UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

GERAÇÃO AUTOMÁTICA DE PLANILHAS A PARTIR DE MODELOS CONCEITUAIS

Léo Carvalho Ramos Antunes

Orientadores
Prof. Márcio de Oliveira Barros
Prof. Alexandre Luis Corrêa

RIO DE JANEIRO, RJ – BRASIL

OUTUBRO DE 2013

GERAÇÃO AUTOMÁTICA DE PLANILHAS A PARTIR DE MODELOS CONCEITUAIS

Léo Carvalho Ramos Antunes

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Prof. Márcio de Oliveira Barros, D.Sc. (UNIRIO)

Prof. Alexandre Luis Corrêa, D.Sc. (UNIRIO)

Prof. Leonardo Guerreiro Azevedo, D.Sc. (PPGI-UNIRIO; IBM Research – Brasil)

Prof. Alessandro Fabricio Garcia, D.Sc. (PUC-Rio)

Antunes, Léo Carvalho Ramos.
A636 Geração automática de planilhas a partir de modelos conceituais / Léo Carvalho Ramos Antunes, 2013.
100 f. ; 30 cm

Orientador: Márcio de Oliveira Barros.
Coorientador: Alexandre Luis Corrêa.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

1. Planilhas eletrônicas. 2. Modelagem conceitual. 3. Model-driven engineering. I. Barros, Márcio de Oliveira. II. Corrêa, Alexandre Luis. III. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. IV. Título.

CDD - 005.54

Dedico esta dissertação ao meu pai, Sebastião Antunes da Silva *(in memoriam)*, pelo amor incondicional e apoio que me concedeu em todos os momentos.

Agradecimentos

A caminhada do mestrado não foi nada fácil em vários sentidos e eu gostaria de deixar registrado nesse pequeno trecho os meus sinceros agradecimentos a todos aqueles que participaram desta fase e que me auxiliaram para que eu pudesse chegar ao fim.

À Deus em primeiro lugar, pois sem Ele eu não teria condições de alcançar nenhuma das minhas metas e sonhos.

Aos meus orientadores, Márcio Barros e Alexandre Correa, pela orientação completa e inquestionável e pela forma como guiaram este trabalho, mesmo em momentos tão difíceis. Tenho plena admiração por ambos, pois além da orientação acadêmica eles me ensinaram valores de caráter e profissionalismo que quero levar para a vida toda.

À minha linda esposa Nathália, pelo amor e pelos momentos de compreensão, incentivo e paciência, concedidos durante todo este período, principalmente na etapa final desse trabalho. Também à sua mãe Edemilce e sua avó, Eudinéia, que me receberam em sua família como um filho e sempre torceram por mim, orando e me ajudando em todos os momentos.

A minha família, meus pais, Sebastião *(in memoriam)* e Léa e minha irmã, Laís, pelo apoio e incentivo concedidos em toda a minha caminhada acadêmica e em toda a minha vida. Ainda pela compreensão pelos momentos de ausência nesta fase tão difícil da nossa família. Também aos meus avós, Ozéas *(in memoriam)*, Celita, Rosa *(in memoriam)* e Alino e minha madrinha, Solange, pelo apoio e compreensão nos momentos de ausência.

Ao professor Leonardo Azevedo, pela excelente orientação que concedeu durante o projeto final e na escrita de artigos que em muito me auxiliaram no andamento do curso. Além disso, pela amizade que construímos durante este período. Ainda por todos os comentários feitos nos seminários e por ter aceitado participar da banca de avaliação deste trabalho.

Ao professor Alessandro Garcia, por ter aceitado participar da banca de avaliação deste trabalho.

Ao professor Gleison Santos, por todos os comentários feitos nos seminários de acompanhamento.

Ao meu chefe, João Carlos, pela compreensão, amizade e pelos períodos concedidos para que eu pudesse concluir este trabalho. Ainda aos meus colegas de

trabalho, Luis Gustavo, Julio Cabezas e Silvia Benza pela compreensão e auxílio nos momentos de ausência.

Aos amigos, pela compreensão nos momento de ausência.

Finalmente, a todos os professores e funcionários da UNIRIO, pelo trabalho, atenção e tratamento concedidos durante todo o curso.

Antunes, Léo Carvalho Ramos. **Geração Automática de Planilhas a Partir de Modelos Conceituais**. UNIRIO, 2013. 100 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

Planilhas eletrônicas oferecem flexibilidade e facilidade de uso, permitindo que usuários com pouco treinamento ou conhecimento em Computação possam construir soluções computacionais para os seus problemas. Porém, essas características permitem que usuários inexperientes construam planilhas contendo erros. Um dos aspectos comumente empregados para facilitar o desenvolvimento de sistemas de software é a modelagem. Modelos de sistemas possibilitam a visualização, a comunicação e a validação de aspectos de um sistema antes que maiores esforços sejam investidos na sua construção. De forma análoga, modelos podem ser úteis no desenvolvimento de planilhas, permitindo a construção de soluções mais resistentes à introdução de erros e com menor esforço. Este trabalho propõe uma abordagem baseada em *Model-Driven Enginnering (MDE)* para gerar planilhas automaticamente a partir de modelos conceituais com o objetivo de reduzir o número de erros na planilha resultante. Foi realizado um estudo experimental com estudantes e profissionais de Computação para avaliar os erros cometidos nas planilhas. Resultados apontam indícios de que a abordagem é capaz de produzir planilhas resistentes à introdução de diversos tipos de erros cometidos por usuários finais com diferentes perfis e tempo de experiência.

Palavras-chave: *planilhas eletrônicas, modelos conceituais, MDE.*

ABSTRACT

The flexibility and ease of use of spreadsheets allow users with little training or computing knowledge to build computational solutions to solve their problems. On the other hand, these features may lead less experienced users to build erroneous spreadsheets. Modeling is an approach commonly used in software development. Models allow users to visualize, communicate and validate different aspects of a system before starting its construction. Models can also be useful to build less error-prone spreadsheets with less effort. This work proposes a Model-Driven Engineering (MDE) approach that automatically generates spreadsheets from conceptual models aiming at reducing the number of errors usually found in spreadsheets. An empirical study with students and Information Systems practitioners was performed and its results has shown evidence that the proposed approach can avoid the introduction of several types of errors often present in spreadsheets manually created by users with different background and experience.

Keywords: *spreadsheets, conceptual models, MDE*.

SUMÁRIO

Capítulo I – Introdução

1.1 Motivação

Planilhas eletrônicas foram introduzidas na década de 80, com a aplicação chamada *VisiCalc* (O'Donovan, 1984). Depois vieram outras, como o *SuperCalc*, *Lotus 1-2-3* (Bookbinder, 1989), *Quattro Pro* e atualmente o *Microsoft Excel* (Campbell, 1985; O'Leary, 2008) é a principal ferramenta proprietária desse segmento. Além disso, o componente *Calc* da *suite* OpenOffice.org (Riley 2009) pode ser visto como uma alternativa *open-source* desse tipo de aplicação.

Planilhas são muito utilizadas por programadores não-profissionais para desenvolver aplicações de negócio. Esses programadores também são chamados de *programadores usuários finais* (*end-userprogrammers - EUP*). Um *EUP* pode ser, por exemplo, um professor, um engenheiro, um físico, uma secretária, um contador ou um gerente. De fato, qualquer um que não seja um programador profissional pode ser considerado um *EUP.* O interesse de um *EUP* na programação de computadores normalmente está limitado a completar uma tarefa; em geral, ele não está interessado na programação em si (Cunha, 2010). O que faz os *EUPs* diferentes de programadores profissionais são seus objetivos: profissionais são pagos para construir e manter sistemas de software ao longo do tempo; *EUPs*, por outro lado, escrevem programas para ajudá-los a atingir algum objetivo no âmbito de sua própria especialidade (Ko *et al.*, 2011).

O número potencial de *EUPs* é bem maior que o número de programadores profissionais. Um estudo realizado por Scaffidi *et al.*(2005) mostra que, só nos EUA, o número de *EUPs* foi estimado em 11 milhões, comparado a 2,75 milhões de programadores profissionais (à época). Esse fator sugere que as linguagens e ferramentas utilizadas na construção de planilhas eletrônicas são também um alvo potencial para a aplicação de princípios de linguagens de programação tradicionais e da Engenharia de Software.

Planilhas eletrônicas oferecem flexibilidade e facilidade de uso, permitindo que usuários com pouco treinamento ou conhecimento em Computação possam construir soluções computacionais para os seus problemas. O lado ruim dessas características é que elas permitem que usuários inexperientes construam planilhas contendo erros. Vários estudos relatam que mais de 90% das planilhas criadas contém erros (Panko, 2000; Rajalingham *et al.*, 2001; Powell e Baker, 2003; EuSpRIG, 2011). Esses erros podem ser introduzidos não apenas durante a criação de uma planilha, mas também quando ela é modificada. O problema se torna ainda maior quando o usuário precisa modificar uma planilha construída por outros, sem possuir conhecimento completo das suas funcionalidades. Esses fatores tornam sua evolução difícil e propensa a erros. Além disso, alguns desses erros podem ter alto impacto na produtividade de uma empresa (Croll, 2007; 2009), podendo levá-la a perder recursos financeiros e produtivos significativos (EuSpRIG, 2011). Para citar um exemplo, o Sistema Bancário da Jamaica entrou em colapso no fim da década de 90, em parte devido ao uso de planilhas e a problemas em gerenciá-las e controlá-las (Lemieux, 2002; 2008).

A necessidade de apoiar o desenvolvimento de software pelos usuários finais, área conhecida como *End-User Software Engineering*, já foi reconhecida como um campo de pesquisa dentro da Engenharia de Software há algum tempo. Algumas iniciativas, relatadas em Ko *et al.* (2011), procuram investigar como abordagens mais rigorosas da Engenharia de Software podem ser aplicadas nesse contexto, já que, embora esses usuários não tenham os mesmos objetivos que os desenvolvedores de software profissionais, eles enfrentam desafios similares, incluindo o entendimento dos requisitos e questões relacionadas a decisões sobre projeto, reutilização, integração, teste e correção de defeitos.

Um dos aspectos comumente empregados para facilitar o desenvolvimento de sistemas de software é a modelagem. Modelos de sistemas possibilitam a visualização, a comunicação e a validação de aspectos de um sistema antes da sua construção. Um modelo de sistema é representado em uma linguagem de modelagem, que pode empregar uma combinação de representações gráficas e textuais (Rumbaugh *et al.*, 2004). Modelos podem ser construídos em diferentes níveis de detalhe, que variam desde uma representação do domínio onde o problema que se pretende resolver se manifesta, até a representação de uma solução computacional que aborde o problema. Um modelo conceitual é um modelo que permite entender e simplificar um problema (Fowler, 1997). Esse modelo pode ser representado de forma explícita, em uma linguagem de alto nível, visando descrever

os requisitos dos usuários, incluindo, por exemplo, os conceitos, atributos, relacionamentos e restrições do domínio. Como esses conceitos não incluem detalhes de implementação, normalmente são mais fáceis de entender, possibilitando seu uso na comunicação com usuários não técnicos (Elmasri e Navathe, 2009).

O trabalho para industrializar o desenvolvimento de software tem sido motivado pelo crescimento contínuo da complexidade dos softwares produzidos. Em particular, pesquisas na área de *Model-Driven Engineering* (MDE) estão preocupadas com a redução da distância entre a representação do domínio do problema e as suas implementações, buscando desenvolver tecnologias que apoiam transformações sistemáticas de representações abstratas do problema em implementações concretas de software. Na visão MDE de desenvolvimento de software, modelos são os principais artefatos e serão transformados em sistemas executáveis por meio de tecnologias especialmente desenvolvidas para esse fim (France e Rumpe, 2007).

1.2 Objetivo da Dissertação

O principal objetivo deste trabalho consiste em gerar planilhas automaticamente a partir de um modelo conceitual do problema que elas visam resolver. Essa geração é realizada por meio de uma abordagem baseada em *Model-Driven Enginnering (MDE)*. Para materializar o modelo conceitual que serve como insumo para a geração da planilha, várias representações podem ser utilizadas, como diagramas de classes UML *(Unified Modeling Language)*, modelos entidade-relacionamento (MER) e Ontologias.

No contexto de desenvolvimento de software orientado a objetos, a UML – *Unified Modeling Language* (Booch *et al.*, 2005) é, atualmente, o padrão *de facto* para a elaboração de modelos, sendo bastante difundida tanto na indústria como na academia. Resultado do processo de unificação de linguagens de modelagem propostas por diferentes autores (Rumbaugh, 1990) (Jacobson, 1992) (Booch, 1994), a UML foi adotada como padrão pelo OMG – *Object Management Group* (OMG, 1999) que, desde então, assumiu a responsabilidade por sua evolução.

A forma mais comum de utilização da UML consiste na elaboração de modelos contendo representações gráficas semiformais, empregando os diversos diagramas definidos pela linguagem, complementadas com anotações elaboradas em linguagem natural (Correa, 2006). Em cenários que demandam maior precisão dos modelos, a OCL – *Object Constraint Language* (Warmer e Kleppe, 2003) – é uma linguagem que pode ser utilizada para especificar, de forma precisa, elementos que as notações

gráficas da UML não são capazes de representar, como restrições, expressões associadas a atributos derivados e expressões de consulta. Em sua primeira versão, a OCL era parte integrante da especificação da UML. No entanto, a partir da versão 2.0, ela passou a ter uma especificação própria (OMG, 2003a).

Neste trabalho, diagramas de classe UML foram escolhidos para representar os conceitos, propriedades, relacionamentos e operações contidos no domínio para o qual se deseja construir uma planilha. Sendo assim, propomos uma abordagem que possibilita a geração da estrutura da planilha a partir da transformação dos diagramas de classe da UML. Para isso, foram definidas regras de transformação dos elementos presentes no diagrama (classes, atributos, associações e operações) em elementos componentes da planilha (abas, tabelas e colunas), gerando assim uma planilha que corresponde ao conhecimento expresso no modelo. Além disso, a OCL é utilizada na especificação da semântica das operações presentes nos diagramas de classes, que serão transformadas em fórmulas da planilha. Para isso, também foram definidas regras de transformação de expressões OCL em fórmulas da planilha.

Com relação à construção do modelo conceitual, esta abordagem está direcionada a usuários com conhecimentos técnicos de diagramas de classe UML e especificação OCL. Porém, uma vez que estes modelos estejam construídos, completos e corretos, a abordagem pode ser utilizada por usuários finais de planilhas, com o intuito de gerar a planilha resultante. Como resultado da utilização da abordagem, almejamos reduzir o número de erros presentes na planilha resultante.

1.3 Organização da Dissertação

Esta Dissertação está organizada em cinco capítulos, que incluem este capítulo introdutório. No capítulo 2, são fornecidos os conceitos que proporcionam o embasamento teórico necessário à compreensão do trabalho aqui descrito, bem como os trabalhos relacionados com esta Dissertação. O capítulo 3 apresenta a solução proposta, detalhando as transformações realizadas a partir dos diagramas de classe UML e expressões OCL para produzir os elementos e fórmulas da planilha. O capítulo 4 apresenta um estudo experimental que avaliou os erros que usuários eventualmente cometem ao desenvolver uma planilha eletrônica e uma discussão sobre os erros que a solução proposta pretende evitar. O capítulo 5 apresenta as conclusões desta Dissertação, com um resumo do trabalho efetuado, suas contribuições, limitações da proposta apresentada e sugestões de trabalhos futuros.

Capítulo II – Revisão Bibliográfica

Este capítulo descreve os principais trabalhos relacionados aos conceitos utilizados na proposta de geração de planilhas a partir de modelos, e está estruturado em quatro seções. A seção 2.1 apresenta uma breve introdução sobre planilhas e os trabalhos relacionados com o objetivo de diminuir a incidência de erros durante a sua construção e manutenção. A seção 2.2 apresenta uma breve introdução à estrutura utilizada pelo OMG para a definição de modelos e metamodelos, como os metamodelos da UML e da OCL, por exemplo. A seção 2.3 descreve as principais características da OCL. Por fim, a seção 2.4 apresenta as considerações finais sobre este capítulo.

2.1 Planilhas Eletrônicas

Uma planilha é um documento digital criado a partir de uma aplicação de software específica. Esse documento simula uma folha de cálculo composta por múltiplas células que juntas formam uma matriz de linhas e colunas. Cada célula dessa matriz pode conter um texto alfanumérico, um valor numérico ou uma fórmula. Uma fórmula define como o valor de uma célula é calculado a partir do conteúdo de outras células ou de valores constantes. Por exemplo, uma fórmula pode definir uma célula como o somatório dos valores das células de uma determinada coluna. Uma célula pode ainda ser definida por uma referência a outra, possuindo, nesse caso, o mesmo valor da célula referenciada. Uma das funcionalidades mais interessantes das planilhas é a habilidade de recalcular automaticamente toda a matriz após uma modificação ser aplicada a uma célula (Cunha, 2011).

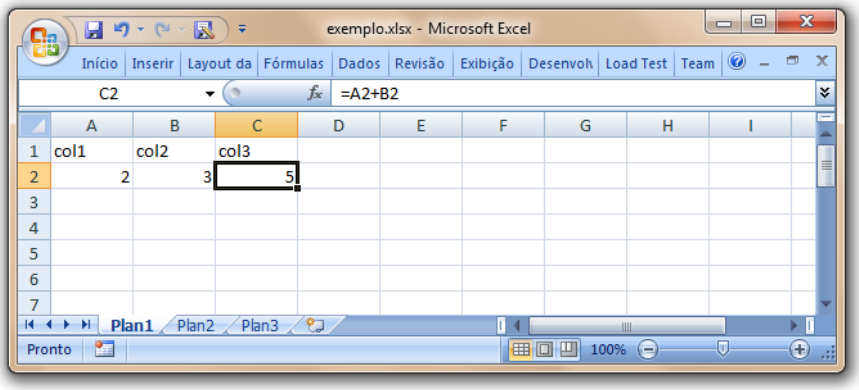


Figura 1 – Exemplo de planilha

Em uma planilha, células são tipicamente identificadas pelo par coluna (normalmente uma letra maiúscula) e linha (normalmente um número). Na planilha da Figura 1, as células *A1*, *B1* e *C1* contêm valores constantes, representados pelos textos alfanuméricos *col1*, *col2* e *col3*, respectivamente. As células *A2* e *B2* contêm os valores constantes numéricos *2* e *3*, respectivamente. Já a célula *C2* contém uma fórmula que define o valor da célula como a soma dos valores das células *A2* e *B2*, conforme indicado no painel de funções acima da matriz de células.

2.1.1 Taxonomia de Erros em Planilhas

Planilhas oferecem flexibilidade e permitem que seus usuários construam soluções computacionais sem conhecer linguagens de programação ou outros recursos utilizados por Engenheiros de Software profissionais. Porém, essa liberdade aumenta a possibilidade de que sejam criadas planilhas com erros. Na direção de melhor entender os tipos de erros mais comuns presentes em planilhas, Panko e Aurigemma (2010) propuseram uma taxonomia de erros tipicamente encontrados em planilhas, apresentada na Figura 2.

O primeiro nível da classificação divide os erros em dois grupos: intencionais e não intencionais. Erros intencionais são violações de padrões e regras de desenvolvimento de planilhas da empresa, com o intuito de prejudicar ou favorecer interesses pessoais. Erros não intencionais são introduzidos por falhas humanas. Os erros não intencionais são divididos em erros quantitativos e qualitativos.

Erros quantitativos são fórmulas ou células de dados incorretas que tornam o modelo incorreto. Eles são ainda divididos em erros de planejamento e de execução. A distinção entre um e outro está no instante em que o usuário começa a desenvolver a planilha. Um erro cometido antes desse instante é um erro de planejamento. Um erro cometido após esse instante é um erro de execução. Os erros de planejamento

são divididos em erros de planejamento de domínio e de planilha. Essa divisão considera que o planejamento tem dois aspectos: as fórmulas em si e o conhecimento do domínio. Sendo assim, o erro pode ser causado por uma fórmula mal empregada ou pela aplicação errada do conhecimento do domínio. Erros de execução, por sua vez, são divididos em deslizes e lapsos. Deslizes são erros sensoriais ou motores, como um erro de digitação ou uma referência para a célula errada. Lapsos ocorrem quando algo é esquecido, ou seja, um erro de memória do desenvolvedor.

