalhe. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção do detalhe ocorre na própria lista.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List Master Detail List**
Descrição	O padrão *View Tree List Master Detail List* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma árvore complementada com uma lista e o detalhe em uma lista. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *Tree* e uma *List* para o mestre e opcionalmente um *Filter* e uma *List* para o detalhe. A inserção ou edição do mestre ocorre na lista. A inserção do detalhe ocorre na própria lista.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List Master Detail List Form**
Descrição	O padrão *View Tree List Master Detail List Form* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma árvore complementada com uma lista e o detalhe em uma lista complementada com um formulário. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *Tree* e uma *List* para o mestre e opcionalmente um *Filter* e uma *List* e um *Form* para o detalhe. A inserção ou edição do mestre ocorre na lista. A inserção do detalhe ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List Form Master Detail List**
Descrição	O padrão *View Tree List Form Master Detail List* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma árvore complementada com uma lista e um formulário e o detalhe em uma lista. O mestre esta vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *Tree*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e uma *List* para o detalhe. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma modal. A inserção do detalhe ocorre na própria lista.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List Form Master Detail List Form**
Descrição	O padrão *View Tree List Form Master Detail List Form* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e detalhe. O mestre está em uma árvore complementada com uma lista e um formulário e o detalhe em uma lista também complementada por um formulário. O mestre está vinculado a uma classe e o detalhe a outra classe dependente dessa. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para o detalhe. Ele é composto de uma *View*, uma *Tree*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e uma *List* e um *Form* para o detalhe. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção do detalhe ocorre no formulário que pode ser aberto na mesma *View* ou numa nova

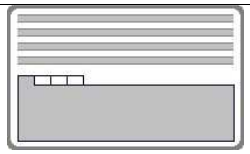
	ViewForm de forma *modal*.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

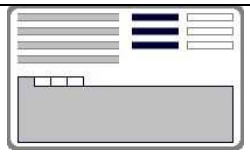
Multi Composite View Patterns

As *Views* deste padrão são utilizadas para mostrar e manipular dados relacionados, conhecidos como mestre e múltiplos detalhes (*Master Multi Detail*). O mestre está em uma lista ou em uma árvore e os detalhes separados em abas (*Tabs*). Podem conter operações básicas CRUD sobre os dados do mestre e para cada detalhe. Elas podem ou não conter opção de filtrar os dados do mestre. Dentro de cada aba podemos ter listas, formulários ou árvores.

View List Master Multi Detail

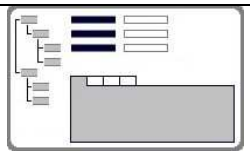
Nestes padrões, o *Master* estará numa lista e podemos ter algumas composições diferentes tanto para o *master* como para o *detail*.

Representação	
Nome	**View List Master Multi Detail Tab**
Descrição	O padrão *View List Master Detail Tab* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e de múltiplos detalhes. O mestre está em uma lista e os detalhes estão em abas. Dentro de cada aba pode ter formulários, listas ou árvores. O mestre está vinculado a uma classe e os detalhes em outras classes dependentes da classe do mestre. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para cada detalhe. Ele é composto de uma *View*, uma *List* para o mestre e opcionalmente um *Filter* e um *Tab*. Pode haver diversas abas para os detalhes e dentro de cada aba o detalhe pode ser implementado por *Form, List* ou *Tree*. A inserção ou edição do mestre ocorre na lista. A inserção do detalhe ocorre para cada aba.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

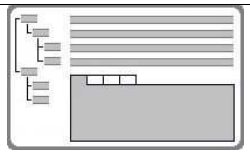
Representação	
Nome	**View List Form Master Multi Detail Tab**
Descrição	O padrão *View List Form Master Detail Tab* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e de múltiplos detalhes. O mestre está em uma lista complementada por um formulário e os detalhes estão em abas. Dentro de cada aba pode ter formulários, listas ou árvores. O mestre está vinculado a uma classe e os detalhes em outras classes dependentes da classe do mestre. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para cada detalhe. Ele é composto de uma *View*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e um *Tab* com abas para cada detalhe e dentro de cada aba o detalhe pode ser implementado por *Form*, *List* ou *Tree*. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção do detalhe ocorre para cada aba.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

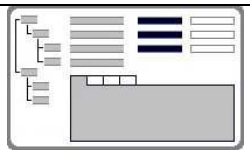
View Tree Master Multi Detail

Nestes padrões, o *Master* estará numa árvore e podemos ter algumas composições diferentes tanto para o *master* como para o *detail*.

Representação	
Nome	**View Tree Form Master Multi Detail Tab**
Descrição	O padrão *View Tree Form Master Detail Tab* representa as interfaces de usuário que são utilizadas para apresentar dados de *mestre* e de múltiplos detalhes. O mestre está em uma árvore

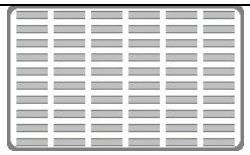
	complementada por um formulário e os detalhes estão em abas. Dentro de cada aba pode ter formulários, listas ou árvores. O mestre está vinculado a uma classe e os detalhes em outras classes dependentes da classe do mestre. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para cada detalhe. Ele é composto de uma *View*, uma *Tree* e um *Form* para o mestre e opcionalmente um *Filter* e um *Tab* com abas para cada detalhe e dentro de cada aba o detalhe pode ser implementado por *Form*, *List* ou *Tree*. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção do detalhe ocorre para cada aba.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Tree List Master Multi Detail Tab**
Descrição	O padrão *View Tree List Master Detail Tab* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e de múltiplos detalhes. O mestre está em uma árvore complementada por uma lista e os detalhes estão em abas. Dentro de cada aba pode ter formulários, listas ou árvores. O mestre está vinculado a uma classe e os detalhes em outras classes dependentes da classe do mestre. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para cada detalhe. Ele é composto de uma *View*, uma *Tree* e uma *List* para o mestre e opcionalmente um *Filter* e um *Tab* com abas para cada detalhe e dentro de cada aba o detalhe pode ser implementado por *Form*, *List* ou *Tree*. A inserção ou edição do mestre ocorre na lista. A inserção do detalhe ocorre para cada aba.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

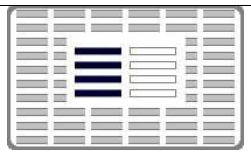
Representação	
Nome	**View Tree List Form Master Multi Detail Tab**
Descrição	O padrão *View Tree List Form Master Detail Tab* representa as interfaces de usuário que são utilizadas para apresentar dados de mestre e de múltiplos detalhes. O mestre está em uma árvore complementada por uma lista e um formulário e os detalhes estão em abas. Dentro de cada aba pode ter formulários, listas ou árvores. O mestre está vinculado a uma classe e os detalhes em outras classes dependentes da classe do mestre. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao mestre. Este padrão tem as *Actions* de CRUD para o mestre e também para cada detalhe. Ele é composto de uma *View*, uma *Tree*, uma *List* e um *Form* para o mestre e opcionalmente um *Filter* e um *Tab* com abas para cada detalhe e dentro de cada aba o detalhe pode ser implementado por *Form*, *List* ou *Tree*. A inserção ou edição do mestre ocorre no formulário que pode ser aberto na mesma *View* ou numa nova *ViewForm* de forma *modal*. A inserção do detalhe ocorre para cada aba.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

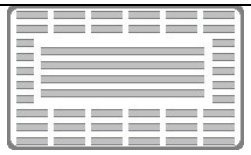
Crosstab View Pattern

Um padrão *Crosstab View* permite ver os dados como uma referência cruzada. Um exemplo típico de um *Crosstab* é um fluxo de caixa, onde nas colunas temos o período, normalmente meses e nas linhas temos contas ou descrições. Um elemento da *Crosstab*, identificado por uma linha e uma coluna, é chamado de célula.

Representação	
Nome	**View Crosstab**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar ou editar dados em referência cruzada. Os dados estão vinculados a uma ou mais classes. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão

	aplicadas ao carregar a lista. A lista tem a particularidade de ser do tipo grade. Ele é composto de uma *View* e um *List* para a referência cruzada e opcionalmente um *Filter*. Podemos ter as *Actions* de CRUD para uma célula, o que é feito na própria lista.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Crosstab Form**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar e editar dados em referência cruzada. Os dados estão vinculados a uma ou mais classes. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao carregar a lista. A lista tem a particularidade de ser do tipo grade. Ele é composto de uma *View*, um *List* para a referência cruzada e um *Form* e opcionalmente um *Filter*. Podemos ter as *Actions* de CRUD para uma célula. A edição ou inserção é feita em um formulário de uma *View modal*.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