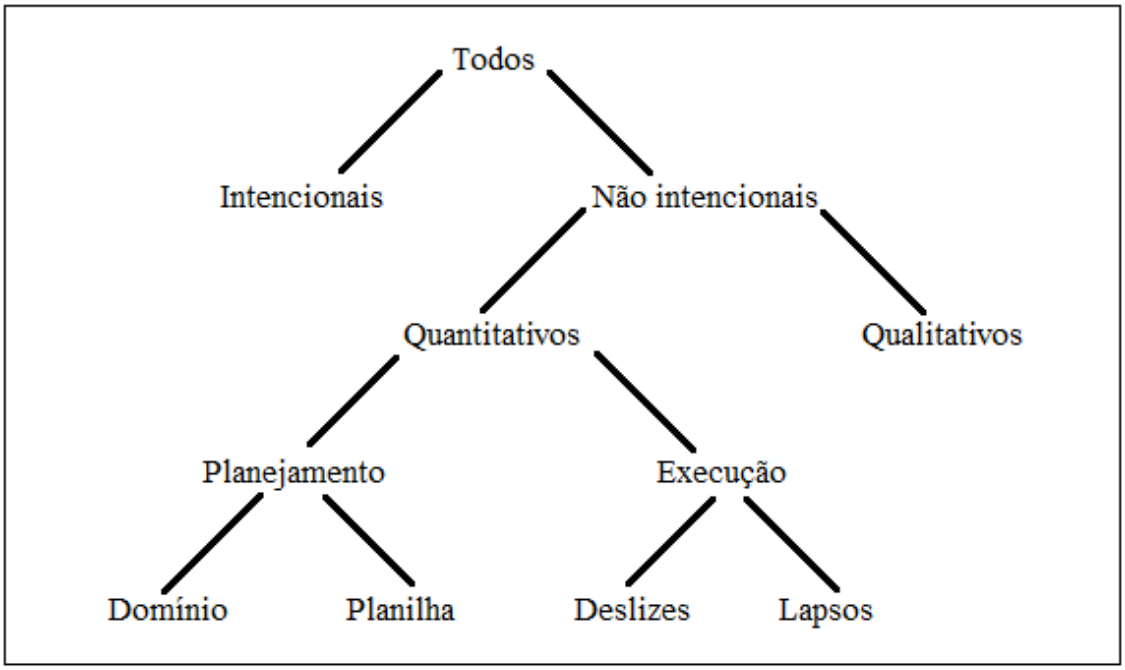


Figura 2 – Taxonomia de erros em planilhas (adaptada de Panko e Aurigemma, 2010)

Erros qualitativos, por sua vez, podem levar a problemas quantitativos mais tarde, mas não tornam o modelo incorreto imediatamente. Tipicamente, esses erros acontecem quando boas práticas de desenvolvimento de planilhas não são seguidas, como a criação de validações de dados para células que exigem valores pré-definidos ou o bloqueio de células que não podem ser editadas.

2.1.2 **Propostas de Apoio ao Desenvolvimento de Planilhas**

Nos próximos tópicos serão apresentadas algumas soluções existentes seja para diminuir a incidência de erros em planilhas, seja para facilitar o entendimento da sua estrutura, bem como algumas lacunas que foram objeto desta Dissertação.

2.1.2.1 **Modelagem de Planilhas**

A ideia de utilizar técnicas de modelagem durante a elaboração de planilhas visando evitar a introdução de erros foi abordada por diversos trabalhos. Ronen *et al.* (1989) propõem uma abordagem estruturada para construção de planilhas. Nessa abordagem, eles sugerem um ciclo de desenvolvimento para a elaboração das planilhas similar a ciclos de vida de desenvolvimento de sistemas. Neste ciclo, são definidas etapas como identificação do problema, construção da planilha, testes, documentação e treinamento, e as transições entre elas. Além disso, eles propõem um formato padrão para a organização da informação e a utilização de diagramas denominados SFD (*Spreadsheet Flow Diagram*) para a especificação das fórmulas.

Rajalingham *et al.* (2000) apresentam um método para elaboração de planilhas composta por técnicas, recomendações e regras adaptadas de conceitos e regras da Engenharia de Software, dentre as quais destacam-se a decomposição hierárquica das fórmulas, a separação entre os dados e as operações e a definição única de elementos na planilha.

Powell e Baker (2003) ressaltam a importância da modelagem e sugerem a utilização de gráficos de influência que representam a definição das fórmulas em função das suas entradas e saídas, de forma similar ao SFD mencionado acima.

Esses trabalhos se relacionam com o presente trabalho por ressaltarem a importância da modelagem da estrutura dos elementos de uma planilha como forma de prevenção de erros. Eles propõem diferentes notações para a modelagem de planilhas, enquanto que o presente trabalho utiliza padrões como a UML e OCL para cumprir esse propósito.

2.1.2.2 **Testes de Planilhas**

Outros trabalhos investigam aspectos ligados a testes e verificação de correção de planilhas. Rothermel *et al.* (1998) apresentam um método de testes em planilhas baseado em técnicas para definição de casos de teste a partir da representação formal das suas fórmulas e de grafos de relacionamento entre células. Foram propostos alguns refinamentos do método (Rothermel *et al.* (2001); Rothermel *et al.* (2000); Burnett *et al.* (2002)) bem como estudos experimentais visando avaliá-lo.

Clermont *et al.* (2008) propõem uma ferramenta para apoiar processos de auditoria em planilhas, visando a identificação de irregularidades ou erros, por meio da visualização da estrutura de planilhas a partir de três tipos de relações de equivalência entre as células. Por exemplo, é possível identificar que todas as células de uma coluna de uma região da planilha são logicamente equivalentes com exceção de uma,

o que indica um provável erro na fórmula da célula divergente. Panko e Aurigemma (2010) propõem uma revisão da taxonomia de erros em planilhas originalmente proposta por Panko e Halverson (2001), com o intuito de auxiliar pesquisadores na identificação e classificação dos diferentes erros encontrados em planilhas.

Além disso, Cunha *et al.* (2012e,i) propõem um método para identificar construções inadequadas (*"bad smells"*) em planilhas. O primeiro passo desse método consiste na definição de um catálogo de construções inadequadas baseado em experiências pessoais dos autores. Essas construções incluem fórmulas que referenciam células vazias, células vazias em tabelas, células que fogem ao padrão das demais células da coluna, fórmulas que diferem das demais fórmulas da coluna, entre outras. Definido o catálogo, as construções inadequadas de uma planilha específica podem ser identificadas, classificadas e destacadas diretamente na planilha, segundo um conjunto representativo de cores.

Os trabalhos supracitados estão focados na correção de planilhas existentes. Eles estão relacionados com o nosso trabalho no que diz respeito aos erros identificados, que serviram de base para a avaliação dos diferentes tipos de erro que podem estar presentes nas planilhas. No entanto, a abordagem apresentada neste trabalho consiste em produzir planilhas evitando a introdução de erros pelos usuários finais, ao invés de identificar erros já presentes em planilhas.

2.1.2.3 **Entendimento da Estrutura de Planilhas**

Alguns trabalhos propõem mecanismos para apoiar o entendimento da estrutura de uma planilha. Davis (1996) propôs uma ferramenta chamada *"Online Data Dependency Tool*" que extrai um diagrama de fluxos da planilha, como proposto por Ronen *et al.* (1989), para fins de documentação. A ferramenta nunca foi implementada, possivelmente pela dificuldade de automatizar o *layout* gráfico dos diagramas propostos. Foi realizada uma avaliação com um diagrama de fluxos da planilha gerado manualmente, e os autores compararam sua utilidade com um diagrama que mostrava as dependências entre células diretamente na planilha, similar à ferramenta *Audit Toolbar* do Excel. Os resultados mostraram que participantes identificaram mais dependências entre células com os diagramas de fluxos da planilha do que sem qualquer diagrama, embora o diagrama que mostrava as dependências entre células diretamente na planilha tenha tido um desempenho ligeiramente melhor.

Clermont (2004) introduziu o conceito de grafo de dependência de dados (*Data Dependency Graph – DDG*), mostrando para os usuários finais a relação entre todas as células da planilha a partir de uma ferramenta de auditoria. Porém, esse conceito

só apresenta nomes de células (A1, por exemplo), não mostrando fórmulas de cálculo. Para preencher essa lacuna, Hermans *et al.* (2011) propõem uma abordagem que apresenta a estrutura de uma planilha com diagramas de fluxo de dados (DFD) de múltiplos níveis. Ela fornece três diferentes visões no fluxo de dados, que permitem ao usuário analisar o DFD de forma *top-down*, sendo especialmente útil no caso em que uma planilha será repassada para outra pessoa não familiarizada com a mesma.

Enquanto esses trabalhos têm o intuito de extrair diagramas de fluxos de dados de fórmulas de planilhas existentes, o presente trabalho visa construir as fórmulas da planilha a partir de expressões OCL. Ambas são representações de mais alto nível das fórmulas, porém os diagramas visam facilitar o entendimento de uma planilha já existente, enquanto que o foco do presente trabalho está na geração automática das fórmulas da planilha.

2.1.2.4 **Edição Segura de Planilhas**

A ideia de evitar erros na edição de planilhas também foi abordada por alguns trabalhos. Cunha *et al.* (2009a, 2012f) destacam a lacuna que existe no campo da organização de dados estruturados em planilhas. Essa falta de estruturação pode levar à redundância de dados, assim como à perda ou corrupção de dados, durante a edição de uma planilha. Os autores propõem mecanismos de assistência de edição para usuários de planilhas a partir das dependências funcionais entre os dados. Esses mecanismos foram adaptados das técnicas de normalização de bancos de dados relacionais com o objetivo de evitar redundâncias e permitir a edição e remoção segura de dados e de suas referências (mesmo com redundância).

Em Cunha *et al.* (2012b) e Cunha *et al.* (2012c) foram realizados estudos empíricos para avaliar a produtividade de usuários e os erros por eles produzidos utilizando os diferentes tipos de assistência propostos. Os resultados encontrados apresentam indícios de que técnicas que utilizem MDE (*Model-Driven Engineering*) podem auxiliar os usuários na reestruturação e edição de planilhas existentes, indicando esse campo como um tópico promissor de pesquisa. Esses trabalhos também forneceram indícios de que técnicas que limitem os passos do usuário, como a inserção de validações de dados, estão relacionadas com a diminuição dos erros encontrados em planilhas.

2.1.2.5 **Transformação de Modelos em Planilhas e Vice-Versa**

Com relação à transformação de modelos envolvendo planilhas, podemos separar os trabalhos reportados nesse contexto de acordo com o sentido da transformação:

transformação de modelos em planilhas ou construção de modelos a partir de planilhas.

Cunha *et al.* (2009b) utilizam a mesma técnica apresentada em Cunha *et al.* (2009a) para extrair dependências funcionais dos dados da planilha e construir um esquema relacional a partir dessas dependências. Para isso, eles apresentam regras que transformam esquemas de bancos de dados relacionais em planilhas e vice-versa. Porém, o trabalho é limitado à estrutura da planilha, i.e., conceitos, atributos e relacionamentos, sem abordar, por exemplo, as fórmulas que descrevem os dados calculados.

Hermans *et al.* (2010) propõem uma abordagem de engenharia reversa para gerar um diagrama de classe UML a partir de uma planilha. O diagrama de classes resultante pode ser utilizado por engenheiros de software para entender, refinar ou reimplementar as funcionalidades da planilha. Os autores propõem uma abordagem de transformação baseada em padrões predefinidos de planilhas. Além de tratar somente da engenharia reversa de planilhas, a abordagem proposta também não aborda a transformação das fórmulas, mas somente dos relacionamentos entre as classes.

Engels e Erwig (2005) introduziram o conceito de *ClassSheets*, que consiste em um modelo para expressar a lógica de negócio representada nos dados de uma planilha. Esse modelo é a base de uma abordagem de desenvolvimento de planilhas baseada em modelos. A lógica de negócio é definida através de um formalismo abstrato e, como resultado, os usuários podem entender, manter e evoluir planilhas complexas analisando a sua estrutura por meio desses modelos (*ClassSheets*), sem precisar inferi-la a partir dos seus dados.

A Figura 3 apresenta um *ClassSheet* de uma planilha de voos de uma companhia aérea bem como a planilha resultante da transformação desse modelo. Esse modelo tem o intuito de representar os voos armazenando as informações do voo, de seus pilotos e aviões. Eles representam as informações relativas aos voos no cruzamento entre os conceitos piloto e avião (origem, destino, data e hora do voo).

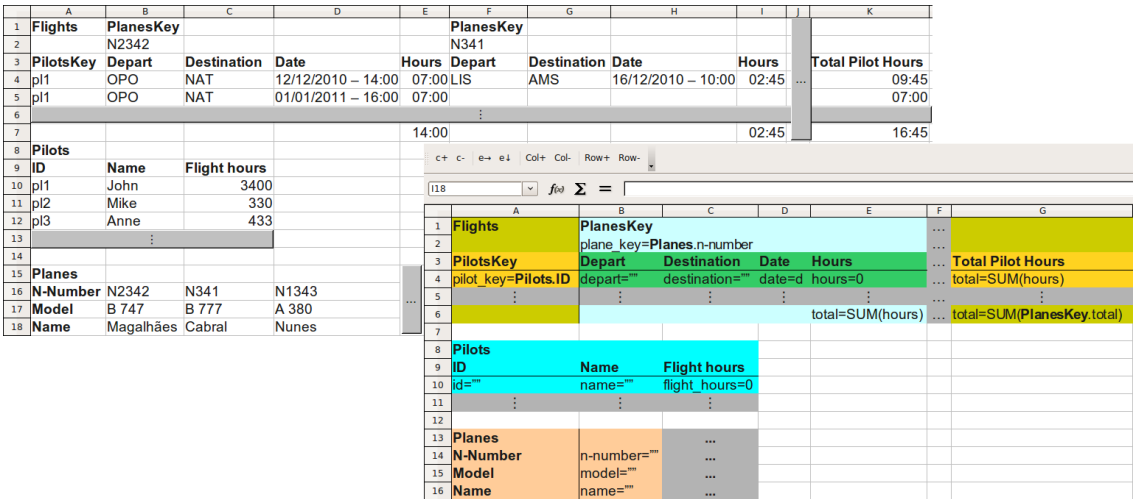


Figura 3 – Exemplo de ClassSheet e planilha correspondente

A proposta dos autores é que a estrutura da planilha seja sempre modificada a partir do modelo (*ClassSheet*). Além disso, a inserção dos dados é orientada por botões específicos, de modo a garantir a consistência dos dados. Esse trabalho é o que mais se aproxima do que realizamos em nossa proposta, pois visa a transformação de um modelo em uma planilha. Uma das principais diferenças é que a planilha gerada está diretamente conectada a uma infraestrutura para manter a consistência dos dados, muito diferente do que usuários de planilhas estão acostumados a lidar. Além disso, o foco do trabalho está na estrutura da planilha, sendo o suporte a fórmulas ainda muito simples. A elaboração de fórmulas mais complexas ainda demanda dos usuários um bom conhecimento em planilhas. Ainda, deve ser considerado o tempo necessário para o entendimento do padrão de representação dos *ClassSheets* de forma que se possa criar uma planilha. Nosso trabalho é baseado em modelos elaborados em UML, que é um padrão *de-facto* para representação gráfica de modelos de software. Além disso, o uso de recursos de modelagem como a OCL contribui para reduzir a complexidade de implementação de planilhas mais elaboradas, como, por exemplo, aquelas contendo fórmulas que envolvem associações entre classes.

O conceito de *ClasSheets* serviu como base para diversos outros trabalhos recentes no âmbito de leitura, criação e manutenção de planilhas eletrônicas realizadas por um grupo de pesquisa da Universidade do Minho, em Portugal, chamado *SSaaPP* (*SpreadSheets as a Programming Paradigm*). Três principais tópicos foram desenvolvidos: a evolução de planilhas (Cunha *et al.*, 2011a; Mendes, 2012a; Cunha *et al.*, 2011d; Cunha *et al.*, 2012b; Cunha *et al.*, 2012c; Cunha *et al.*,

2012d); consultas em planilhas (Cunha *et al.*, 2013b; Belo *et al.*, 2013) e transformação (Cunha *et al.*, 2010;Cunha *et al.*, 2012a; Cunha *et al.*, 2012h).

O primeiro tópico envolve trabalhos que visam acompanhar a evolução das planilhas com base no modelo (*ClassSheets*), ou seja, manter o modelo e a planilha consistentes através de regras de transformação bidirecionais. A ideia é garantir a evolução segura da planilha quando o modelo ou a planilha são modificados, mantendo o sincronismo entre eles. Cunha *et al.* (2012b-c) apresentam um framework para criar, evoluir e inferir *ClassSheets* no mesmo ambiente de construção da planilha. O problema desse tipo de abordagem é que o modelo está vinculado à planilha e não pode ser reaproveitado para a construção de outros tipos de aplicação.

O segundo tópico tem o intuito de criar mecanismos de consulta a dados na planilha de forma declarativa baseados no modelo da planilha. Cunha *et al.* (2012c) propõem uma técnica e uma linguagem onde consultas podem ser definidas em um formato semelhante à SQL (Structured Query Language), linguagem de consultas a bancos de dados, e processadas de forma a retornar os dados da planilha. Belo *et al.*(2013) apresentam a ferramenta *QuerySheet*, responsável por fornecer o ambiente para construção e execução das consultas.

Em uma extensão do trabalho de Cunha et al. (2009), o grupo apresenta trabalhos para transformação de planilhas em *ClassSheets* (Cunha *et al.*, 2010) e a transformação de *ClassSheets* em diagramas UML com restrições OCL (Cunha *et al.*, 2012a). Este segundo, mais ligado à proposta do nosso trabalho, visa a transformação de *ClassSheets* em diagramas de classe e restrições OCL simples com invariantes, mas não aborda operações de consulta, além de ser restrito à engenharia reversa.

Como uma extensão dos modelos de planilhas (*ClassSheets*), Cunha *et al.* (2012h) propõem definições de regras e expressões regulares nos *ClassSheets* para auxiliar na edição das planilhas. Nesse trabalho, porém, só foram tratadas regras e expressões para auxiliar na edição futura da planilha, semelhante ao proposto por Cunha *et al.* (2009a; 2012f). Regras e expressões para definições de fórmulas não foram abordadas.

2.1.2.6 **Classificação dos Trabalhos Relacionados**

Modelagem de planilhas	Ronen et al. (1989); Rajalingham et al. (2000); Powell e Baker (2003).
Teste de planilhas	Rothermel et al. (1998); Rothermel et al. (2001); Rothermel et al. (2000); Burnett et al. (2002); Clermont et al. (2008); Panko e Halverson (2001); Panko e Aurigemma (2010); Cunha et al. (2012e,i).
Entendimento da estrutura de planilhas	Davis (1996); Clermont (2004); Hermans et al. (2011).
Edição segura de planilhas	Cunha et al. (2009a, 2012f); Cunha et al. (2012b); Cunha et al. (2012c).
Transformação de modelos em planilhas e vice-versa	Cunha et al. (2009b); Hermans et al. (2010); Engels e Erwig (2005); (Cunha et al., 2011a; Mendes, 2012a; Cunha et al., 2011d; Cunha et al., 2012b; Cunha et al., 2012c; Cunha et al.;, 2012d; Cunha et al., 2013b; Belo et al., 2013; Cunha et al., 2010;Cunha et al., 2012a; Cunha et al., 2012h).

2.2 Modelos de Sistemas e os Padrões OMG

Modelos de sistemas possibilitam a visualização, a comunicação e a validação de aspectos de um sistema antes da sua construção. Um modelo de sistema é representado em uma linguagem de modelagem, que pode empregar uma combinação de representações gráficas e textuais (Rumbaugh *et al.*, 2004).

A elaboração de modelos envolve a aplicação de três princípios fundamentais ao desenvolvimento de sistemas complexos: decomposição, abstração e hierarquia (Booch, 1994). Através da decomposição, um sistema complexo pode ser dividido em elementos de menor complexidade (Parnas, 1972). A abstração permite que os desenvolvedores se concentrem nos aspectos relevantes de um problema, em um determinado nível de generalização (Ross *et al.*, 1975). As abstrações podem ser organizadas em hierarquias (Pfleeger, 2001), possibilitando a representação explícita de propriedades comuns e distintas entre diferentes elementos (Booch, 1994).

No final da década de 90, após ser adotada como padrão pelo OMG, a UML passou a ser amplamente utilizada por diversos métodos, técnicas e ferramentas de apoio à modelagem de sistemas, assim como no desenvolvimento de software em diversas áreas, tais como comércio eletrônico, jogos, automação comercial e

bancária, telecomunicações, robótica, aviação, dentre outras (Booch, 1999). A UML é uma linguagem que pode ser utilizada na especificação, visualização, construção e documentação de artefatos de sistemas de software, de negócio e de outros sistemas (OMG, 1999).

A UML define um conjunto de notações gráficas que podem ser utilizadas para descrever diversos aspectos de um sistema. Essas notações permitem a elaboração de diferentes tipos de diagramas como diagramas de classes, de objetos, de casos de uso, de atividades, de sequência, de colaboração, de estados, de implementação e de implantação.