Representação	
Nome	**View Crosstab List**
Descrição	Este padrão representa as interfaces de usuário que são utilizadas para apresentar e editar dados em referência cruzada. Os dados estão vinculados a uma ou mais classes. Caso tenha um filtro associado, este filtro contém as restrições dos dados que serão aplicadas ao carregar a lista. A lista tem a particularidade de ser do tipo grade. Ele é composto de uma *View*, um *List* para a referência cruzada e uma *List* para edição ou inserção e opcionalmente um *Filter*. Podemos ter as *Actions* de CRUD para uma célula. A edição ou

	inserção é feita em um formulário de uma *View modal*.
Atributos	**Name**: Deve ter o nome da *view*. M**enuStructure**: A estrutura para compor o *menu*. **Actions**: {Exit} esta é a *action* possível.

APÊNDICE D METAMODELO COMPLETO

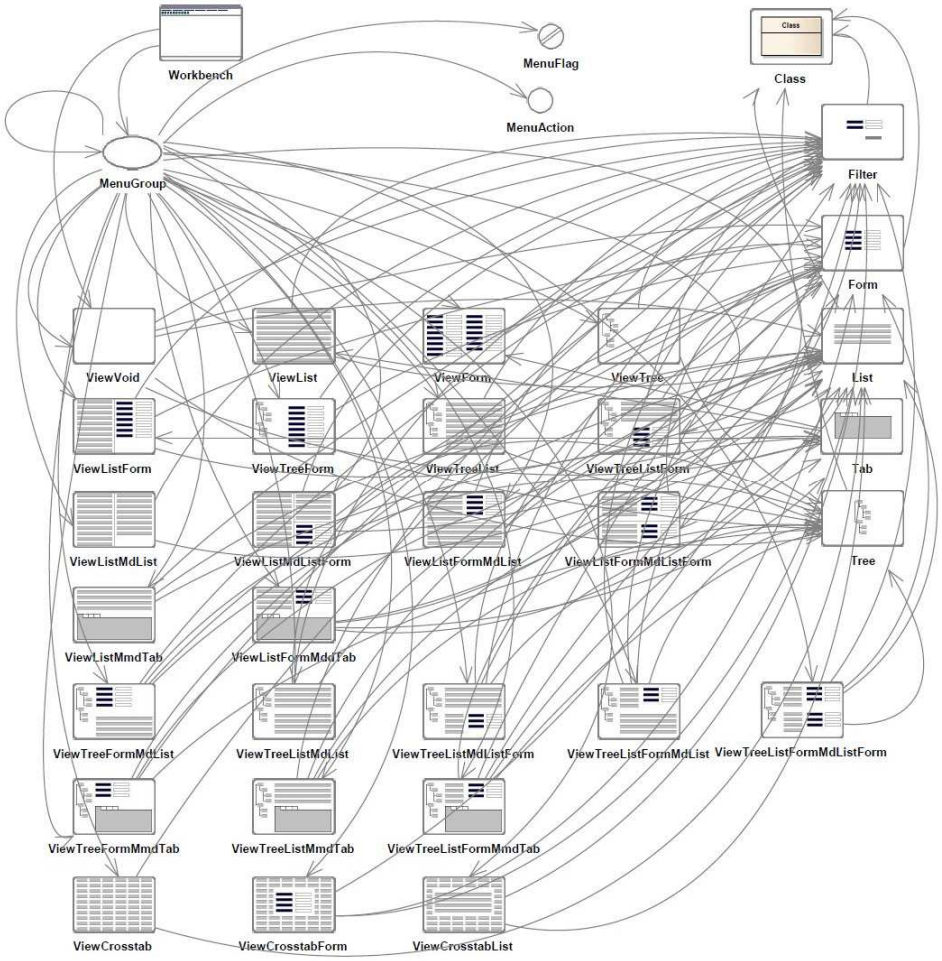
Considerando que cada elemento de *interface* do usuário pode ser considerado como um vértice e que o relacionamento entre os elementos podem ser considerados como arestas, foi criado o metamodelo como uma estrutura de grafos. Assim cada nó representa um elemento possível da modelagem e as arestas são as relações possíveis entre estes elementos. As restrições são definidas nas arestas e os atributos são definidos tanto nas arestas como em cada nó, dependendo do tipo de atributo.

Os elementos que são utilizados como vértices estão definidos no Apêndice C e foram construídos a partir da normalização dos sistemas, que está descrita no Apêndice B. Os sistemas analisados estão descritos no Apêndice A.

O metamodelo completo é uma figura de grafos composta com todos os vértices e arestas que o torna difícil de compreender. A Figura D.1 apresenta este metamodelo. Para facilitar sua compreensão ele será apresentado por visões, separadas de acordo com os padrões do Apêndice C.

Os atributos são colocados em uma visão separada também.

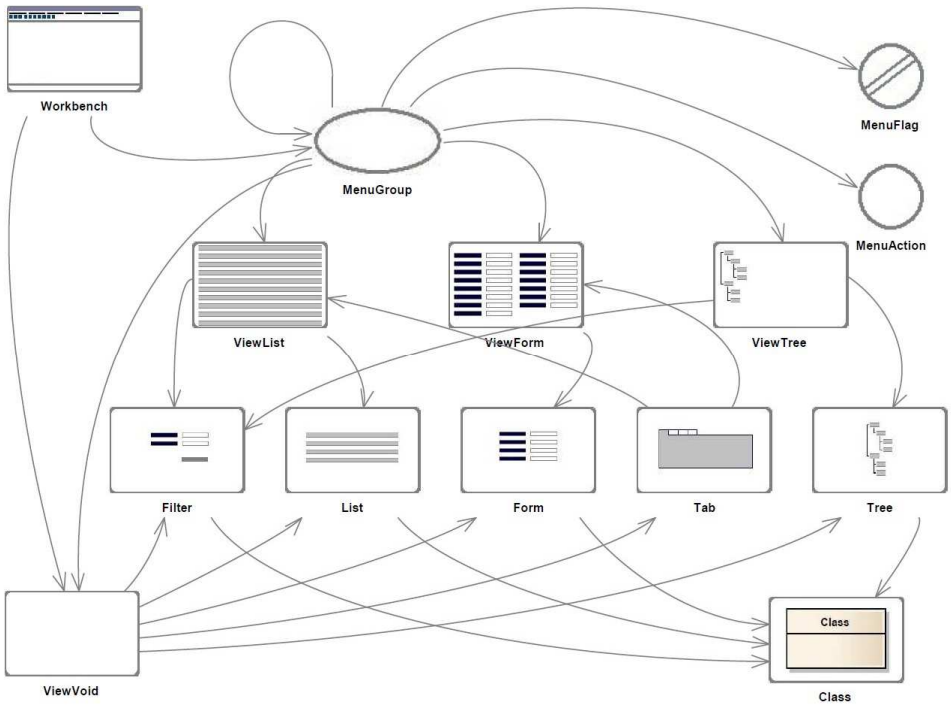
Figura D.1: Metamodelo Completo.



Metamodelo com uma View (One View Graph)

A Figura D.2 têm visão do metamodelo que apresenta o relacionamento dos vértices com as *Views* e *Single Views* de um único elemento.

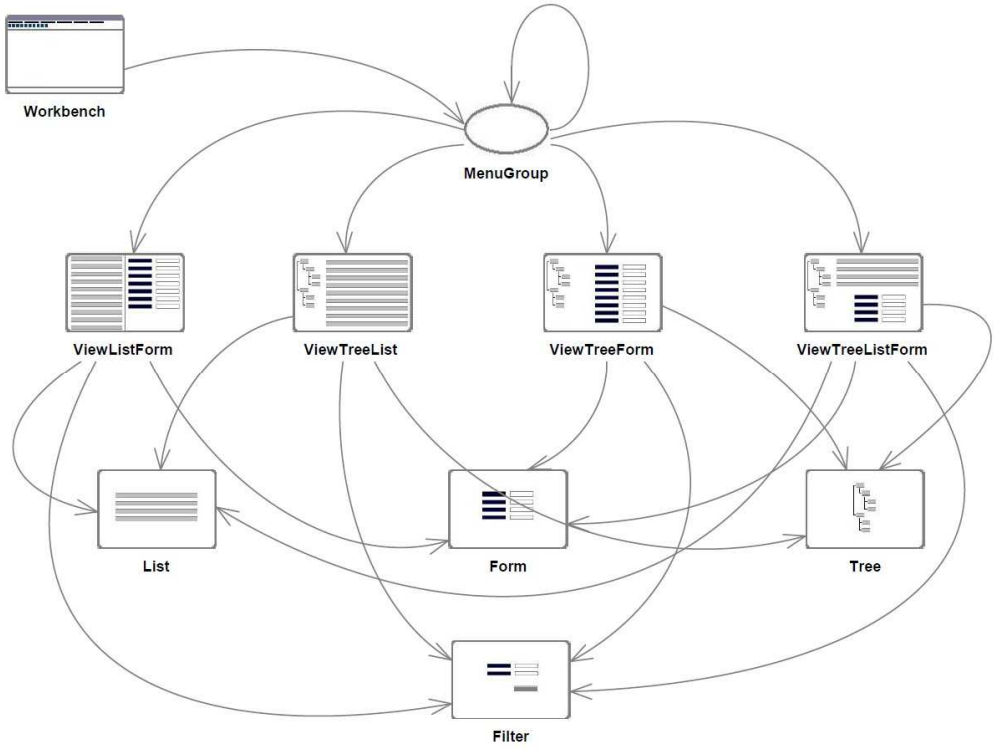
Figura D.2: Metamodelo *One View Graph.*