A sintaxe abstrata da UML é definida segundo uma abordagem baseada em metamodelos. A Figura 4 apresenta a estrutura geral da arquitetura empregada nessa abordagem. O nível M3 corresponde ao MOF (*Meta Object Facility*). O MOF é um padrão, também adotado pelo OMG (OMG, 2002a), que define uma linguagem abstrata e uma estrutura para a especificação, construção e gerência de metamodelos de forma independente de tecnologia. As construções definidas no MOF seguem o paradigma de orientação a objetos, compartilhando um conjunto de elementos com a UML como, por exemplo, classes, atributos, operações e associações.

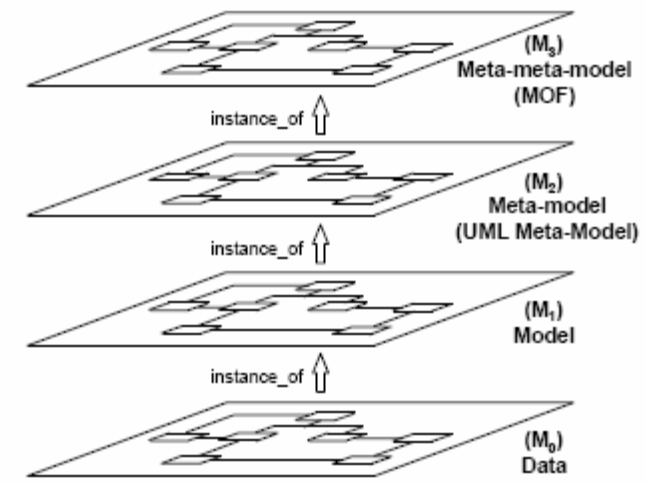


Figura 4–Estrutura de Metamodelos da OMG (OMG, 2002a)

O nível M2 corresponde aos metamodelos definidos a partir do MOF, ou seja, os elementos de um metamodelo M2 são instâncias de elementos definidos no MOF (M3). O metamodelo da UML é um exemplo de metamodelo M2.

O nível M1 corresponde aos modelos que são gerados a partir dos metamodelos M2. A principal responsabilidade dos modelos M1 é permitir a elaboração de representações de diferentes domínios como, por exemplo, sistemas

de software ou processos de negócio. O modelo de um software de gestão de uma locadora de vídeos é um exemplo de um modelo M1.

Um modelo M1 define o que deve acontecer quando seus elementos são instanciados no nível M0. Em linhas gerais, o nível M0 corresponde ao espaço de objetos e de relacionamentos entre objetos resultantes da instanciação das classes e associações definidas em um modelo M1.

A Figura 5 apresenta um exemplo onde esses diferentes níveis são empregados. No nível M3, é definido um elemento denominado *Class*. Esse elemento é instanciado na definição do metamodelo da UML (nível M2), dando origem a dois elementos básicos: *Attribute* e *Class*. No nível M1, que corresponde ao modelo de uma locadora de vídeos, a classe Vídeo é uma instância do elemento *Class*, definido em M2, enquanto que o seu atributo *title* corresponde a uma instância do elemento *Attribute*. Os elementos do nível M0 correspondem a objetos da classe *Video* criados durante a execução do sistema, como, por exemplo, o objeto *aVideo*.

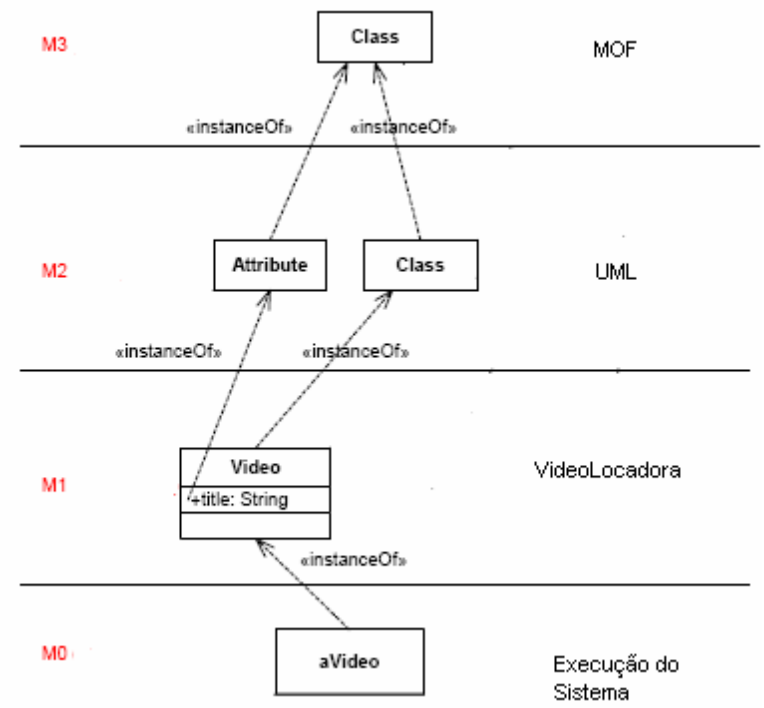


Figura 5 – Exemplo de modelos e metamodelos

2.3 Object Constraint Language (OCL)

Embora a UML ofereça um grande número de diagramas que possibilitam a descrição de diversos aspectos de um sistema, eles não são suficientes para descrever todos os detalhes que compõem um modelo de software ou mesmo um metamodelo. Restrições, regras e definições contratuais de operações são alguns exemplos de

informações que não são cobertas por esses diagramas e que demandam uma especificação mais precisa (Warmer e Kleppe, 2003).

Frequentemente, essas definições são descritas em linguagem natural. Entretanto, especificações produzidas em linguagem natural estão recorrentemente ligadas a problemas de ambiguidade (Berry e Kamsties, 2004). O emprego de uma linguagem formal na produção dessas definições é uma das alternativas para lidar com a ambiguidade, abrindo a possibilidade de apoio automatizado ao longo do processo de desenvolvimento (Pfleeger, 2001).

A OCL é uma linguagem de especificação textual e declarativa, também adotada como padrão pelo OMG (OMG, 2003a), que permite a definição precisa de restrições em modelos produzidos com a UML, bem como em metamodelos definidos a partir do MOF. As restrições são especificadas em OCL por meio de expressões que não modificam o espaço de objetos onde elas são avaliadas. Na arquitetura de quatro níveis descrita anteriormente, esse espaço de objetos pode corresponder ao estado de um sistema (nível M0), a elementos de um modelo (nível M1) ou a elementos de um metamodelo (nível M2). Embora não possam produzir mudanças em um espaço de objetos, expressões OCL podem ser utilizadas para especificar as mudanças que devem ser produzidas por operações, através de pós-condições, por exemplo.

2.3.1 **Tipos de Restrições e Expressões em OCL**

As restrições e expressões especificadas em OCL devem ser associadas a elementos (classes, atributos, operações, associações) de um modelo. Dentre os diferentes tipos de definições que podem ser especificadas em OCL, destacam-se:

- ***Invariante (inv):*** corresponde a uma expressão que é associada a um classificador do modelo, indicando que o resultado da sua avaliação deve ser verdadeiro para todas as instâncias que sejam de um tipo compatível com esse classificador;
- ***Derivação de atributos e associações (derive):*** corresponde a uma expressão que define a regra de derivação do valor de um atributo ou de uma associação a partir de outros elementos do modelo;
- ***Corpo de operação de consulta (body):*** corresponde a uma expressão que especifica o resultado de uma operação de consulta definida em um classificador do modelo;
- ***Valor inicial de atributos (init):*** corresponde a uma expressão que define o valor inicial de um atributo no momento em que for criado um objeto do classificador onde esse atributo tenha sido definido;

- ***Pré e pós-condições (pre e post):*** correspondem a expressões utilizadas para definir, de forma declarativa, a semântica das operações de um modelo. Na OCL, a definição de precondições e pós-condições para as operações das classes de um modelo está associada ao princípio de Projeto por Contrato (MEYER, 1992), onde uma operação é responsável por produzir certos resultados (obrigações ou pós-condições), apenas se certas condições (direitos ou pré-condições) forem atendidas.

2.3.2 **Tipos**

Na OCL, toda expressão tem um tipo que determina o domínio do seu resultado e das operações que podem ser aplicadas. Quatro tipos primitivos são predefinidos pela linguagem: *Boolean*, *Integer*, *Real* e *String*. Além dos tipos primitivos, os classificadores e as enumerações definidos no modelo também fazem parte dos tipos disponíveis para as expressões OCL.

Expressões OCL podem resultar em um valor primitivo, um objeto, uma *tupla* ou uma coleção desses elementos. A OCL define quatro tipos de coleção (*Set*, *Bag*, *Sequence* e *OrderedSet*), que correspondem à combinação de duas propriedades: a possibilidade de ocorrência de repetição de elementos na coleção e a existência de ordem entre estes elementos[1].

A OCL define um conjunto de operações que permite a manipulação de valores dos tipos primitivos e também das coleções, como operações lógicas (*and*, *or*, *xor*, *implies*, *if-then-else-endif*), aritiméticas (+, -, *, /), manipulação de *strings* (*size*, *concat*, *substring*) e coleções (*size*, *includes*, *isEmpty*). A relação completa dos tipos e de suas respectivas operações é descrita na especificação da linguagem (OMG, 2003a). As operações de consulta definidas nos classificadores do modelo também podem ser utilizadas nas expressões.

2.3.3 **Navegação**

A OCL é uma linguagem que possibilita expressar navegações no modelo, ou seja, a partir de um elemento inicial, é possível navegar pelas associações definidas no modelo. O elemento inicial de uma navegação pode corresponder a um objeto ou uma coleção de objetos. Cada navegação resulta em uma coleção contendo os objetos associados ao elemento inicial. O tipo da coleção resultante é definido de acordo com a multiplicidade e com o classificador destino da navegação. A navegação por uma

[1] Bag e Sequence possuem repetição de elementos; Set e OrderedSet não possuem. Sequence e OrderedSet definem ordem entre os elementos; Set e Bag não definem.

associação é definida por uma expressão com a estrutura <*origem*>.<*papel*>, onde *origem* corresponde a um objeto ou a uma coleção de objetos de uma classe *A*, e *papel* corresponde ao nome do papel (*rolename*) de uma classe associada à classe *A*.

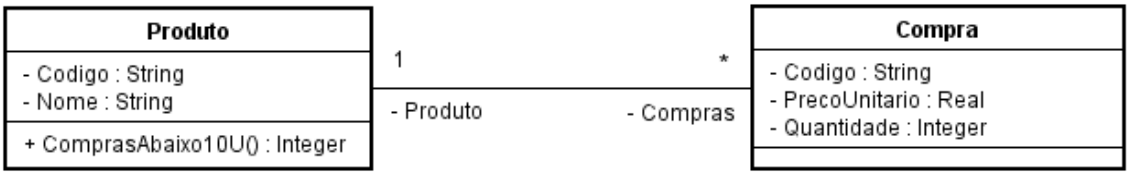


Figura 6 – Exemplo para explicar Navegação em OCL

```
01  context Produto::ComprasAbaixo10U() : Integer
02  body: self.Compras->select(compra | compra.Quantidade< 10)->size()
```

Figura 7 – Exemplo de Expressão de Navegação em OCL

A Figura 6 apresenta um pequeno exemplo que representa as compras de um determinado produto. Um produto pode ser comprado várias vezes. Para cada compra, são armazenadas as seguintes informações: código da compra, preço unitário pago pelo produto e a quantidade comprada.

A Figura 7 apresenta um exemplo de definição da semântica de uma operação de consulta em OCL. A operação *ComprasAbaixo10U* da classe *Produto* retorna o número de compras do produto com quantidade inferior a 10 unidades. A expressão *self.Compras* resulta na coleção de objetos da classe *Compra* ligados ao produto sobre o qual essa operação será aplicada, correspondendo à navegação pela associação entre as classes *Produto* e *Compra*. Em função da multiplicidade de *Compra* nessa associação, a coleção resultante é do tipo *Set(Compra)*. As chamadas às operações de manipulação de coleção são precedidas do operador *->.* Nesse mesmo exemplo, a operação *select* é aplicada sobre a coleção resultante da expressão *self.Compras,* resultando no subconjunto das compras que possuam um valor inferior a 10 para o atributo Quantidade. Sobre esse subconjunto resultante, é aplicada a operação *size*, que retorna o número de elementos desse subconjunto*.*

2.4 Considerações Finais

Este capítulo apresentou os principais conceitos e trabalhos relacionados que são relevantes para o entendimento dos demais capítulos desta Dissertação. Foram apresentados trabalhos relacionados com o objetivo de diminuir a incidência de erros

durante a construção e manutenção das planilhas, bem como algumas lacunas que existem na especificação dos conceitos e das fórmulas presentes nessas planilhas. Também foi apresentada uma breve introdução à estrutura utilizada pelo OMG para a definição de modelos e metamodelos, necessária para o entendimento do processo de transformação dos elementos do modelo nos elementos da planilha. Além disso, foram apresentadas as principais características da OCL, importantes para explicar as transformações de expressões OCL em fórmulas da planilha.

Capítulo III – Proposta de Solução

Construir uma planilha para apoiar a resolução de um problema não é uma atividade simples, sendo comum encontrarmos planilhas contendo erros. O capítulo anterior apresentou diversas soluções que procuram minimizar esse problema. Algumas delas propõem um processo de desenvolvimento de planilhas, buscando assim obter resultados padronizados e menos propensos a erros. Outras procuram facilitar a identificação dos erros por meio de abordagens de visualização da estrutura da planilha ou de métodos para execução sistemática de testes em planilhas.

Este capítulo apresenta a solução proposta para minimizar a ocorrência de erros em planilhas. Essa solução é baseada no conceito de *Model-Driven-Enginnering* (MDE), e consiste em construir automaticamente uma planilha a partir de um modelo conceitual do problema que ela visa resolver.

Este capítulo está dividido da seguinte forma: a seção 3.1 apresenta uma breve descrição do conceito de MDE; a seção 3.2 apresenta a visão geral da solução. A seção 3.3 descreve a representação empregada na construção do modelo conceitual. A seção 3.4 apresenta os elementos utilizados na elaboração de modelos de planilhas. A seção 3.5 explica as transformações de elementos de um modelo conceitual em elementos de um modelo de planilha. A seção 3.6 apresenta alguns aspectos da implementação da solução. Finalmente, a seção 3.7 tece considerações finais sobre o capítulo.

3.1 Visão Geral

O trabalho para industrializar o desenvolvimento de software tem sido motivado pelo crescimento contínuo da complexidade dos sistemas produzidos. Em particular, pesquisas na área de *Model-Driven Engineering* (MDE) estão preocupadas em reduzir a distância entre a representação do domínio do problema e as suas implementações em software por meio de tecnologias que possibilitem transformações sistemáticas de representações abstratas do problema em implementações concretas de software. Na visão MDE de desenvolvimento de software, modelos são os principais artefatos do

desenvolvimento, e os desenvolvedores utilizam tecnologias baseadas em software para transformar modelos em sistemas executáveis (France e Rumpe, 2007).

Em 2001, o OMG, no seu papel de organização que publica e mantém padrões para desenvolvimento de sistemas complexos, lançou um *framework* baseado em conceitos de MDE, chamado *Model-Driven Architecture* (MDA) (OMG, 2003c; Soley*et al.,* 2001), que acabou se tornando uma das implementações mais utilizadas na indústria. Os pilares do MDA são os seguintes padrões: *Meta Object Facility* (MOF) (OMG, 2002a), UML (OMG, 1999) e Query-View-Transformation, este último utilizado para especificar e implementar transformações de modelos (e.g., transformações PIM-PSM) (OMG, 2002b). Além do MDA, podemos citar outras iniciativas de aplicação dos conceitos do MDE, como as Fábricas de Software (Greenfield e Short, 2004), *Model Integrated Computing* (MIC) (Sztipanovits e Karsai, 1997), além de diversas outras abordagens proprietárias utilizadas na indústria.

A abordagem proposta adapta o conceito de geração de produtos de software a partir de transformações automáticas para o contexto de planilhas eletrônicas. A proposta consiste na geração de uma planilha a partir de um modelo conceitual do problema, mapeando os elementos presentes no modelo conceitual em elementos correspondentes aos componentes típicos de planilhas eletrônicas, como abas, tabelas e fórmulas. A Figura 8 apresenta a visão geral desta proposta.

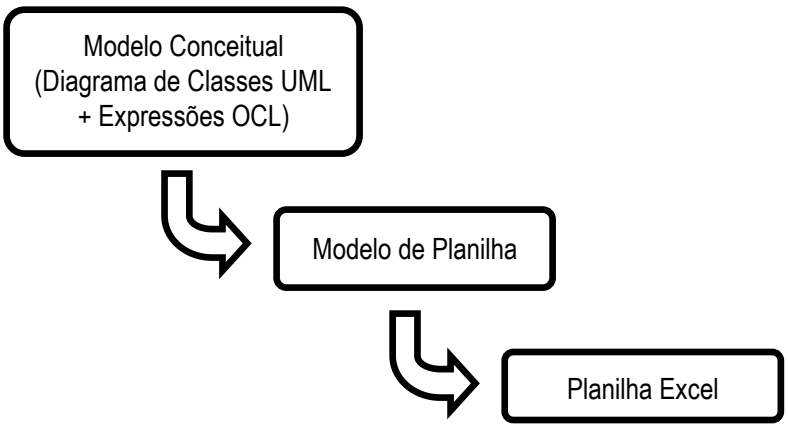


Figura 8 – Visão geral da proposta

A abordagem assume a existência de um modelo conceitual, correto e completo, contendo informações sobre os conceitos, propriedades, regras e relacionamentos que compõem o domínio do problema que se pretende resolver com a planilha. Esse modelo pode estar expresso de diversas formas, tais como: diagramas UML (tipicamente utilizados na construção de sistemas de software);

diagramas entidade-relacionamento (normalmente utilizados na modelagem conceitual de dados em projetos de bancos de dados); ontologias (mais frequentemente empregadas para especificar domínios na Web Semântica).

No contexto de desenvolvimento de software orientado a objetos, a UML (Booch et al., 2005) é o padrão *de facto* para a elaboração de modelos, sendo bastante difundida tanto na indústria como na academia. A forma mais comum de utilização da UML consiste na elaboração de modelos contendo representações gráficas semiformais, empregando os diversos diagramas definidos pela linguagem, complementadas com anotações em linguagem natural. Em cenários que demandam maior precisão do modelo, a OCL (Warmer e Kleppe, 2003) é uma linguagem que pode ser utilizada para especificar, de forma precisa, elementos que as notações gráficas da UML não são capazes de representar, como restrições, expressões associadas a atributos derivados e expressões de consulta.

Neste trabalho, diagramas de classe UML foram escolhidos para representar os conceitos, propriedades, relacionamentos e operações que descrevem o domínio do problema. A partir dessa escolha, foram definidas regras de transformação dos elementos presentes nos diagramas de classes (classes, atributos, associações e operações) em elementos de uma planilha (abas, tabelas e colunas), gerando assim um *modelo de planilha* correspondente ao conhecimento expresso no modelo conceitual.

Além disso, a OCL é utilizada na especificação da semântica das operações de consulta presentes nos diagramas de classes, com o intuito de transformá-las em fórmulas de cálculo da planilha. Para isso, foram definidas regras de transformação de expressões OCL em fórmulas de planilha. As expressões OCL estão vinculadas a elementos específicos do modelo de classes, tipicamente operações de classes. De forma análoga, as fórmulas geradas a partir das transformações são vinculadas ao elemento do modelo de planilha correspondente, normalmente colunas das tabelas.

Uma vez que o modelo da planilha tenha sido gerado, a abordagem prevê a transformação desse modelo em um arquivo de planilha que possa ser aberto em um software específico de planilha. Neste trabalho, consideramos a utilização do formato xlsx, utilizado no Microsoft Office Excel 2007.

3.2 Modelo Conceitual

Um modelo conceitual é um modelo que permite entender e simplificar um problema (Fowler, 1997). Esse modelo pode ser representado de forma explícita em uma

linguagem de alto nível, e tipicamente descreve conceitos, atributos, relacionamentos e restrições do universo do problema. Como os modelos conceituais não incluem detalhes de implementação, eles podem ser utilizados para apoiar a comunicação com usuários não técnicos (Elmasri e Navathe, 2009).

Este trabalho assume que o modelo conceitual é representado por meio de diagramas de classe UML e expressões OCL. Esta seção apresenta um exemplo de modelo conceitual de um problema utilizando essa representação. O modelo corresponde aos requisitos para o controle de estoque de uma rede de minimercados, chamada *Minimundo*. A rede *Minimundo* possui um conjunto de lojas divididas em departamentos. Cada departamento possui produtos próprios, comprados de fornecedores e revendidos para os clientes nas lojas. A Figura 9 apresenta o diagrama de classes do problema.