Metamodelo com duas Views (Double View Graph)

A Figura D.3 apresenta a visão do metamodelo com o relacionamento dos vértices que tem as *Views* com os *Single Views* de dois elementos.

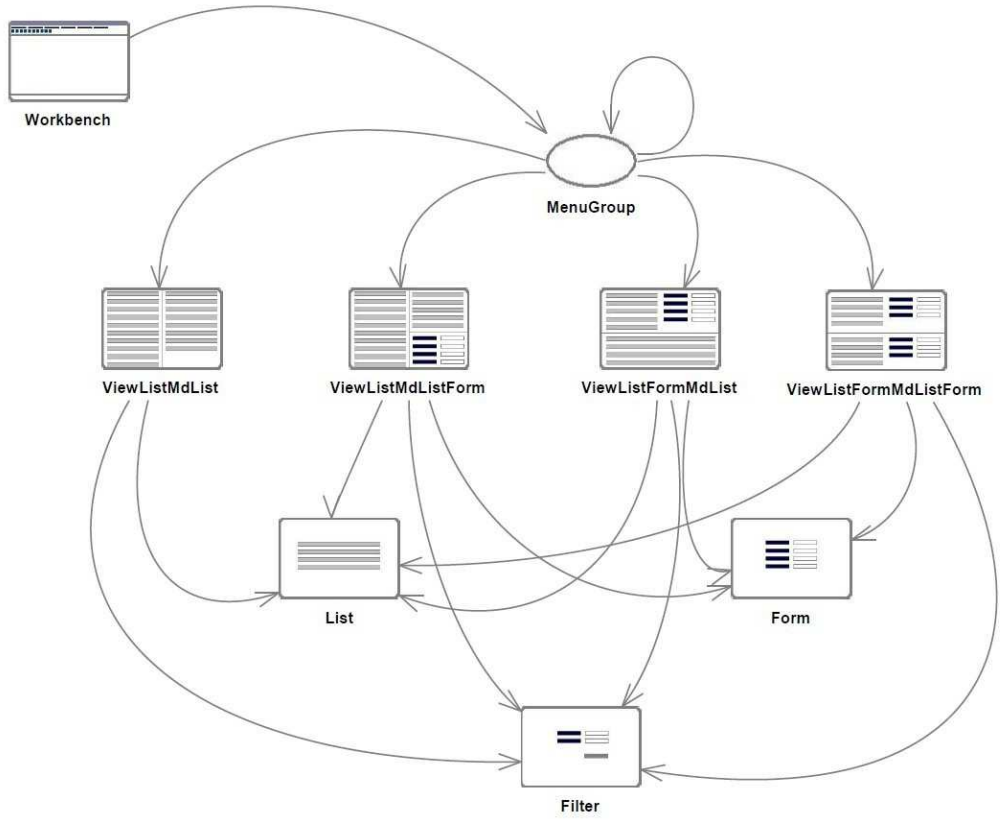
Figura D.3: Metamodelo *Double Views Graph.*



Metamodelo com Composite List Master (List Master Graph)

Na Figura D.4 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views* com os *Master Detail* e onde o *master* é uma Lista.

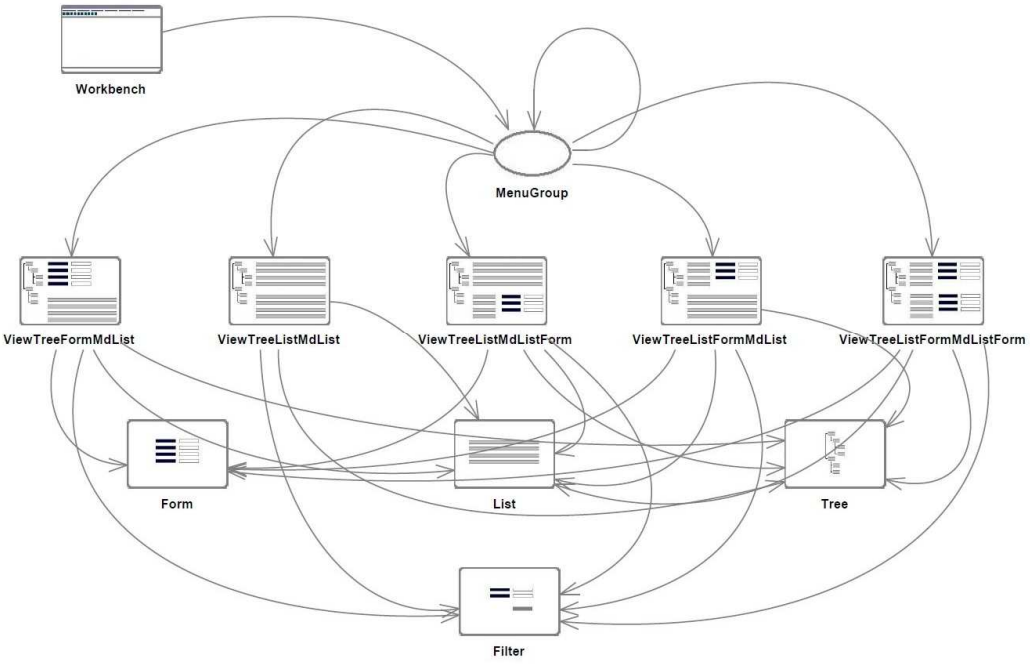
Figura D.4: Metamodelo *List Master Graph*.



Metamodelo com Composite Tree Master (Treet Master Graph)

Na Figura D.5 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views* com os *Master Detail* e onde o *master* é uma Árvore.

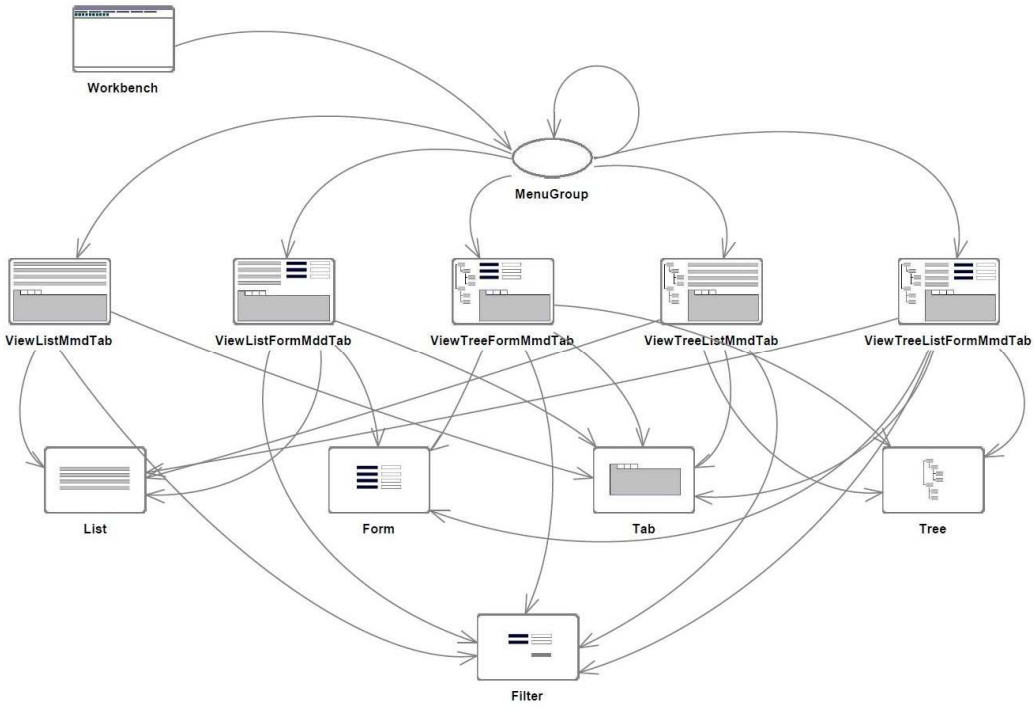
Figura D.5: Metamodelo *Tree Master Graph.*



Metamodelo com Multi Composite (Master Multi Graph)

Na Figura D.6 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views* com os *Master multi Detail*.