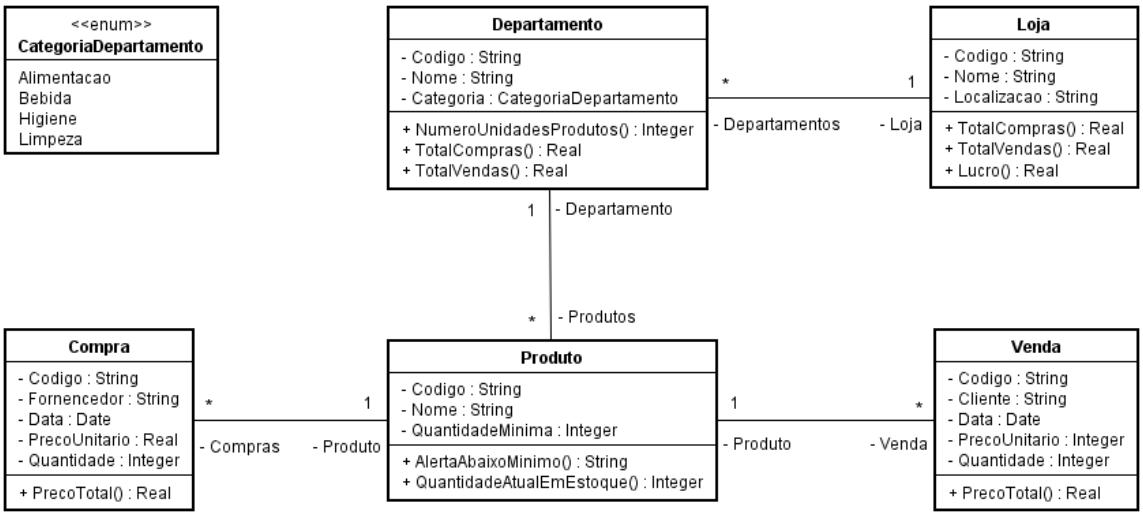


Figura 9 - Diagrama de Classes da Rede Minimundo

Em um diagrama de classes UML, os conceitos são representados por meio de classes. O conceito *Loja,* por exemplo, é representado pela classe *Loja* no diagrama. Os relacionamentos entre os conceitos são representados por associações que definem a multiplicidade de cada um dos seus participantes e também os papéis que cada classe assume na associação. A associação entre *Loja* e *Departamento*, por exemplo, declara que uma *Loja* pode ter zero ou mais departamentos (multiplicidade *), e que um departamento está associado a uma e somente uma loja (multiplicidade 1). Além disso, cada conceito pode ser descrito por um conjunto de informações representadas por meio de atributos da classe. No exemplo, o conceito *Loja* é descrito por informações como código, nome e localização, representadas pelos atributos da

classe *Loja*. Os conceitos também podem definir operações que retornam informações relevantes sobre os mesmos. Por exemplo, o conceito *Loja* define a operação *TotalCompras,* que retorna o valor total gasto em compras por aquela loja.

O modelo apresentado na Figura 9 foi complementado com um conjunto de expressões OCL associadas às operações de consulta das classes do exemplo. A Figura 10 apresenta a especificação dos resultados de cada operação do modelo por meio de expressões OCL.

```
01 context Loja::Lucro() : Real
02 body: TotalVendas() - TotalCompras()
03
04 context Loja::TotalCompras() : Real
05 body: Departamentos.TotalCompras()->sum()
06
07 context Loja::TotalVendas() : Real
08 body: Departamentos.TotalVendas()->sum()
09
10 context Compra::PrecoTotal() : Real
11 body: PrecoUnitario * Quantidade
12
13 context Venda::PrecoTotal() : Real
14 body: PrecoUnitario * Quantidade
```

Figura 10 – Exemplo de especificações de operações em OCL

As principais características da OCL foram apresentadas na seção 2.3. A palavra reservada *context* define o contexto da expressão, isto é, o elemento do modelo ao qual a expressão OCL está vinculada. Por exemplo, a última expressão (linha 13) especifica a semântica da operação *PrecoTotal* da classe *Venda* (*context Venda::PrecoTotal()*). Cada operação de consulta definida no modelo pode ter sua semântica especificada por uma expressão OCL definida após a palavra reservada *body*. Na operação *Venda::PrecoTotal*, por exemplo, a expressão OCL corresponde à multiplicação do valor do atributo *PrecoUnitario* (preço unitário da venda) pelo valor do atributo *Quantidade* (quantidade vendida do produto vendido), ambos definidos na classe *Venda*.

Uma vez que a abordagem proposta é baseada em transformações definidas a partir de metamodelos, esta seção descreve brevemente os principais elementos dos metamodelos de referência para o modelo conceitual, i.e., UML e OCL.

A Figura 11 apresenta os principais elementos do metamodelo da UML, relacionados com a definição de classes, atributos e operações, enquanto que a Figura 12 contém o subconjunto de elementos relacionados com a definição de associações em um modelo UML.

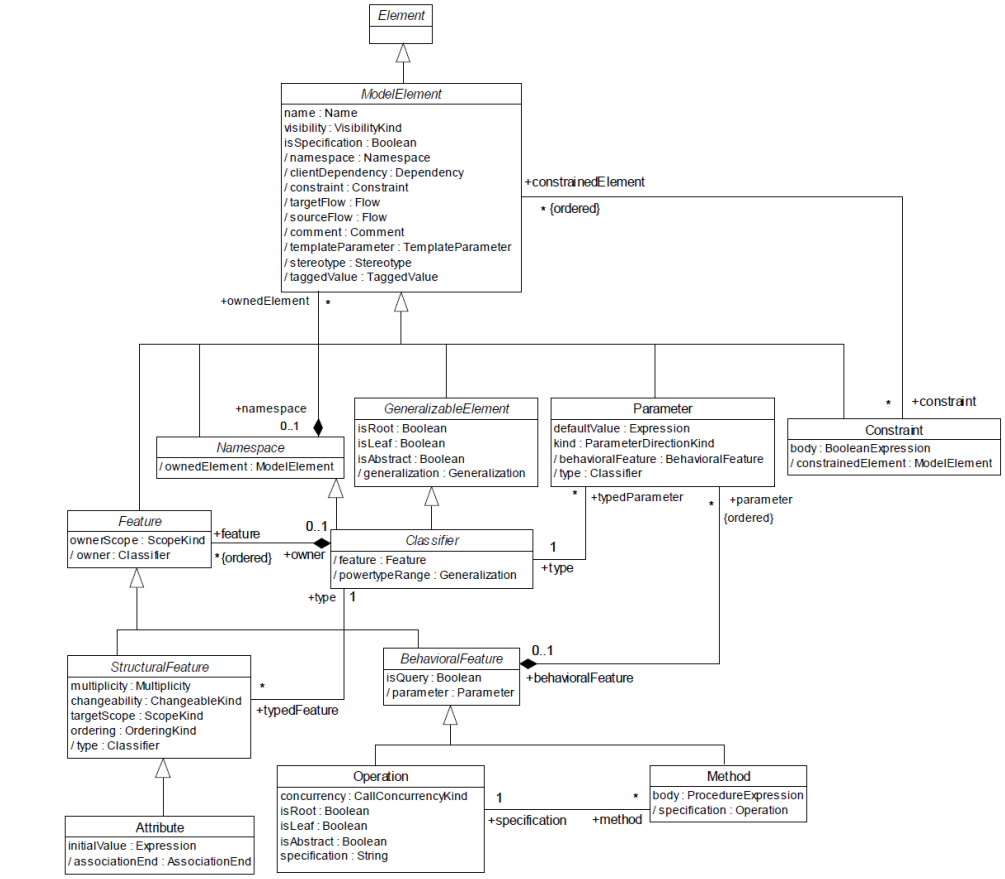


Figura 11 – Metamodelo da UML – Backbone

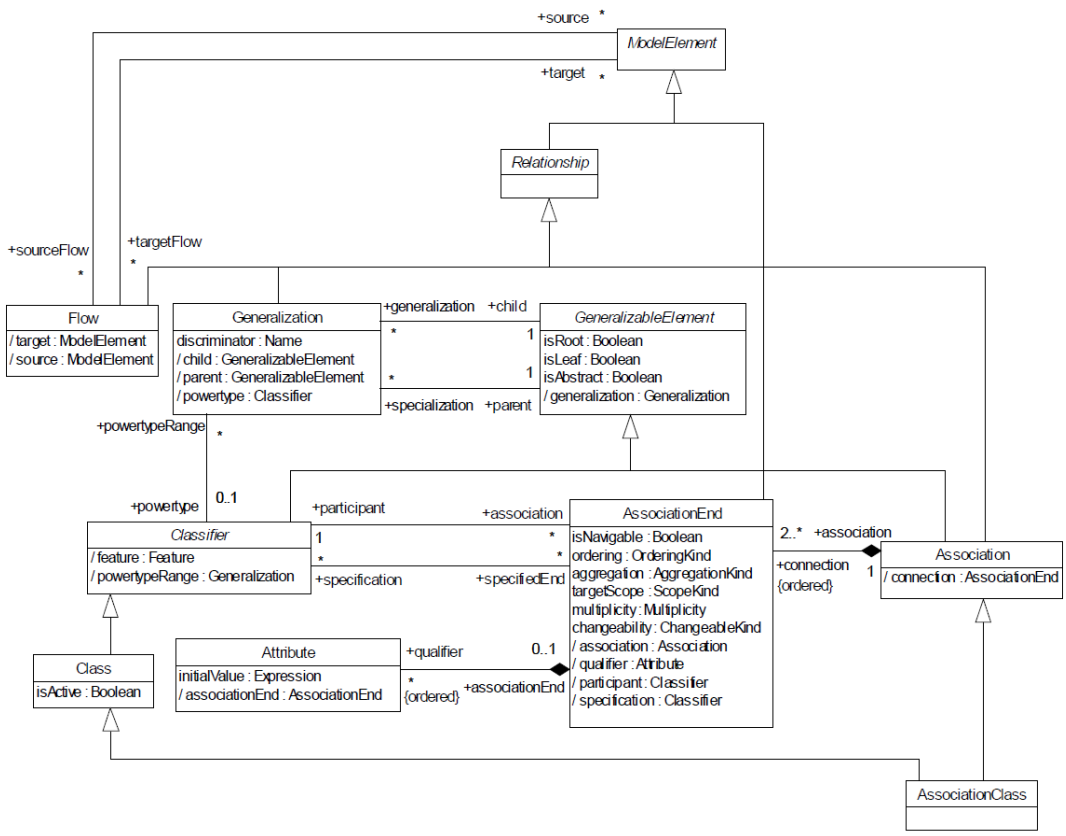


Figura 12 – Metamodelo da UML – Associações

Um diagrama de classes, como o apresentado na Figura 9, contém instâncias de elementos definidos no metamodelo da UML. Para exemplificar essa relação entre modelo e metamodelo, a Figura 13 apresenta o diagrama de objetos do metamodelo UML correspondente à classe *Departamento* do modelo exemplo.

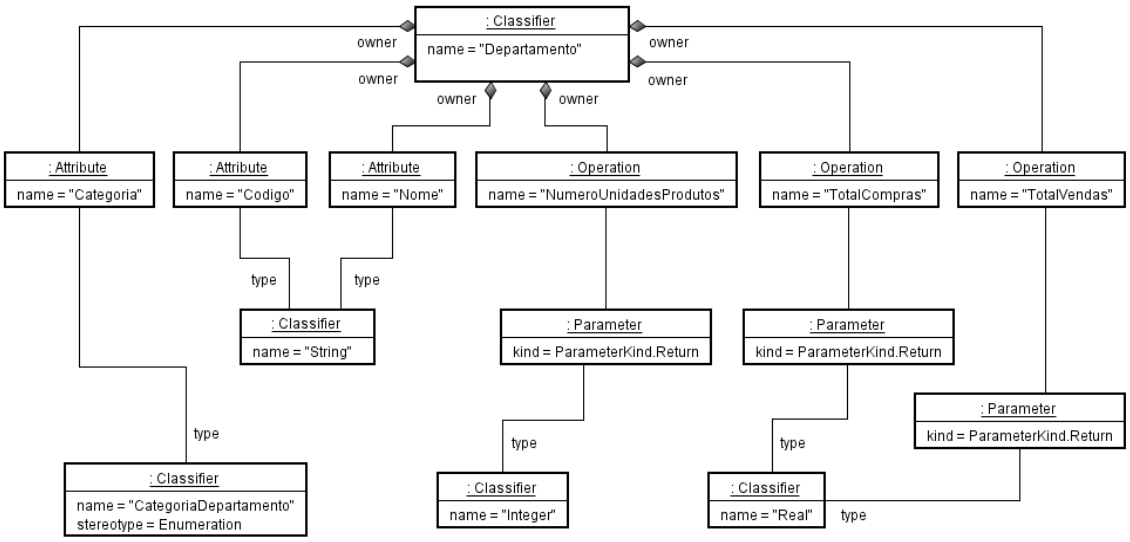


Figura 13 – Classe Departamento como instância do metamodelo UML

No metamodelo da UML, uma classe é representada por uma instância do elemento *Classifier*. No exemplo da Figura 13, a classe *Departamento* corresponde a uma instância de *Classifier*. Os atributos de *Departamento* correspondem a instâncias do elemento *Attribute*. As operações da classe *Departamento* correspondem a instâncias da classe *Operation*. Os elementos *Attribute* e *Operation* são especializações do elemento *Feature*, que possui uma associação com a classe *Classifier,* estabelecendo o *Classifier* onde aquela *Feature* foi definida. Ainda no exemplo, a relação entre a classe *Departamento* e seus atributos e operações é definida por meio de uma ligação entre cada instância de *Feature* (*Attribute* ou *Operation*) e a instância de *Classifier* que corresponde ao dono (*owner*) da *Feature*, papel que o *Classifier* assume na associação.

Cada atributo tem um tipo. Todos os tipos de dados também são definidos como instâncias de *Classifier*. Um elemento do tipo *Attribute* possui uma associação com um *Classifier* que indica o seu tipo. Por exemplo, o atributo *Nome* da classe *Departamento* possui uma ligação com a instância de *Classifier* que representa o tipo *String*.

Os parâmetros e o tipo de retorno de uma operação são representados por meio de ligações entre a instância de *Operation* com instâncias do elemento

Parameter, uma para cada parâmetro ou retorno. O elemento *Parameter* define o tipo do parâmetro (entrada ou retorno) e está associado com um *Classifier* que define o tipo do parâmetro. Por exemplo, a operação *TotalCompras* possui uma ligação com uma instância da classe *Parameter* do tipo retorno. Essa instância de *Parameter* está ligada a uma instância de *Classifier* correspondente ao tipo de dado *Real*, determinando que a operação retorne um valor Real como resultado.

É importante ressaltar que o escopo deste trabalho não engloba todos os aspectos definidos no metamodelo UML referentes à construção de diagramas de classes. Construções como generalização e especialização, associações binárias com multiplicidade * em ambos os lados, associações que não sejam binárias, atributos multivalorados, agregação/composição e interfaces não foram incluídas no escopo deste trabalho.

Expressões OCL podem ser vistas como uma representação em uma sintaxe concreta de instâncias do metamodelo da OCL. A Figura 14 apresenta os principais elementos desse metamodelo.

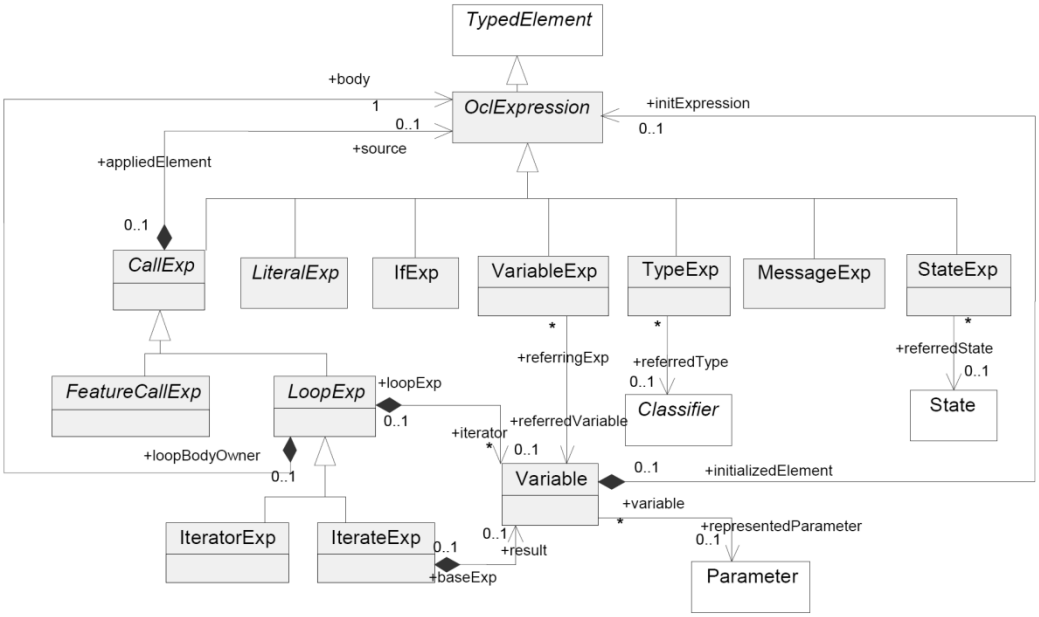


Figura 14 – Metamodelo da sintaxe abstrata da OCL

Para exemplificar a relação entre expressões e o metamodelo da OCL, a Figura 15 apresenta o diagrama de objetos correspondente à expressão *PrecoUnitario * Quantidade* associada à operação *PrecoTotal* da classe *Venda*. O elemento central desse diagrama é uma instância da classe *OperationCallExp,* que representa uma expressão correspondente à chamada a uma operação de um *Classifier*. Essa classe herda da classe *FeatureCallExp*, apresentada na Figura 14. Essa associação é

representada através da ligação dessa instância de *OperationCallExp* com uma instância de *Operation*. A operação '*' (multiplicação) é definida no *Classifier Real*, definido como um tipo padrão na especificação da OCL. Essa operação recebe como parâmetro outro elemento do tipo *Real,* e retorna um novo elemento do tipo *Real*, representados pelas ligações da instância de *Operation* com instâncias de *Parameter*.

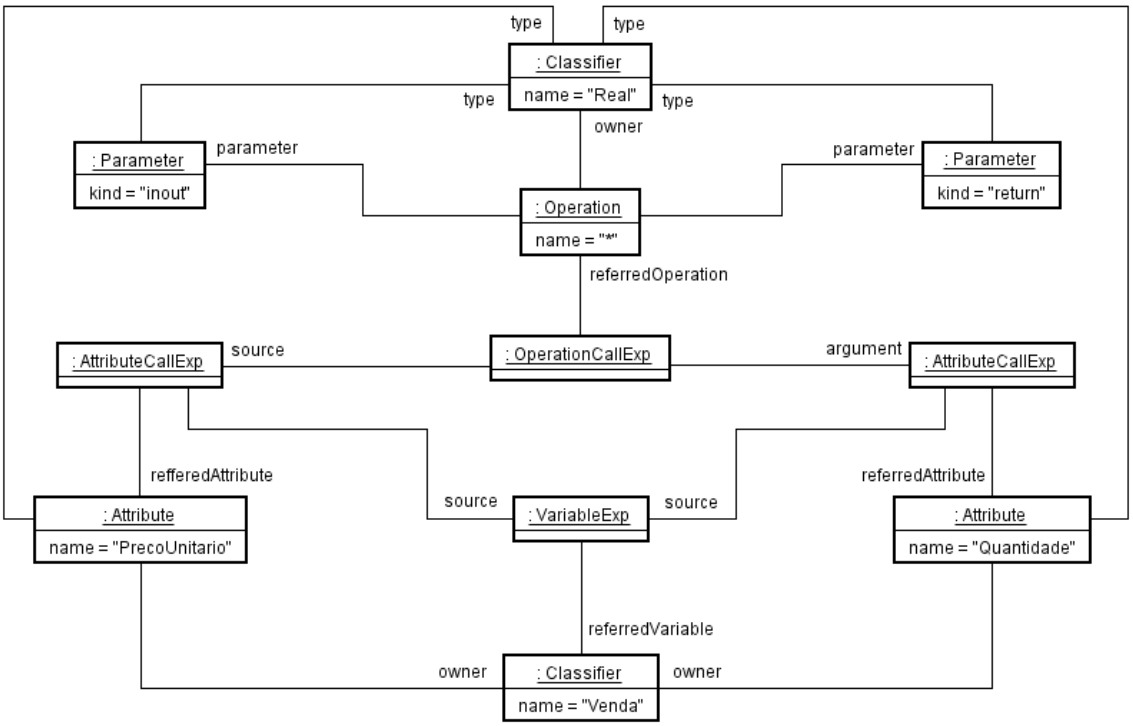


Figura 15 – Expressão *PrecoTotal* como instância do metamodelo da OCL

Uma instância de *OperationCallExp* está sempre ligada a uma fonte (source) que originou a chamada e aos argumentos (elementos com o papel *argument*) utilizados na chamada dessa operação. No exemplo da Figura 15, a fonte da *OperationCallExp* correspondente à expressão *PrecoUnitario * Quantidade* é uma instância da classe *AttributeCallExp*. Essa classe representa um acesso a um atributo de um *Classifier*. Essa associação é representada pela ligação de *AttributeCallExp* com uma instância de *Attribute*, que, no caso, é o atributo *PrecoUnitario* do *Classifier Venda*. Da mesma forma, o argumento dessa *OperationCallExp* é um *AttributeCallExp*, representando o acesso ao atributo *Quantidade* do *Classifier Venda.* As duas instâncias de *AttributeCallExp* estão ligadas a uma instância da classe *VariableExp* que representa uma variável (*self)* definida na expressão. Essa variável representa uma instância da classe que define o contexto da expressão, no caso um objeto da classe *Venda* ao qual será feita a chamada da operação *PrecoVenda*.

Uma expressão pode ser vinculada a um elemento do modelo de diversas formas, como apresentado na seção 2.3.1. No escopo deste trabalho só foram consideradas expressões de consulta vinculadas a operações do modelo.

A Figura 16 apresenta como esse vínculo é feito a partir dos metamodelos da UML e da OCL. A UML prevê, em seu metamodelo, que qualquer elemento definido no modelo (*ModelElement*) pode ter restrições (*Constraint*) associadas a ele. No caso de expressões vinculadas a operações de um *Classifier*, esse elemento é uma instância de *Operation*.

Uma restrição está associada a uma expressão (*Expression*) que pode ser representada de diversas formas, como em OCL, por exemplo. A classe *ExpressionInOcl* representa toda a definição da expressão, enquanto que o conteúdo da expressão está armazenado em uma expressão OCL (*OclExpression*). Quando a expressão é definida a partir de uma restrição OCL do tipo *body*, esse conteúdo assume o papel de bodyExpression na associação com a definição da expressão (*ExpressionInOcl*).

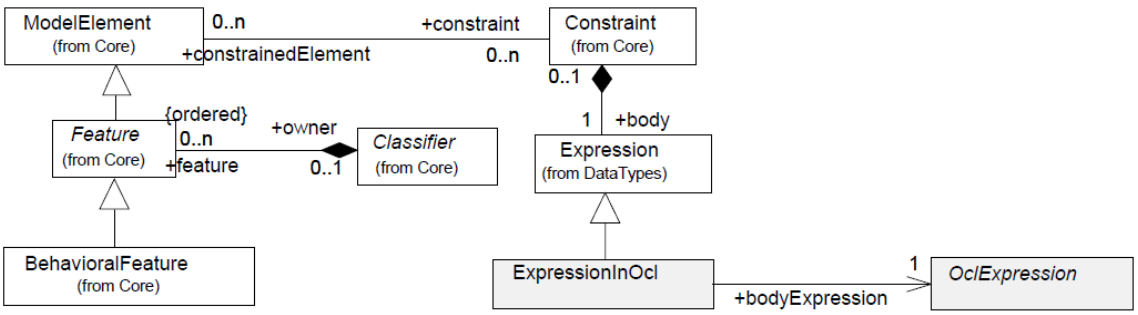


Figura 16 – Expressão OCL definida em uma restrição do tipo Body

3.3 Metamodelo de Planilhas

Assim como diagramas de classe UML e expressões OCL podem ser vistos como instâncias de seus respectivos metamodelos, uma planilha também pode ser definida como uma instância de um metamodelo de planilhas. Esse mapeamento permite a definição de regras de transformação entre elementos dos metamodelos da UML e da OCL em elementos do metamodelo de planilhas.

Um metamodelo de planilhas foi definido neste trabalho com base em observações sobre a forma como geralmente as planilhas são construídas no dia-dia em diversos domínios. A generalização desse metamodelo necessita de uma avaliação mais abrangente das estruturas das planilhas construídas, mas essa avaliação está fora do escopo deste trabalho.

Outros metamodelos de planilhas foram propostos, como em (Engels e Erwig, 2005). A sintaxe deste metamodelo é apresentada na Figura 17 e a Figura 18 apresenta instâncias dos elementos dessa sintaxe.

$$
\begin{array}{lcll}
f \in Fml & ::= & \varphi \mid n.a \mid \varphi(f,\ldots,f) & (formulas) \\
b \in Block & ::= & \varphi \mid a = f \mid b \mid b \mid b \hat{} b & (blocks) \\
\ell \in Lab & ::= & h \mid v \mid \_n & (class\ labels) \\
h \in Hor & ::= & \underline{n} \mid \underline{|n} & (horizontal) \\
v \in Ver & ::= & \overline{|n} \mid \underline{|n} & (vertical) \\
c \in Class & ::= & \ell : b \mid \ell : b^{\downarrow} \mid c \hat{} c & (classes) \\
s \in Sheet & ::= & c \mid c^{\rightarrow} \mid s \mid s & (sheets)
\end{array}
$$

Figura 17 – Sintaxe de um ClassSheet

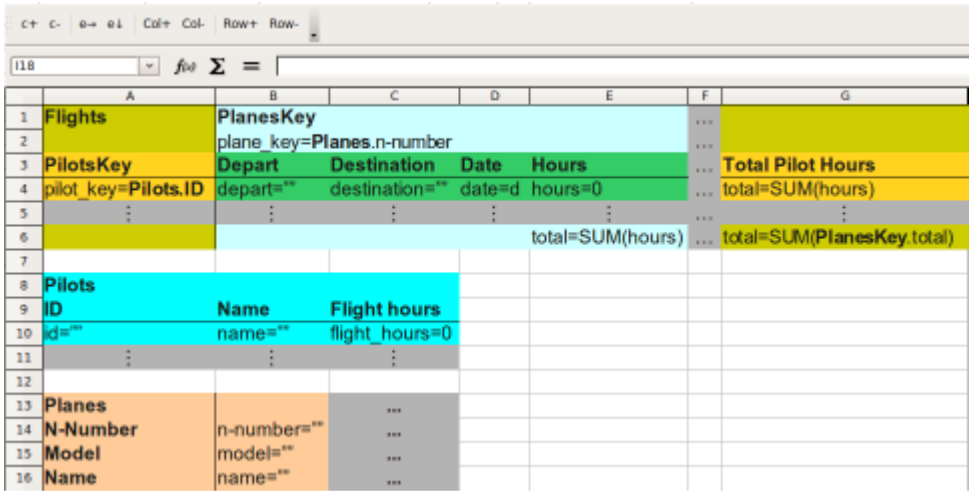


Figura 18 – Modelo de ClassSheet

Esse metamodelo difere do proposto neste trabalho em dois aspectos fundamentais: ele não trata tabelas, mas apenas blocos de dados que se expandem “verticalmente” e “horizontalmente”. Além disso, ele prevê a criação de uma planilha bidimensional para representar relacionamentos entre classes que além de ser uma construção menos intuitiva chega a ser um pouco confusa em algumas situações. Essas foram as principais limitações que motivaram a proposição de um metamodelo que melhor represente o que entendemos como uma planilha bem estruturada. Os componentes desse metamodelo são apresentados na Figura 19.

Esse metamodelo foi elaborado de forma iterativa e incremental de acordo com os modelos elaborados e o que era necessário para representá-los. Sendo assim, de acordo com novos requisitos esse metamodelo pode ser estendido e modificado.

Além disso, esse metamodelo não contempla a abstração de fórmulas, sendo elas representadas simplesmente por textos.

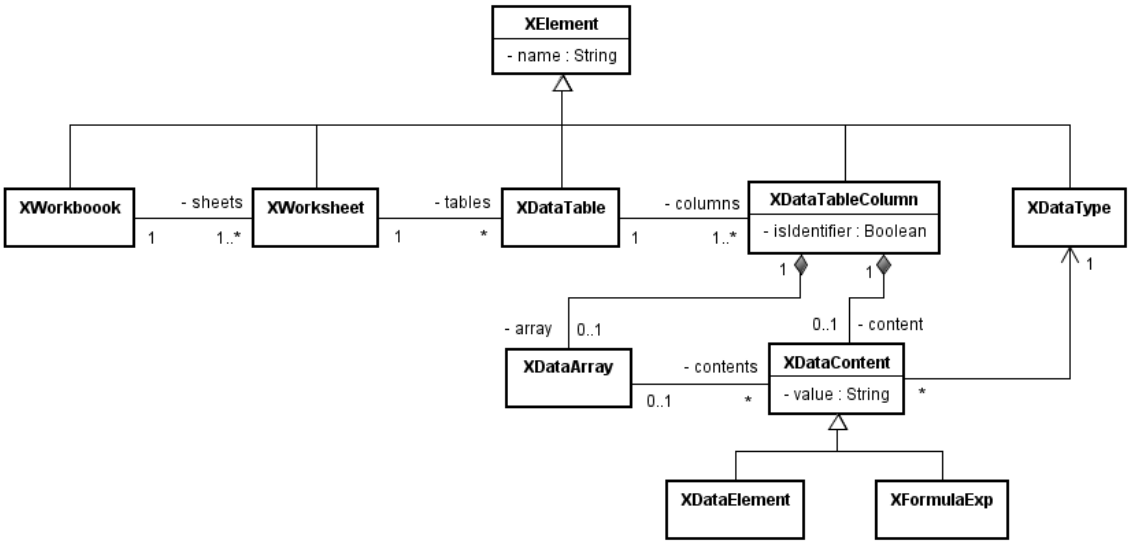


Figura 19 – Metamodelo de planilhas

Os principais elementos desse metamodelo são *XWorkbook*, *XWorksheet* e *XDataTable*. Esses elementos são especializações da abstração *XElement,* que representa um elemento genérico de uma planilha. *XWorkbook* é a classe que representa toda a planilha, ou seja, a composição final de todos os elementos da planilha. Um *XWorkbook* pode conter um ou mais *XWorksheets*, que correspondem às abas da planilha. Um *XWorksheet* pode estar ligado a zero ou mais instâncias da classe *XDataTable*.

Um *XDataTable* é uma tabela da planilha. Consideramos uma tabela de uma planilha como uma estrutura em forma de matriz, com linhas e colunas, onde cada coluna possui um cabeçalho que descreve o conteúdo dos seus elementos. Uma tabela, por sua vez, pode conter um ou mais *XDataTableColumns*, que são as colunas da tabela. Um XDataTableColumn pode ser uma coluna identificadora ou não, informação representada pelo atributo *isIdentifier*. Sendo assim, uma tabela pode possuir uma ou mais colunas identificadoras, responsáveis por identificar, de forma única, cada linha da tabela.

Elementos do tipo *XDataTableColumn* podem conter um *XDataContent*, que representa o conteúdo de dados daquela coluna, ou um *XDataArray*, que contém um conjunto de elementos do tipo *XDataContent*. Um XDataContent pode ser uma entrada de dados simples (*XDataElement*) ou uma fórmula (*XFormulaExp*), sempre

atendendo a um tipo de dado (*XDataType*). O conteúdo do *XDataContent* é armazenado no atributo *value*, que é sempre um texto.

Para ilustrar a aplicação desse metamodelo, apresentamos uma planilha no Microsoft Excel (Figura 20) e o diagrama de objetos da aba de *Produtos* dessa planilha (Figura 21), representando uma instanciação do metamodelo de planilhas.

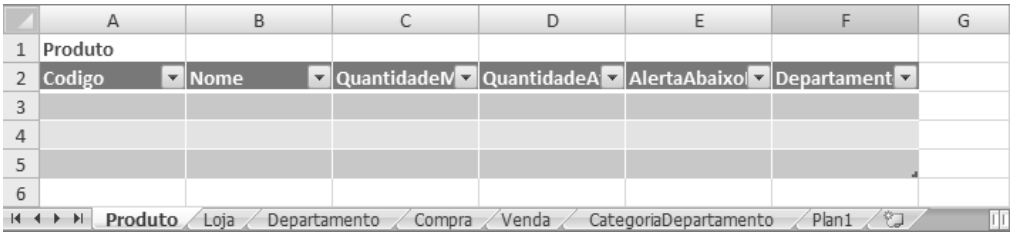


Figura 20 – Exemplo de Planilha

Essa planilha, representada por uma instância de *XWorkbook,* contém 7 abas (*XWorksheets*). Na aba *Produto*, por exemplo, é definida uma tabela (*XDataTable*) com 6 colunas (*XDataTableColumn*). O *XDataTableColumn* com o nome *Codigo* é a coluna identificadora da tabela, representado pelo atributo *isIdentifier* como *True.* O conteúdo da coluna *Nome* é um dado simples de entrada (XDataElement), e por isso tem seu atributo *value* com valor vazio (“”). Como o conteúdo da coluna *QuantidadeAtualEmEstoque* é uma fórmula que retorna um dado do tipo *Número Inteiro*, a instância de *XDataTableColumn* correspondente a essa coluna está ligada a uma instância de *XFormulaExp,* cujo atributo *value* contém a fórmula.

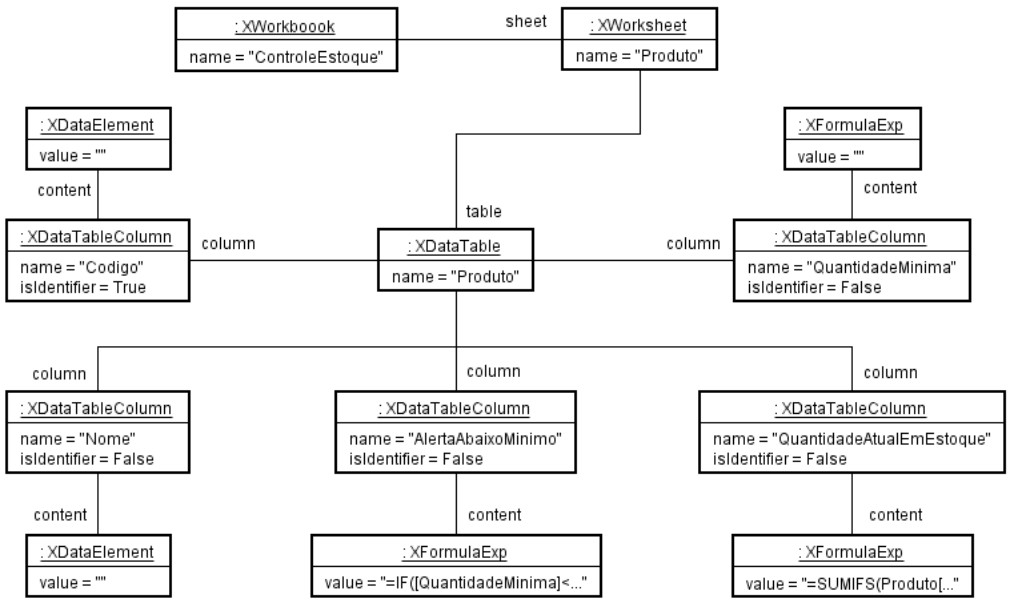


Figura 21 – Aba Produto como instância do Metamodelo da Planilha

3.4 Transformação do Modelo Conceitual em Modelo de Planilha

As transformações das classes, atributos, operações e associações de um modelo conceitual foram elaboradas a partir de adaptações da proposta apresentada por Heuser (2009) para transformação de um MER (Modelo Entidade-Relacionamento) em um Modelo Relacional. Na proposta de solução do presente trabalho, as classes do modelo assumem o papel das entidades, enquanto que as tabelas da planilha assumem o lugar das tabelas do modelo relacional.

Um modelo conceitual pode ser implementado por diferentes modelos de planilha, todos possíveis implementações corretas do modelo conceitual. Entretanto, cada diferente modelo de planilha pode implicar em maior facilidade ou dificuldade de manutenção da planilha construída. As regras propostas neste trabalho têm o intuito de gerar um modelo de planilha inicial que reduza as chances de introdução de erros pelos usuários finais, mas que possa sofrer adaptações de acordo com os requisitos de negócio de cada organização. Para exemplificar as transformações propostas, o mesmo modelo conceitual apresentado na Figura 9 será utilizado.

3.4.1 **Tradução do Modelo**

Esta transformação consiste em construir o elemento central do modelo de planilhas, representando a planilha como um todo. Logo, a primeira transformação realizada é a transformação do elemento central do modelo origem, que, no metamodelo da UML, corresponde ao elemento *Model*. Todas as classes, enumerações e associações do modelo pertencem a esse elemento central. A transformação do elemento *Model* é definida da seguinte forma:

> Criar uma instância de *XWorkbook* (planilha) com o atributo *name* possuindo o mesmo valor do atributo *name* do elemento *Model* do modelo conceitual.

3.4.2 **Tradução de Classes**

Os principais elementos de um diagrama de classes são as classes. No metamodelo da UML, uma classe é representada como instância de um *Classifier*. As classes são diferenciadas das enumerações somente por uma característica: toda enumeração possui uma ligação com um *Stereotype* do tipo *Enumeration*. Portanto, todo *Classifier* que não for uma enumeração é uma classe. Sendo assim, foi definida a seguinte transformação:

Para cada *Classifier* que não possuir uma ligação com um *Stereotype* s tal que s.name = '*Enumeration':*

- criar um novo *XWorksheet* (aba da planilha) com o mesmo nome do *Classifier* e ligá-lo ao *XWorkbook* criado no passo anterior*,*
- *criar* um novo *XDataTable* (tabela) com o mesmo nome do *Classifier* e ligá-lo ao *XWorksheet* criado.

Para exemplificar essa tradução, considere o diagrama de objetos apresentado na Figura 22. A planilha resultante possui cinco abas, uma para cada classe do modelo conceitual. Em cada aba, uma tabela correspondente à respectiva classe foi criada. Os nomes das abas e os nomes das tabelas são iguais aos nomes das classes do modelo.

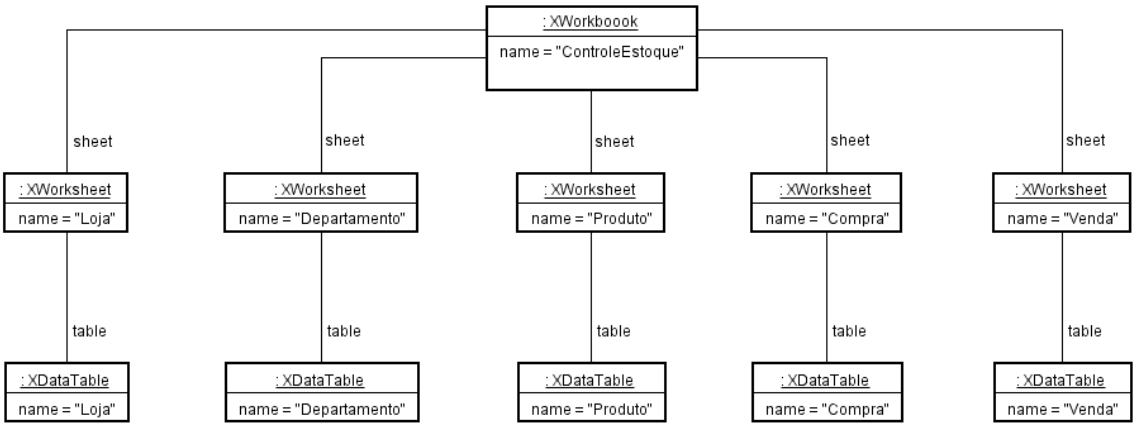


Figura 22 – Planilha resultante do passo de tradução de classes

3.4.3 **Tradução de Atributos**

Em uma classe, suas características são representadas por atributos. No metamodelo da UML, um atributo é representado por uma instância da classe *Attribute*. Para armazenar a ligação entre uma classe e seus atributos, um *Classifier* mantém uma lista de elementos que pertencem a ele, chamada de *ownedElements*. Essa lista contém as *Features* de um *Classifier*, que podem ser tanto atributos (*Attributtes*) quanto operações (*Operations*). Sendo assim, foi definida a seguinte transformação:

> Para cada *Classifier* c que não possuir uma ligação com um *Stereotype* s tal que s.name = '*Enumeration':*
>
> Para cada elemento em *c.ownedElements* que seja um *Attribute:*
>
> Criar um novo *XDataTableColumn* (coluna) com o nome do atributo, ligando-o ao *XDataTable* correspondente ao *Classifier c*.
>
> Se o atributo estiver ligado a uma instância de *Stereotype com nome = "Id"*
>
> *Então* coluna.*IsIdentifier* := verdadeiro.
>
> Senão coluna.*IsIdentifier* := falso.
>
> Fim Se

Os atributos identificadores da entidade se tornarão colunas correspondentes à identificação única de cada instância (linha) da tabela, que serão chamadas de *colunas-chave*. O restante dos atributos se tornarão colunas não identificadoras da tabela. Os nomes das colunas serão iguais aos nomes dos atributos.

Para exemplificar essa tradução, um diagrama de objetos com os elementos criados para representar os atributos da classe *Produto* é mostrado na Figura 23. Esse diagrama contém três instâncias de *XDataTableColumn* ligados à instância *XDataTable* Produto, cujos valores para o atributo nome (*Codigo*, *Nome* e *QuantidadeMinima)* correspondem aos nomes dos atributos da classe Produto. O *XDataTableColumn Codigo* é a coluna identificadora (*isIdentifier = True*) e as demais colunas não são identificadoras (*isIdentifier = False*).