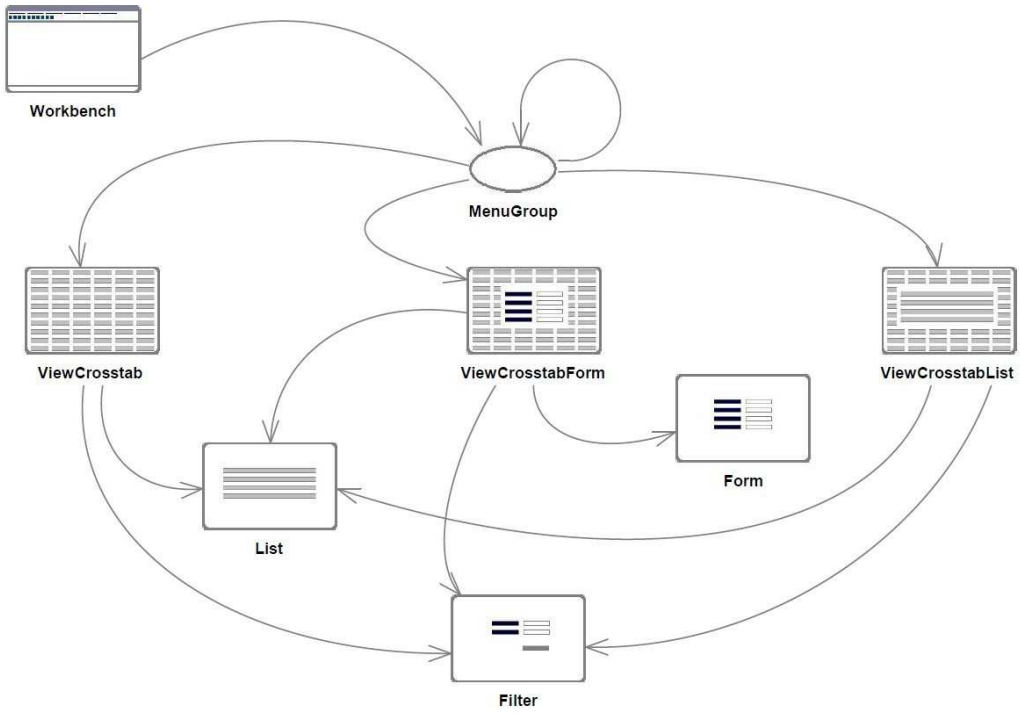
Figura D.6: Metamodelo *Master Multi Graph.*



Metamodelo Crosstab (Crosstab Graph)

Na Figura D.7 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views Crosstab.*

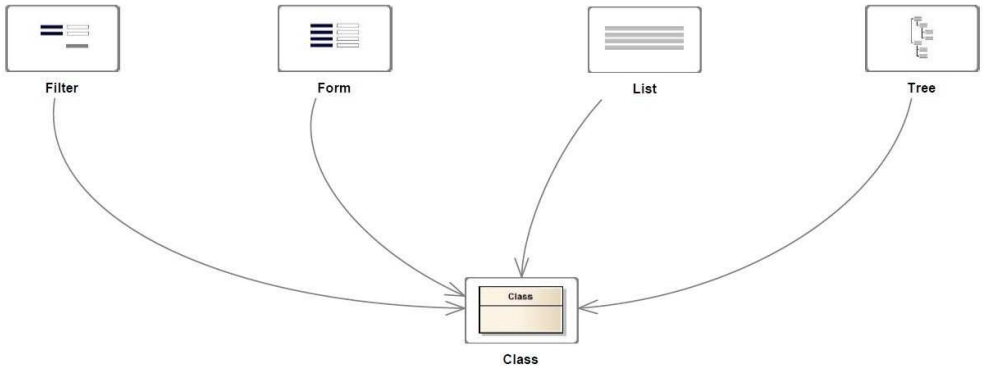
Figura D.7: Metamodelo *Crosstab Graph.*



Metamodelo Elementos Primitivos (Primitive Graph)

Na Figura D.8 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views* primitivas.