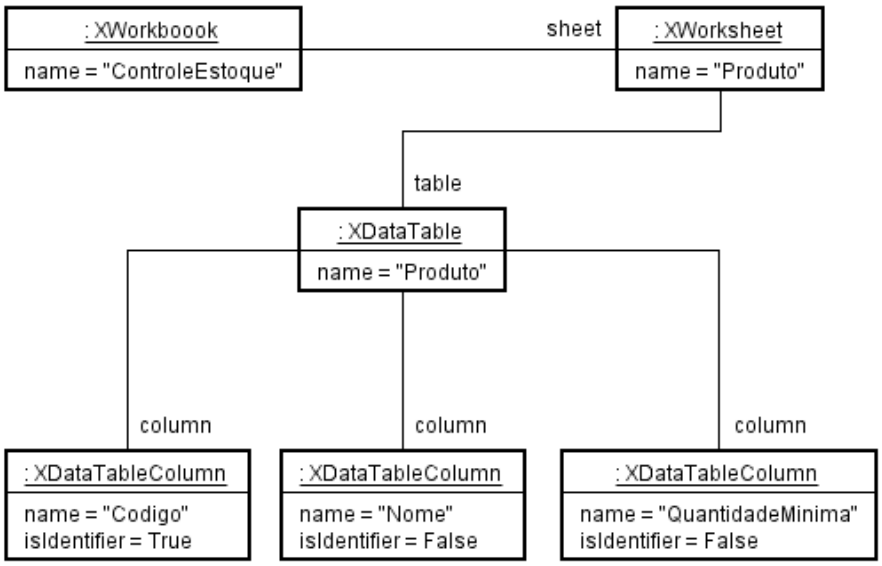


Figura 23 – Diagrama de objetos – Atributos de Produto

3.4.4 **Tradução de Enumerações**

Em um diagrama de classes, enumerações são responsáveis por representar um conjunto completo de valores possíveis para uma determinada característica. Da mesma forma que a Classe, uma Enumeração é representada por uma instância da classe *Classifier* que possui uma ligação com uma instância de um estereótipo *Enumeration.* A regra para transformação de enumerações é apresentada a seguir.

> Para cada *Classifier* que possuir uma ligação com um *Stereotype* s tal que s.name = '*Enumeration':*
>
> - criar um novo *XWorksheet* (aba da planilha) com o mesmo nome do *Classifier e ligá-lo ao XWorkbook* criado na primeira transformação*;*
> - *criar* um novo *XDataTable* (tabela) com o mesmo nome do *Classifier* e ligá-lo ao *XWorksheet* criado.
> - criar um *XDataTableColumn* (coluna) ligado ao *XDataTable* criado com o atributo *isIdentifier* = True.
> - criar um *XDataArray ligado ao XDataTableColumn criado no passo anterior.*
> - Para cada *Attribute* pertencente ao *Classifier,* criar um novo *XDataContent* com valor igual ao nome do atributo e ligá-lo ao XDataArray criado no passo anterior.

Para exemplificar esta transformação, a Figura 23 representa a enumeração *CategoriaDepartamento*, apresentada no modelo da Figura 12. Neste exemplo, essa enumeração foi transformada em uma tabela chamada *CategoriaDepartamento* com uma coluna, também chamada de *CategoriaDepartamento*. Os elementos dessa tabela correspondem aos valores possíveis na enumeração: *Bebida*, *Higiene*, *Limpeza* e *Alimentacao*.

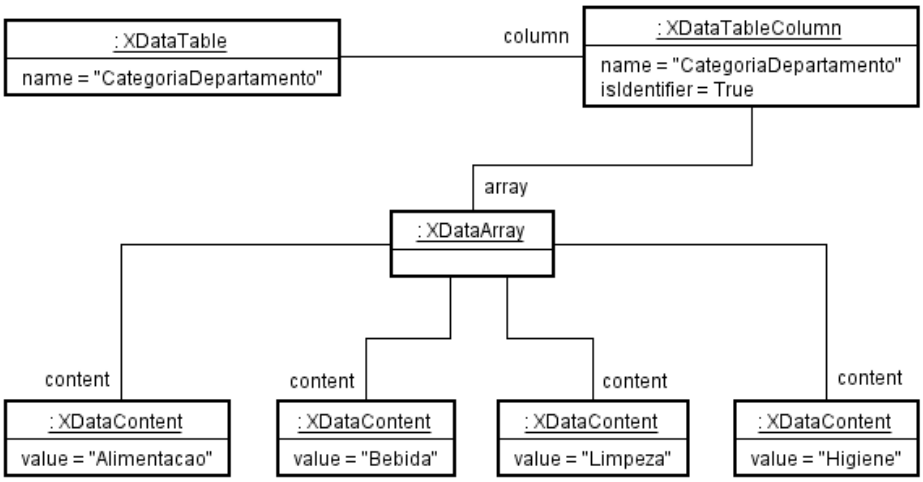


Figura 24 – Exemplo de transformação de enumerações

3.4.5 **Tradução de Operações**

Uma operação em um diagrama de classes fornece uma interface que permite obter alguma informação sobre os objetos da classe onde ela está definida. No metamodelo da UML, uma operação é representada por uma instância da classe *Operation*. Da mesma forma que um atributo, ela também está presente na lista de *ownedElements* de um *Classifier*. Então, a seguinte regra de transformação foi criada:

> Para cada *Classifier* c que não possuir uma ligação com um *Stereotype* s tal que s.name = '*Enumeration':*
>
> Para cada elemento em *c.ownedElements* que seja do tipo *Operation:*
>
> criar um novo *XDataTableColumn* (coluna) com o mesmo nome da operação no *XDataTable* correspondente ao *Classifier c*.

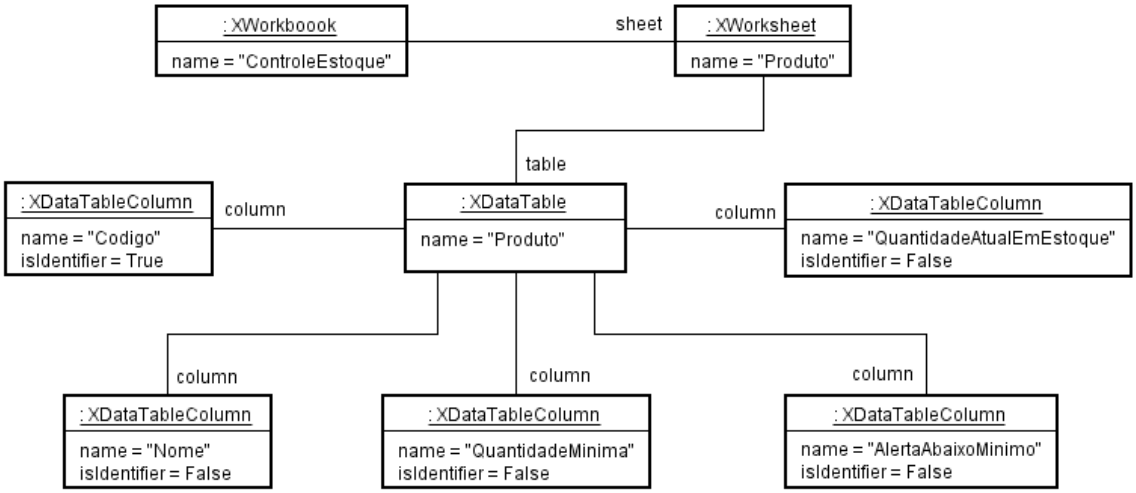


Figura 25 – Diagrama de Objetos - Operações de Produto

A Figura 25 mostra um diagrama de objetos referente à transformação das operações da classe *Produto* da Figura 12 em elementos do modelo de planilha. As operações *QuantidadeEmEstoque* e *AlertaAbaixoMinimo* foram transformadas em colunas da tabela Produto com o mesmo nome da operação e com o atributo *isIdentifier* igual a *False*, significando que não são colunas identificadoras.

3.4.6 **Tradução de Associações**

De acordo com (Heuser, 2009), o fator determinante para a tradução escolhida no caso de relacionamentos é a cardinalidade mínima e máxima das entidades que participam do relacionamento. O autor apresenta três formas de tradução de relacionamentos que foram adaptadas para este trabalho. São elas: tabela própria, colunas adicionais dentro de tabelas e fusão de tabelas. No caso de diagramas de classes, o relacionamento se traduz em associação e a cardinalidade mínima e máxima se traduzem em multiplicidade. Essa correspondência é apresentada na Tabela 1.

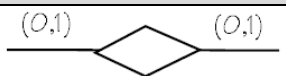
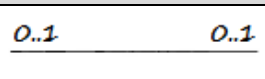
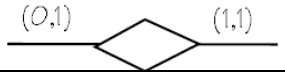
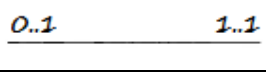
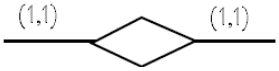
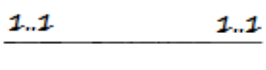
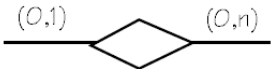
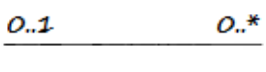
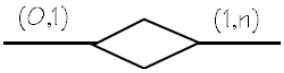
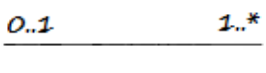
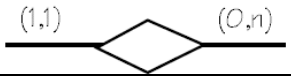
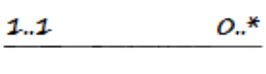
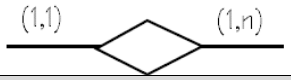
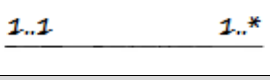
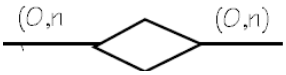
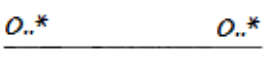
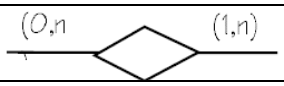
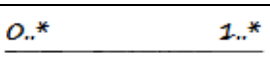
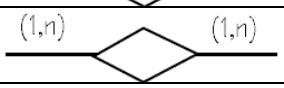
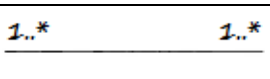
No primeiro caso, a associação é implementada através de uma tabela própria. Essa tabela contém colunas correspondentes aos identificadores das classes relacionadas (colunas-referência), onde cada um desses identificadores se traduz em uma coluna que faz referência à tabela que implementa a classe referenciada, e colunas correspondentes aos atributos da associação. A identificação das instâncias dessa tabela é o conjunto das colunas correspondentes aos identificadores das classes relacionadas.

A segunda alternativa de implementação de uma associação é a inserção de colunas em uma tabela correspondente a uma das entidades que participam do relacionamento. Esse tipo de tradução somente é possível quando uma das entidades que participa do relacionamento tem cardinalidade máxima 1 (um). A tradução consiste em inserir na tabela correspondente à entidade com cardinalidade máxima 1 (um) colunas correspondentes ao identificador da entidade relacionada, que formam uma chave estrangeira em relação à tabela que implementa a entidade relacionada, e colunas correspondentes aos atributos do relacionamento.

A terceira e última forma de implementar um relacionamento é por meio da fusão das tabelas referentes às entidades envolvidas no relacionamento. A tradução consiste em implementar todos os atributos de ambas entidades, bem como os atributos do relacionamento, em uma única entidade.

A Tabela 1 apresenta uma visão geral das regras utilizadas em nossa proposta. Essa tabela foi adaptada de (Heuser, 2009) e as respectivas equivalências entre o MER e o modelo de classes UML foram acrescentadas.

Tabela 1 – Regras de transformação de associações (adaptada de Heuser, 2009)

Tipo de Relacionamento		Regras de implementação		
MER	*UML*	*Tabela própria*	*Adição de colunas*	*Fusão de tabelas*
Relacionamentos 1:1				
		±	✓	X
		∓	±	✓
		∓	∓	✓
Relacionamentos 1:n				
		±	✓	X
		±	✓	X
		∓	✓	X
		∓	✓	X
Relacionamentos n:n				
		✓	X	X
		✓	X	X
		✓	X	X

Para cada tipo de relacionamento, a tabela mostra a alternativa que deve ser usada (alternativa preferencial, indicada pelo símbolo ✓). Para alguns tipos de relacionamentos, existem outras alternativas que geram uma implementação correta, mas que por princípios do projeto lógico que o autor se baseia, não constituem a melhor implementação (indicadas pelos símbolos ± e ∓, em ordem decrescente de preferência de uso), bem como uma alternativa que não deve ser usada (indicada pelo símbolo X), pois levam a construções inválidas.

Esses princípios acima citados se baseiam em evitar junções, ou seja, ter dados necessários a uma consulta em uma única linha, diminuir o número de chaves e evitar campos opcionais, ou seja, aqueles que podem assumir o valor vazio. Esses princípios podem ser adaptados para este trabalho no que diz respeito à visualização

e consulta dos dados nas tabelas do Excel. Porém, as regras de transformação propostas neste trabalho utilizam sempre a alternativa preferencial (✓).

No metamodelo da UML, conforme apresentado na Figura 12, uma associação é representada pelo elemento *Association*. Um *Association* tem dois ou mais *AssociationEnds* ligados a ele. Nesta abordagem, consideramos apenas associações binárias, ou seja, onde existem apenas dois *AssociationEnds*. Cada *AssociationEnd* está ligado ao *Classifier* relacionado e também à multiplicidade correspondente (*Multiplicity*). Cada *Multiplicity* define uma cardinalidade mínima (*lowerValue*) e uma cardinalidade máxima (*upperValue*).

Um *Classifier* mantém uma lista chamada *associationEnds* que contém todas as associações das quais esse *Classifier* participa. Sendo assim, a seguinte regra foi definida para transformação de associações:

> Para cada *Classifier* c que não possuir uma ligação com um *Stereotype* s tal que s.name = '*Enumeration':*
>
> Para cada *AssociationEnd assocEnd* em *c.associationEnds:*
>
> Se *assocEnd.multiplicity.upperValue* = 1
>
> criar uma nova coluna na tabela correspondente ao *Classifier* de *assocEnd* com o nome do *Classifier* c.
>
> Fim Se

Para exemplificar um dos tipos de transformação de relacionamentos, vamos ilustrar o mais utilizado. Considere a associação entre a classe *Departamento* e a classe *Produto* na Figura 12. Nesse caso, um departamento possui um ou mais produtos. De acordo com a Tabela 1, essa associação será traduzida na adição de uma coluna na tabela *Produto* chamada *Departamento*, correspondente ao nome do papel que *Departamento* assume na associação com Produto. A Figura 26 apresenta o diagrama de objetos referente à tabela *Produto* após essa transformação.

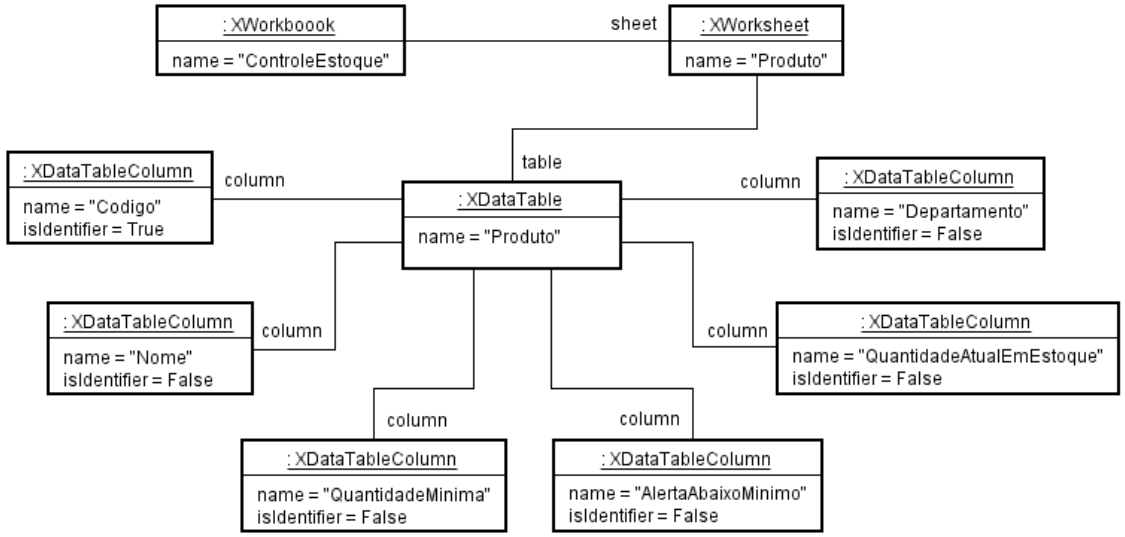


Figura 26 – Exemplo de transformação de associação (1..*)

3.5 Transformação de Expressões OCL

Nesta seção são descritas as regras de transformação de expressões OCL para as fórmulas da planilha. Uma expressão OCL é transformada em uma fórmula de planilha em formato texto por meio de regras de transformação aplicadas a cada nó da árvore de instâncias do metamodelo da OCL correspondente à expressão. A Figura 27 apresenta alguns exemplos de expressões OCL para o diagrama da Figura 9 que serão utilizados nas subseções seguintes.

```
01  context Compra::CompraValida() : Boolean
02  body: Quantidade > 0 and Quantidade <= 10
03
04  context Compra::PrecoTotal() : Real
05  body: PrecoUnitario * Quantidade
06
07  context Produto::TotalCompras() : Real
08  body: Compras.Quantidade->sum()
09
10  context Produto::TotalComprasAcima20() : Integer
11  body: Compras->select(c | c.Quantidade > 20)->size()
12
13  context Produto::Alerta() : String
14  body: if QuantidadeEmEstoque() < QuantidadeMinima
15        then "Comprar" else "Suficiente" endif
16
17  context Produto::QuantidadeEmEstoque() : Integer
18  body: Compras.Quantidade->sum() - Vendas.Quantidade->sum()
19
20  context Departamento::TotalVendas() : Real
21  body: Produtos.Vendas.PrecoTotal()->sum()
```

Figura 27 – Expressões OCL

As subseções a seguir apresentam como cada tipo de expressão OCL é transformado em uma parte da fórmula final na planilha.

3.5.1 **Expressões Literais (LiteralExp)**

Um nó do tipo *LiteralExp* corresponde a uma expressão literal, ou seja, uma expressão que contém um valor de um tipo primitivo. Um *LiteralExp* pode ser um *BooleanLiteralExp*, um *RealLiteralExp*, um *IntegerLiteralExp* ou um *StringLiteralExp*, que correspondem a valores dos tipos primitivos *Boolean*, *Real*, *Integer* e *String*, respectivamente. Essas expressões são transformadas diretamente para o texto correspondente na fórmula. No caso específico de um *StringLiteralExp*, o texto é colocado entre aspas (“”).

A expressão apresentada na Figura 27, linhas 01-02 (`Quantidade > 0 and Quantidade <= 10`), retorna verdadeiro se uma compra tiver entre 1 (um) e 10 (dez) produtos, inclusive. Os números *0* e *10* presentes nessa condição são expressões literais do tipo *Integer*, sendo representadas, portanto, por instâncias de *IntegerLiteralExp*. Essas instâncias são traduzidas diretamente para os textos *0* e *10* na fórmula da planilha.

A expressão condicional apresentada nas linhas 13-15 (`if QuantidadeEmEstoque() < QuantidadeMinima then "Comprar" else "Suficiente" endif`) retorna o valor “Comprar”, se o número de produtos em estoque for menor do que a quantidade mínima exigida para aquele produto ou, caso contrário, o valor “Suficiente”. Os valores *“Comprar”* e *“Suficiente”* são representados por instâncias de *StringLiteralExp*. Logo, elas são traduzidas para os textos entre aspas *“Comprar”* e *“Suficiente”*.

3.5.2 **Expressões de Variáveis (VariableExp)**

Um nó do tipo *VariableExp* é uma expressão que está associada a uma variável de um determinado tipo, representado por um *Classifier* do modelo. Uma expressão desse tipo é transformada em uma referência à tabela correspondente ao tipo da variável à qual a expressão está associada.

A expressão apresentada nas linhas 04-05 (`PrecoUnitario * Quantidade`) declara que a operação *PrecoTotal* da classe *Compra* retorna um número *Real* que é o resultado da multiplicação dos atributos *PrecoUnitario* e *Quantidade* da mesma classe. Nessa expressão, *PrecoUnitario* é uma abreviação para a expressão *self.PrecoUnitario*, o mesmo ocorrendo com *Quantidade*. Como a variável *self* é do

tipo Compra, ela será traduzida para uma referência à tabela Compra, que corresponde à classe Compra.

3.5.3 **Expressões de Chamada a Atributos (AttributeCallExp)**

Um nó *AttributeCallExp* corresponde a uma expressão que contém uma chamada a um atributo de um *Classifier* do modelo. Ela pode ser uma chamada a um atributo da classe ou a um atributo de uma instância, que é a expressão fonte (source) dessa expressão. No escopo deste trabalho só foram tratadas transformações de atributos de instâncias. Essas chamadas a atributos estão sempre no seguinte formato: <*source*>.<*attribute*>. No caso de chamadas a atributos de instâncias, o source é sempre uma expressão do tipo *VariableExp*, cuja transformação foi apresentada na seção 3.5.2. Já o atributo é transformado em uma referência a uma coluna de uma tabela, que corresponde a um atributo de um *Classifier*, A transformação desse tipo de expressão gera um texto no seguinte formato: <source>[<attribute>], onde <source> é o resultado do processamento do nó fonte da expressão e <attribute> é o resultado da transformação de um atributo.

Voltando à expressão das linhas 04-05 (`PrecoUnitario * Quantidade`), relembramos que *PrecoUnitario* é uma abreviação para a expressão *self.PrecoUnitario*, o mesmo ocorrendo com *Quantidade*, sendo que ambas correspondem a instâncias de *AttributeCallExp*. Como a fonte das duas expressões é a variável *self* e ela é uma expressão do tipo VariableExp, o resultado do processamento do source será o resultado do processamento dessa expressão. Depois disso, o atributo é traduzido. O atributo PrecoUnitario é traduzido em uma referência à coluna PrecoUnitario e o atributo Quantidade em uma referência para a coluna Quantidade, ambas da tabela Compra, que correspondem aos atributos PrecoUnitario e Quantidade da classe Compra. Dessa forma, essas expressões são traduzidas para *Compra[PrecoUnitario]* e *Compra[Quantidade]*, respectivamente.

3.5.4 **Expressões de Chamada a Operações (OperationCallExp)**

Um *OperationCallExp* é uma expressão que contém uma chamada a uma operação de um *Classifier*. Um OperationCallExp contém uma expressão fonte (*source*) e uma lista de argumentos (*arguments*), que também são expressões. De acordo com a operação, transformações diferentes são feitas.

i) **Operações aritméticas** (+,-,*,/) e **relacionais** (>,<,<=,<=,=,<>) são transformadas diretamente nos respectivos operadores nas fórmulas. O resultado da

transformação da expressão fonte da operação será inserido no lado esquerdo do operador na fórmula e o resultado da transformação do argumento será inserido do lado direito, de acordo com o seguinte formato: <*source*> <*operation*> <*argument*>;

Voltando ao exemplo das linhas 04-05 (`self.PrecoUnitario * self.Quantidade`), a operação aritmética '*' (multiplicação) é representada por uma instância da classe *OperationCallExp*. Se aplicarmos a regra definida anteriormente, esta expressão é transformada diretamente para o texto '*', pois operações aritméticas também são declaradas dessa forma no Excel. Sendo assim, podemos mostrar como a expressão inteira será transformada em uma fórmula. Essa fórmula será criada na coluna *PrecoTotal* da tabela *Compra*. A fórmula declara que o resultado da célula daquela coluna é a multiplicação entre o valor da coluna *PrecoUnitario* e o valor da coluna *Quantidade*[2] (Figura 28).

```
=Compra[PrecoUnitario]*Compra[Quantidade]
```

Figura 28 – Exemplo de fórmula no Excel (multiplicação)

Voltando ao exemplo das linhas 01-02 (`Quantidade > 0 and Quantidade <= 10`), as operações relacionais '>' e '<=', também são representadas por instâncias da classe *OperationCallExp*. Aplicando a regra definida anteriormente, elas também são transformadas diretamente para os textos '>' e '<='. Do lado esquerdo do operador, ficará o resultado do processamento das fontes da operação. Em ambos, a fonte é uma expressão do tipo *AttributeCallExp*, cuja transformação foi apresentada na seção 3.5.3. Do lado direito do operador, ficará o resultado do processamento do argumento da operação. Nos dois casos, o argumento é uma expressão do tipo *StringLiteralExp*, cuja transformação foi apresentada na seção 3.5.1. O resultado dessas duas transformações será *[Quantidade] > 0* e *[Quantidade] <= 10*, respectivamente.

ii) **Operações booleanas** (and e or): são transformadas nas funções AND() e OR() utilizadas nas fórmulas do Excel. O resultado da transformação da fonte será inserido como primeiro argumento da função. O resultado do processamento do

[2] Quando uma fórmula faz referência para colunas da mesma tabela, o próprio Excel simplifica a referência estruturada de forma que só o nome da coluna é declarado. No exemplo da Figura 28, a fórmula seria simplificada para "=[PrecoUnitario]*[Quantidade]".

argumento será inserido como segundo argumento da função. O formato da fórmula será o seguinte: <operation> (<source>, <argument>);

Ainda no exemplo das linhas 01-02 (`Quantidade > 0 and Quantidade <= 10`), temos a operação booleana 'and', também representada por uma instância da classe *OperationCallExp.* Aplicando a regra para operações booleanas, essa expressão será traduzida para uma função do Excel chamada AND(). Esta função recebe N parâmetros. Os parâmetros passados para essa função serão os resultados do processamento da expressão fonte (source) e do argumento (argument) da expressão. A fonte e o argumento dessa expressão são as expressões com as operações condicionais '>' e '<=' apresentadas anteriormente. A fórmula resultante é apresentada na Figura 29. Esta fórmula será criada na coluna CompraValida da tabela Compra.

```
=AND([Quantidade]>0,[Quantidade]<=10)
```

Figura 29 – Exemplo de fórmula no Excel (AND)

iii) **Operações em coleções:** são transformadas nas funções correspondentes:

- A operação Sum() é transformada na função SUMIFS(), onde especifica-se uma condição que indica se um determinado elemento da coleção deve participar do somatório;
- A operação Size() é transformada na função COUNTIFS(), também com uma condição que indica a participação dos elementos na contagem.

A expressão das linhas 07-08 (`Compras.Quantidade->sum()`) representa o somatório da quantidade de compras de um determinado produto. Essa expressão é a abreviação da expressão `Compras->collect(c | c.Quantidade)->sum()`. Essa expressão é uma instância de *OperationCallExp*. Utilizando a regra de transformação apresentada, essa operação será transformada na função SUMIFS, que espera receber três parâmetros. O primeiro parâmetro é o intervalo que queremos fazer o somatório, que será o resultado do processamento da expressão fonte da operação (`Compras->collect(c | c.Quantidade)`), que é uma expressão do tipo *IteratorExp*. O segundo e o terceiro parâmetros determinam o filtro a ser aplicado à essa soma, que será o resultado do processamento da expressão fonte do *IteratorExp,* que é uma

expressão do tipo *AssociationEndExp.* O resultado deste processamento será a declaração da fórmula: SUMIFS.

Considere agora outra expressão, variante da expressão declarada nas linhas 07-08: `Compras->size()`. Essa expressão faz uma contabilização do número de compras daquele produto. Essa expressão será transformada em uma função chamada COUNIFS. Essa função recebe dois parâmetros que ditam o filtro correspondente à contagem, que será o resultado do processamento da fonte do *IteratorExp*, que é uma expressão do tipo *AssociationEndExp*. O resultado deste processamento será a declaração da fórmula: COUNIFS.

iv) **Operações do modelo:** são transformadas em referências estruturadas a colunas da tabela correspondentes à operação do *Classifier,* de forma análoga à transformação de um *AttributeCallExp*.

Voltando à expressão das linhas 13-15 (`if QuantidadeEmEstoque() < QuantidadeMinima then "Comprar" else "Suficiente" endif`), temos uma chamada a uma operação do modelo *QuantidadeEmEstoque()*, que também é uma abreviação da expressão *self.QuantidadeEmEstoque()*. Essa transformação é muito similar à que foi feita para um AttributeCallExp. Essa expressão será transformada em uma referência a uma coluna de uma tabela, que representa a operação de um *Classifier*. Desse modo, a expressão resultante será *Produto[QuantidadeEmEstoque]*.

3.5.5 **Expressões Condicionais (IfThenElseExp)**

Um *IfThenElseExp* é uma expressão que contém um expressão condicional, que será transformada em uma chamada à função IF(). Essa função recebe três parâmetros. O primeiro parâmetro é a condição a ser avaliada, que será o resultado do processamento da expressão declarada na cláusula IF. O segundo parâmetro é o valor que deve ser retornado caso a condição seja verdadeira, que será o resultado do processamento da cláusula THEN. O terceiro parâmetro é o valor que deve ser retornado caso a condição seja falsa, que será o resultado do processamento da cláusula ELSE.

Voltando ao exemplo das linhas 13-15 (`if QuantidadeEmEstoque() < QuantidadeMinima then "Comprar" else "Suficiente" endif`), temos que criar uma função IF() com três parâmetros: o primeiro é o resultado do processamento de um *OperationCallExp*, que apresentamos na seção 3.5.4; o segundo é o resultado do processamento de um *StringLiteralExp*, que apresentamos na seção 3.5.1; e o

terceiro também é resultado do processamento de um *StringLiteralExp*. A fórmula resultante em Excel é apresentada na Figura 30. Esta fórmula será criada na coluna Alerta da tabela Produto.

```
=IF(Produto[QuantidadeEmEstoque]<Produto[QuantidadeMinima],"Comprar","Suficiente")
```

Figura 30 – Exemplo de fórmula no Excel (IF)

3.5.6 **Expressões de Navegação (AssociationEndCallExp)**

Um *AssociationEndCallExp* é uma expressão que contém uma chamada a um *AssociationEnd.* Relembrando, um *AssociationEnd* é uma das pontas de uma associação entre dois ou mais *Classifiers*. Dependendo do tipo da cardinalidade máxima da multiplicidade do *AssociationEnd*, a transformação é diferente. A princípio só foram tratadas transformações de associações de cardinalidade '*'.

i) No caso de cardinalidade '*', o *AssociationEnd* é transformado em um par referência a atributo-chave/referência a chave-estrangeira no formato: *A[B],C[D]*, onde:

- A é o nome da tabela correspondente ao *Classifier* do *AssociationEnd*;
- B é o nome da coluna correspondente à associação na tabela que descreve o *Classifier* do *AssociationEnd*;
- C é o nome da tabela correspondente ao *Classifier* do *AssociationEnd oposto; e*
- D é o nome da coluna chave da tabela correspondente ao Classifier do AssociationEnd oposto.

Voltando à expressão das linhas 07-08 (`Compras.Quantidade->sum()`), 'Compras' é o nome do papel que a classe Compra assume no relacionamento com a classe Produto. Ele corresponde a uma instância de *AssociationEndCallExp.* Analisando a multiplicidade da associação, verificamos que o valor máximo da cardinalidade do lado de Compra é '*'. Aplicamos a regra de transformação e a transformamos no par de referências *'Compras[Produto],Produto[Codigo]',* onde Compras corresponde ao nome da tabela do Classifier do AssociationEnd, [Produto] corresponde à coluna que representa a associação entre os dois, Produto corresponde ao Classifier do AssociationEnd oposto, e o [Codigo] é a coluna chave daquela tabela.

Agora considere a expressão das linhas 20-21 (`Produtos.Vendas.PrecoTotal()->sum()`). Essa expressão declara que o método TotalVendas da classe Produto retorna um número Real e que ele é igual à soma do preço total das vendas de seus

produtos. Essa expressão contém uma navegação em mais de um nível, ou seja, navega por duas associações do modelo (Departamento-Produto e Produto-Venda) Para traduzirmos essa navegação em dois níveis para o Excel, temos que realizar dois passos.

No primeiro passo, temos que criar uma nova coluna na tabela de Vendas que corresponde ao departamento que aquela Venda pertence. Para isso, temos que navegar pelo sentido inverso das associações para recuperar o Produto que aquela Venda pertence (coluna Produto da tabela Venda) e depois o Departamento que aquele Produto pertence (coluna Departamento da tabela Produto). .

Este processo de descoberta é realizado através da combinação de duas funções do Excel, chamadas de INDEX e MATCH. A fórmula correspondente em Excel é apresentada na Figura 31.

```
=INDEX(Produto,MATCH([Produto],Produto[Codigo],0),COL(Produto[Departamento]))
```

Figura 31 – Exemplo de fórmula no Excel (relacionamentos em 2 níveis)

A função INDEX recebe 3 parâmetros. O primeiro parâmetro é a tabela onde está o elemento que deseja buscar. No nosso caso, esta tabela é a tabela Produto. O segundo parâmetro é o índice da linha onde se encontra o elemento que deseja. O terceiro parâmetro é o índice da coluna onde se encontra o elemento que deseja. Para recuperar a linha onde se encontra o produto, utiliza-se a função MATCH. Esta função recebe igualmente 3 parâmetros. O primeiro é o valor do elemento que deseja procurar, o segundo é a coluna onde deseja procurar e o terceiro é para informar se deseja a correspondência exata (0), ou entre valores (1). Essa função retorna o índice da linha onde se encontra o elemento. Para recuperar a coluna que desejamos, utilizamos a função COL, que recebe como parâmetro uma coluna de uma tabela e retorna o índice correspondente a ela. Com todas essas informações, esta fórmula retorna o departamento correspondente àquela compra.

No segundo passo, é criado um par de referências, conforme apresentado na regra de transformação. A diferença neste caso é que a coluna chave-estrangeira utilizada na transformação é que foi criada no passo anterior. A fórmula será criada na tabela Departamento, coluna TotalVendas (Figura 32).

```
=SUMIFS(Vendas[PrecoTotal],Vendas[Departamento],[Codigo])
```

Figura 32 – Exemplo de fórmula no Excel (relacionamentos em 2 níveis)

3.5.7 **Expressões de Iteração (IteratorExp)**

Um *IteratorExp* é uma expressão que navega em uma coleção para coletar ou filtrar resultados. As duas operações responsáveis por isso são o COLLECT e o SELECT. Operações COLLECT são transformadas em uma chamada ao elemento que a expressão está querendo coletar. Caso seja um atributo, a transformação se dá da mesma forma que um *AttributeCallExp*. Caso seja uma operação de um Classifier do modelo, a transformação se dá da mesma forma que para um *OperationCallExp*.

Voltando à expressão das linhas 07-08 (`Compras->collect(c | c.Quantidade)->sum()`), podemos citar que a operação *collect* recupera o valor do atributo Quantidade de todos os elementos da coleção Compras. A aplicação dessa operação é uma instância da expressão *IteratorExp*. Aplicando a regra de transformação, temos o seguinte resultado: Compra[Quantidade], que corresponde à coluna Quantidade da tabela Compra. A fórmula resultante dessa expressão é apresentada na Figura 33. Esta fórmula soma os elementos da coluna Quantidade da tabela Compra que tenham a coluna Produto igual ao Código do Produto que está realizando a soma. Esta fórmula será criada na coluna TotalCompras da classe Produto.

```
=SUMIFS(Compra[Quantidade],Compra[Produto],Produto[Codigo])
```

Figura 33 – Exemplo de fórmula no Excel (SUMIFS)

Considerando a mesma expressão, uma variante dela seria: `Compras->collect(c | c.PrecoTotal())->sum()`). Agora estamos querendo coletar o resultado de uma operação. Da mesma forma, isso será transformado em uma referência a uma coluna da tabela, neste caso, a coluna PrecoTotal da classe Compra. A fórmula resultante está expressa na Figura 34.

```
=SUMIFS(Compra[PrecoTotal],Compra[Produto],Produto[Codigo])
```

Figura 34 – Exemplo de fórmula no Excel (SUMIFS)

Considere a expressão das linhas 10-11 (`Compras->select(c | c.Quantidade > 20)->size()`). Essa expressão declara que o método TotalComprasAcima20 da classe Produto retorna um número Inteiro e contabiliza quantos produtos tiveram compras acima de 20 unidades. Em nosso caso, o primeiro intervalo de comparação corresponde à coluna Produto da tabela Compras. Essa coluna mantém a referência ao produto à que aquela compra pertence. O segundo intervalo é a coluna

Quantidade da tabela Compras que deve ser maior do que 20, correspondente a *&">20"* na notação do Excel. Ou seja, essa fórmula só fará a contabilização do número de produtos onde o valor correspondente ao produto seja igual ao código do produto correspondente e a quantidade vendida seja superior a 20. A fórmula correspondente é apresentada na Figura 35. O processamento da expressão interna ao SELECT é transformada em um par coluna/condição no formato do Excel de acordo com a operação. Foram tratadas somente operações relacionais para expressões internas ao SELECT.

```
=COUNTIFS(Compra[Produto],Produto[Codigo],Compra[Quantidade],&">20")
```

Figura 35 – Exemplo de fórmula no Excel (COUNTIFS)

3.6 Implementação da Proposta

Esta seção apresenta a implementação da proposta apresentada neste capítulo. Todas as transformações apresentadas foram implementadas utilizando a linguagem de programação C#.NET na ferramenta Visual Studio 2010. A visão geral dessa implementação é apresentada na Figura 36.

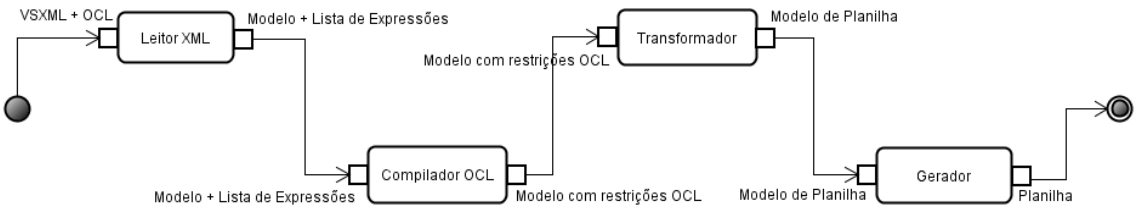


Figura 36 – Processo de Transformação

a) Leitor XML/OCL

Em um processo de automatização, o modelo deve estar representado através de um formato que possa ser interpretado. Existem algumas linguagens disponíveis que cumprem este papel, dentre elas está o XMI (XML MetadataInterchange), que é um formato baseado em XML para troca de informações, definido pela OMG (OMG, 2011b). Algumas ferramentas de modelagem possibilitam a criação do diagrama na forma gráfica e realizam a exportação para algum formato específico, como o XMI. Porém, o formato XMI possui diferentes versões, elaboradas e modificadas ao decorrer dos anos.

A ferramenta proprietária Visual Studio 2010, IDE (Integrated Development Environment) da Microsoft, na versão Ultimate, permite a criação de um projeto de modelagem de sistemas. Dentre os diagramas disponíveis dentro deste projeto, está o diagrama de classes UML. A ferramenta permite a criação do diagrama na forma gráfica e gera um arquivo no formato XML que representa os conceitos definidos na UML. A definição do formato deste arquivo se encontra em (Microsoft, 2010).

Em virtude de fatores como acesso a ferramentas com funcionalidade de exportação para XMI versão 2.1 e de não haver necessidade de utilizar representações tão complexas, o formato XML da Microsoft foi escolhido. Então, foi elaborado um algoritmo que interpreta este formato XML, que chamamos de VSXML, e cria instâncias do metamodelo da UML em memória. Além disso, foi elaborado um leitor de um arquivo OCL que lê o arquivo que contém a declaração das expressões e retorna uma lista de expressões no formato String.

b) Compilador OCL

Este componente é responsável por fazer a análise semântica das expressões OCL e transformá-las em ASTs (Abstract Syntax Trees), ou seja, em árvores de instâncias do metamodelo da OCL. Este compilador foi adaptado da ferramenta Odyssey-PSW, elaborada por (Correa, 2006), no contexto de sua tese de doutorado. Além disso, este componente vincula as expressões OCL aos elementos do modelo.

c) Transformador

Este componente é responsável por gerar o modelo de planilha correspondente ao modelo conceitual de domínio. Ele recebe como insumo o modelo com as expressões vinculadas e aplica as transformações definidas, produzindo um modelo de planilha correspondente.

d) Gerador

Este componente tem a responsabilidade de ler um modelo de planilha gerado pelo Gerador de Planilhas e produzir uma planilha resultante em Excel. Este componente utiliza a biblioteca de comunicação com o Excel (Microsoft.Office.Interop.Excel.dll), baseada em COM (Component Object Model) para gerar um arquivo .xlsx.

3.7 Considerações Finais

Neste capítulo foi apresentada a proposta deste trabalho para transformar modelos conceituais, expressos por diagramas de classe UML e expressões OCL, em planilhas eletrônicas. Foram definidas regras de transformação de elementos definidos no diagrama de classes para elementos da planilha. Além disso, também foram definidas regras de transformação de expressões OCL para fórmulas da planilha, que são a principal contribuição dessa proposta.