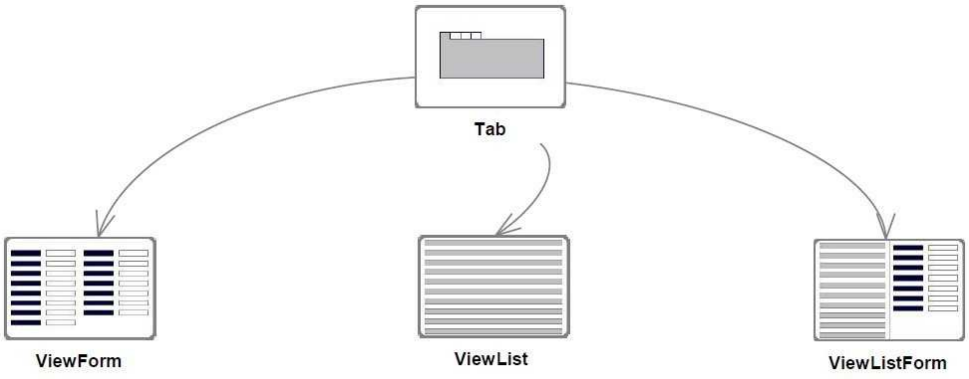
Figura D.8: Metamodelo *Primitive Graph*.



Metamodelo Elementos Primitivos Tab (Primitive Tab Graph)

Na Figura D.9 é apresentado uma visão do metamodelo com o relacionamento dos vértices que tem as *Views Tab*.

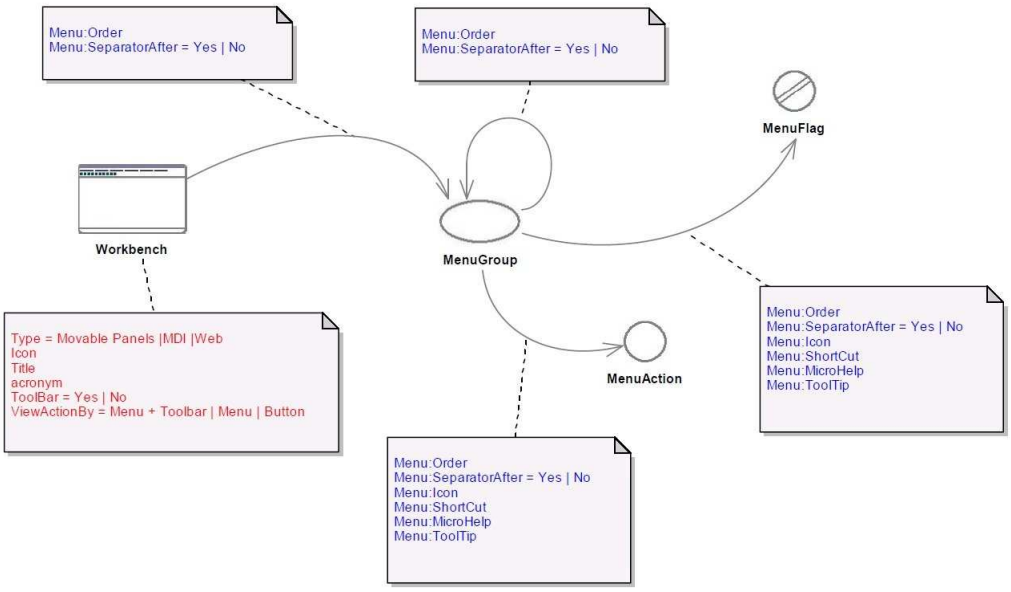
Figura D.9: Metamodelo *Primitive Tab Graph.*



Metamodelo com Atributos (Attributes Graph)

Na Figura D.10 é apresentado uma visão do metamodelo com os atributos dos vértices e das arestas para o *Workbench*, *MenuGroup*, *MenuFlag* e *MenuAction*.

Figura D.10: Metamodelo *Attributes Graph*.



Metamodelo com Atributos de ações (Attribute Actions Graph)

Na Figura D.11 é apresentado uma visão do metamodelo com os atributos dos vértices e das arestas para o *ViewForm*, *ViewList*, *ViewTree* e *ViewVoid*.

Figura D.11: Metamodelo *Attribute Actions Graph*.