Capítulo IV – Estudo Experimental

O capítulo anterior apresentou a solução proposta para minimizar a ocorrência de erros em planilhas. Essa solução tem o objetivo de construir automaticamente uma planilha que evite erros em potencial. Então, foi realizado um estudo experimental que teve o objetivo de verificar e analisar os erros cometidos por usuários de planilhas em sua construção e traçar um paralelo com a planilha construída automaticamente pela proposta.

O restante do capítulo está dividido da seguinte forma: a seção 4.1 apresenta a definição formal do estudo e a abordagem utilizada. A seção 4.2 apresenta o planejamento do estudo, com o detalhamento dos participantes, a metodologia utilizada e o material proposto. A seção 4.3 descreve o experimento piloto realizado, bem como as alterações realizadas no material proposto para os demais participantes. A seção 4.4 apresenta as análises dos dados coletados no experimento. Finalmente, a seção 4.5 relata considerações finais sobre o capítulo.

4.1 Definição

O objetivo deste estudo experimental é avaliar se uma abordagem manual para construção de planilhas é capaz de alcançar resultados iguais ou melhores que os resultados gerados pela proposta apresentada no Capítulo 3 no que diz respeito ao número total de erros presentes na planilha resultante.

Dada uma planilha construída manualmente, para cada erro encontrado em uma de suas células será contabilizado 1 ponto. No final, o total de pontos será calculado para contabilizar o número de erros na planilha. Para classificar os diferentes tipos de erros foi utilizada a taxonomia apresentada por Panko e Aurigemma (2010), descrita na seção 2.1.1. Serão tratados somente erros não intencionais. Além disso, erros de deslize não foram considerados porque tipicamente são erros que usuários cometem na utilização da planilha. Ainda, a subdivisão dos erros de planejamento não foi considerada, ou seja, serão identificados somente erros de planejamento sem especificar se são erros de domínio ou planilha.

Como a abordagem parte do princípio de que o modelo de classes e as expressões OCL já foram elaborados, o esforço empregado na geração automática da planilha se torna irrelevante. Assim, não faz sentido comparar o esforço despendido para criar a planilha manualmente ou através da proposta apresentada no Capítulo 3: o tempo de geração pela ferramenta que implementa a proposta é quase instantâneo e o tempo gasto para construí-la manualmente certamente será maior. Sendo assim, neste estudo consideramos apenas a qualidade do produto final (a planilha) e não o esforço necessário para produzi-lo. A seguir, o objetivo do estudo é apresentado de forma estruturada, seguindo o modelo GQM – Goal/Question/Metric (Basili *et al.*, 1994).

Analisar *a construção de uma planilha*
Com o objetivo de *caracterizar a utilidade do apoio proposto pela abordagem de geração automática de planilhas a partir de modelos de domínio*
Com relação à *redução do número de erros introduzidos na planilha*
Do ponto de vista do *pesquisador*
No contexto de *alunos de graduação e pós-graduação em Computação, assim como profissionais da área de Engenharia de Software e/ou Finanças.*

4.2 Planejamento

Esta seção apresenta o planejamento do estudo. A seção 4.2.1 descreve o perfil dos participantes do estudo. A seção 4.2.2 descreve as etapas de construção das planilhas e os erros que podem ser introduzidos em cada uma, categorizados de acordo com a taxonomia de Panko e Aurigemma (2010). A seção 4.2.3 apresenta a questão de estudo e hipóteses relacionadas. A seção 4.2.4 apresenta os instrumentos utilizados no estudo. A seção 4.2.5 descreve os procedimentos utilizados. A seção 4.2.6 discute as ameaças à validade do estudo.

4.2.1 **Participantes**

Para o cenário do estudo, é preciso definir o papel de cada um dos envolvidos. Temos dois papéis: o cliente, que precisa de uma planilha para os usuários de sua empresa, e o desenvolvedor, que construirá a planilha de forma manual para atender a essa necessidade. O cliente é representado pelo pesquisador, que apresenta o minimundo do problema em questão e responde perguntas feitas pelos desenvolvedores. Os desenvolvedores são os participantes do estudo, que devem construir a planilha de

forma manual. No estudo, envolvemos pessoas com conhecimento limitado sobre o uso de planilhas eletrônicas, bem como pessoas mais experientes.

Os participantes foram requisitados a realizar a atividade de construção de uma planilha para controle de estoque de um minimercado a partir dos requisitos de negócio (descrição textual) e seu respectivo diagrama de classes UML definidos em laboratório (estudo In-Vitro). Os mesmos requisitos e o mesmo diagrama foram apresentados a todos os participantes e eles tiveram que construir a planilha correspondente ao problema proposto. As planilhas construídas pelos participantes foram comparadas à planilha gerada automaticamente pela proposta.

4.2.2 **Etapas de construção das planilhas e Erros Presentes**

Uma planilha é construída em etapas. Em cada uma dessas etapas, erros podem ser cometidos. A seguir, essas etapas serão descritas e alguns erros que podem ser introduzidos pelos desenvolvedores serão apontados, bem como sua correspondência na taxonomia de Panko e Aurigemma (2010).

Na primeira etapa o usuário constrói a estrutura da planilha, com suas abas e tabelas. Esta estrutura deve conter os conceitos presentes na descrição textual e no diagrama de classes e seus respectivos atributos e métodos. São eles:

- **Loja** (Código, Nome, Localização, Lucro, Total Compras, Total Vendas);
- **Departamento** (Código, Nome, Número Unidades Produtos, Total Compras, Total Vendas);
- **Produto** (Código, Nome, Quantidade Mínima, Alerta Abaixo do Mínimo, Quantidade em Estoque);
- **Compra** (Código, Data, Fornecedor, Preço Unitário, Quantidade, Preço Total);
- **Venda** (Código, Data, Cliente, Preço Unitário, Quantidade, Preço Total).

Para cada um dos atributos ou métodos destes conceitos, os seguintes erros podem ser cometidos:

- Atributo ou método do conceito não foi construído *(Lapso);*
- Atributo ou método do conceito foi construído de forma errada *(Planejamento)*: conceitos foram construídos de forma não escalável[3] para

[3] Por exemplo, considerando a relação entre Departamentos e Produtos, uma mesma tabela poderia ser utilizada para registrar estas duas informações. No entanto, se esta abordagem for utilizada, o

inserção de dados, ou seja, usuário terá que alterar a estrutura da planilha para inserir novos dados.

Na segunda etapa o usuário constrói as associações entre os conceitos, através de referências na planilha. Nesta etapa o usuário pode cometer, dentre outros, os seguintes erros, classificados por conceitos.

- **Departamento**
  - Não construiu associação Loja-Departamento *(Lapso);*
  - Não construiu associação com CategoriaDepartamento *(Lapso)*;
  - Construiu de forma errada associação Loja-Departamento *(Planejamento)*;
  - Construiu de forma errada associação com CategoriaDepartamento *(Planejamento);*
  - Coluna Loja não apresentou lojas existentes *(Qualitativo);*
  - Coluna Categoria não apresentou categorias existentes *(Qualitativo);*
  - Validação de dados de Loja não será replicada na inclusão de novo registro *(Qualitativo);*
  - Validação de dados de Categoria não será replicada na inclusão de novo registro *(Qualitativo).*
- **Produto**
  - Não construiu associação Departamento-Produto *(Lapso)*;
  - Construiu de forma errada associação Departamento-Produto *(Planejamento)*;
  - Coluna Departamento não apresentou departamentos existentes *(Qualitativo)*;
  - Validação de dados de Departamento não será replicada na inclusão de novo registro *(Qualitativo)*.
- **Compra**
  - Não construiu associação Produto-Compra *(Lapso)*;
  - Construiu de forma errada associação Produto-Compra *(Planejamento)*;

registro de novos produtos em um departamento exigirá a mudança na estrutura da planilha, possivelmente gerando problemas em referências vindas de outras tabelas da planilha.

    - Coluna Produto não apresentou lista de produtos existentes *(Qualitativo)*;
    - Validação de dados de Produto não será replicada na inclusão de novo registro *(Qualitativo)*.
- **Venda**
    - Não construiu associação Produto-Venda *(Lapso)*;
    - Construiu de forma errada associação Produto-Venda *(Planejamento)*
    - Coluna Produto não apresentou lista de produtos existentes *(Qualitativo)*
    - Validação de dados de Produto não será replicada na inclusão de novo registro *(Qualitativo)*

Na terceira etapa o usuário constrói as fórmulas propostas na descrição textual, que correspondem aos métodos presentes nos conceitos. São eles:

- Fórmula que calcula o preço total de compras de uma loja;
- Fórmula que calcula o preço total de vendas de uma loja;
- Fórmula que calcula o lucro de uma loja;
- Fórmula que calcula o número de produtos de um departamento;
- Fórmula que calcula o preço total de compras de um departamento;
- Fórmula que calcula o preço total de vendas de um departamento;
- Fórmula que calcula o preço total de uma compra;
- Fórmula que calcula o preço total de uma venda;
- Fórmula que calcula a quantidade em estoque de um produto;
- Fórmula que emite um alerta se o estoque de um produto estiver abaixo do mínimo.

Os seguintes erros são esperados para cada fórmula listada acima:

- Fórmula não foi construída *(Lapso)*
- Fórmula está errada *(Planejamento)*: a fórmula foi aplicada na planilha de forma errada. Por exemplo, esqueceu algum parâmetro etc.
- Fórmula não será replicada na inclusão de novo registro *(Qualitativo)*: quando um novo registro é incluído, a fórmula não é replicada.

4.2.3 **Questão do Estudo e Hipóteses Relacionadas**

Nesta seção apresentamos a questão de estudo que visamos responder com a execução do estudo experimental, bem como as hipóteses (nula e alternativa) correspondentes.

> **RQ1:** A construção manual de planilhas eletrônicas implica na introdução de um número de erros significativamente maior do que zero?

A questão de estudo se reduz a identificar se o número de erros nas planilhas construídas manualmente é significativamente diferente de zero. Para realizar esta avaliação, estabelecemos as hipóteses nula e alternativa abaixo.

Hipótese Nula (H0): o número de erros presentes nas planilhas geradas manualmente pelos participantes do estudo é igual à zero.

H0: $E_{manual} = 0$

Hipótese Alternativa (H1): o número de erros presentes nas planilhas geradas manualmente pelos participantes do estudo é maior do que zero.

H1: $E_{manual} > 0$

Para avaliar estas hipóteses envolvemos diversos participantes e contabilizamos o número de erros em suas planilhas conforme explicado na seção 4.1. Em seguida, avaliamos se os dados seguiram uma distribuição normal e aplicamos um teste de inferência estatística de acordo com esta distribuição para avaliar se o número de erros é significativamente maior que zero. Utilizamos testes não paramétricos de *Wilcoxon-Mann-Whitney* para dados que não seguiam uma distribuição normal e *testes T* para dados que seguiam esta distribuição. Conclusões serão apresentadas com nível de confiabilidade de 95% ($\alpha = 0.05$) e os *p-values* dos testes estatísticos serão apresentados.

4.2.4 **Instrumentação**

O primeiro instrumento utilizado foi o formulário de caracterização dos participantes (Anexo A). Neste formulário, o público-alvo foi qualificado quanto ao nível de formação, área de atuação e experiência em leitura, construção e manutenção de planilhas.

Outro instrumento utilizado foi o minimundo do problema abordado (Anexo B). Este instrumento é um documento em formato texto que descreve o que deve ser feito

na planilha, ou seja, os requisitos de negócio. Em conjunto com o minimundo foi disponibilizado um modelo de classes correspondente que também serviu de apoio para a construção da planilha (Anexo C).

Mais um instrumento utilizado foi o material de treinamento em construção de planilhas (Anexo D). Neste material, é apresentado ao participante como criar tabelas, validações de dados e fórmulas no Microsoft Excel. Este foi disponibilizado como uma forma de nivelamento de conhecimento na construção e manutenção de planilhas. Este instrumento foi distribuído em conjunto com o minimundo, como insumo para consulta antes e durante o experimento. Este material foi distribuído em formato audiovisual, explicando e exemplificando cada item. Os usuários puderam consultar os vídeos a qualquer momento da execução do experimento.

No fim do experimento foi feita uma avaliação dos erros presentes nas planilhas geradas pelos participantes, resultando em um novo instrumento: a planilha de contabilização de erros. Essa planilha tem uma representação bidimensional, onde a primeira dimensão define os tipos de erros que podem ocorrer e a segunda representa os participantes do estudo. Cada interseção indica se aquele participante cometeu aquele tipo de erro. Se tiver cometido, o valor será 1 (um); caso contrário, será 0 (zero).

4.2.5 **Procedimentos**

O procedimento para aplicação do estudo foi o seguinte. Em primeiro lugar, o pesquisador fez a apresentação do material a ser utilizado pelo participante e do objetivo do estudo. Depois disso, cada participante preencheu o questionário de caracterização. Em seguida, o participante leu a descrição textual do minimundo e o diagrama de classes, tirando quaisquer dúvidas com o pesquisador.

Após este processo, teve início a etapa de construção da planilha. Esta etapa foi dividida em três partes, conforme descrito na seção 4.2.2. Após o término dessa etapa, cada participante entregou a planilha elaborada ao pesquisador.

O pesquisador recolheu as planilhas elaboradas e aplicou os testes manualmente nas planilhas. Então, contabilizou os erros cometidos pelos participantes e preencheu a planilha de contabilização de erros. No fim, foi realizada a análise dos erros cometidos no decorrer da construção.

4.2.6 **Ameaças à Validade**

Segundo Wohlin *et al.* (2000), em todo estudo experimental existem ameaças à validade das conclusões que poderiam ser obtidas e, portanto, alguns cuidados precisam ser tomados.

Tentamos dar confiabilidade aos resultados deste estudo através da utilização de medidas objetivas e testes estatísticos não paramétricos e paramétricos, de acordo com a distribuição de frequência observada nos dados. Também foram utilizadas avaliações subjetivas para apoiar e explicar os resultados quantitativos. Embora o número de participantes (16) seja relativamente pequeno, tentamos gerar um número razoável de pontos de avaliação ao categorizar os erros possíveis e separá-los em etapas de construção.

Uma ameaça à validade do estudo é a subjetividade no que diz respeito à experiência em planilhas, visto que esse quesito, utilizado nas análises, foi preenchido por cada participante de acordo com o que ele acreditava ser seu conhecimento. Sendo assim, alguns participantes podem ter se apontado como experientes, mesmo sem que este nível de conhecimento pudesse ser comparável com outros com maior experiência na construção e manutenção de planilhas.

Outro detalhe é que utilizamos um único domínio de problema (controle de estoque de um minimercado), ou seja, caso o domínio fosse diferente os resultados poderiam ser diferentes.

4.3 Projeto Piloto

Antes de aplicar o estudo em maior escala, foi realizado um projeto piloto com o objetivo de avaliar o plano do estudo e os instrumentos construídos para ele, visando identificar falhas e corrigi-las. Este projeto piloto teve a participação de uma pessoa com as seguintes características: mestre, com experiência de 15 anos com desenvolvimento de sistemas e com pouca experiência no desenvolvimento de planilhas.

No inicio do projeto piloto foi feita a apresentação do objetivo do estudo e do material. Foi uma apresentação rápida, que durou cerca de 2 minutos. Depois disso, o material, composto pelo questionário de caracterização, a descrição textual do minimundo e seu respectivo modelo de classes, foi entregue ao participante. Com o material em mãos, ele começou a responder ao questionário. Logo depois, ele optou por realizar a leitura da descrição textual do problema. Durante o processo de leitura, realizado em voz alta, ele levantou algumas questões.

O atributo Categoria da classe Produto se confundiu com o relacionamento com a classe Departamento. A categoria estaria transversal ao departamento, gerando possíveis inconsistências no modelo. Por exemplo, poderíamos ter um Departamento de Higiene contendo produtos da Categoria Limpeza. Por este motivo, o atributo Categoria foi transferido para a classe Departamento e seu tipo foi alterado para representar a categoria do departamento, eliminando a inconsistência.

Uma segunda observação foi com relação à classe Venda. Como a classe Compra armazena a informação do fornecedor, seria bom que a classe Venda armazenasse a informação do cliente. Dessa forma, um novo atributo Cliente, do tipo *String*, foi acrescentado à classe Venda. Algo interessante, que foi igualmente considerado, foi acrescentar um novo atributo na classe Loja, indicando a sua localização. Logo, um atributo chamado Local foi adicionado na classe Loja.

Fixar os departamentos passou a falsa impressão de que estes não poderiam mudar. Desta forma, esta restrição foi retirada da descrição textual do minimundo. Além disso, a operação que calcula se um produto está abaixo de mínimo se confundiu com a operação que emite o alerta, de modo que as duas operações foram consolidadas em uma operação que emite um alerta caso o produto esteja com a quantidade abaixo do mínimo. Como última observação, não ficou claro na descrição que a operação que representa a quantidade em estoque de um produto é a diferença entre a quantidade comprada e a quantidade vendida. Para resolver esta questão, uma explicação detalhada foi acrescentada na descrição textual.

Após a leitura da descrição textual, o participante fez a leitura do modelo. Nesta leitura, ele fez uma comparação detalhada dos elementos presentes no modelo com a descrição textual. Após esta verificação, ele passou a utilizar somente o modelo para as etapas posteriores. Esta etapa durou cerca de 20 minutos.

Depois disso, o participante começou a etapa de construção da estrutura da planilha. Durante esta etapa, algo muito interessante foi observado. O modelo de planilha escolhido foi semelhante ao modelo da construção automática: um par aba/tabela para cada conceito. Isso pode ter ocorrido por ser a forma mais natural de pensar na estrutura da planilha tendo como base um modelo pré-definido. No decorrer da construção da estrutura, a maior dúvida foi associada ao relacionamento entre os conceitos. Mais uma vez, estes relacionamentos foram representados da mesma forma que a geração automática prevê, com um campo para representar o identificador do conceito relacionado. Talvez, a experiência prévia do usuário em

sistemas de informação e bancos de dados tenha influenciado nesta decisão. A etapa de construção da estrutura da planilha durou cerca de 20 minutos.

A etapa seguinte foi a de construção das fórmulas da planilha. Esta etapa foi a mais complicada, em virtude dos conhecimentos prévios necessários. Então, o pesquisador solicitou que o participante começasse pelas fórmulas para as quais tinha conhecimento e, para qualquer outra, que solicitasse ajuda a ele. Fórmulas simples em um mesmo contexto foram feitas sem problemas (por exemplo, as fórmulas que calculam o preço total de venda e compra). Porém, as fórmulas que envolvem relacionamentos entre conceitos exigiam um conhecimento ligeiramente mais complexo sobre a construção de fórmulas. Então, o pesquisador solicitou ao participante que descrevesse o que gostaria de realizar na planilha. Após esta descrição, o pesquisador revelou a fórmula necessária para atingir o objetivo. Por exemplo, considere a fórmula que calcula o total de vendas de um determinado produto. O participante descreveu que necessitava somar os elementos da coluna preço total apenas das vendas relacionadas ao Id daquele produto. Logo, foi apresentada a fórmula *SOMASES (SUMIFS)*, bem como seus parâmetros e seu resultado. Da mesma forma, foram apresentadas as fórmulas *SE (IF)* e *CONT.SES (COUNTIFS)*. Após aprender estes conceitos, o participante aplicou-os para resolver as outras fórmulas.

Depois disso, ele questionou sobre alguma forma de apresentar os dados de categoria a fim de evitar erros de inclusão de dados. Foi apresentado como criar listas de validação. Após isso, o participante aplicou nos demais pontos da planilha. Para que não houvesse interferência do pesquisador nos demais experimentos, foram disponibilizados vídeos que explicam as funcionalidades do Excel necessárias para realizar o estudo.

4.4 Análise

Esta seção apresenta as análises dos dados obtidos a partir da execução do estudo experimental. A seção 4.4.1 apresenta a caracterização dos participantes. A seção 4.4.2 apresenta uma análise dos erros cometidos organizados de acordo com sua categoria e a etapa de construção onde foram cometidos. A seção 4.4.3 apresenta análises de correlação dos dados sobre erros com a experiência dos usuários na construção de planilhas e em desenvolvimento de sistemas. Finalmente, a seção 4.4.4 apresenta os testes estatísticos para responder à questão de estudo.

4.4.1 **Caracterização dos Participantes**

O estudo foi realizado com participantes com características distintas com base nos seguintes critérios: nível de escolaridade, área de atuação e experiência em planilhas. Esta caracterização foi realizada por meio de um questionário individual (Anexo A). A Tabela 2 apresenta as características de cada um dos participantes conforme estes critérios.

A Tabela 3 apresenta a distribuição do número de participantes por formação. A maior parte dos participantes do estudo são estudantes de graduação (10). O restante deles está dividido entre Graduados (1), estudantes de Mestrado (2), Mestres (1), estudantes de Doutorado (1) e Doutores (1).

Tabela 2 – Caracterização dos Participantes

Participante	Formação	Atuação Profissional (anos)		Exp. Planilhas
		Análise e Desenvolvimento de Sistemas	*Infraestrutura*	
P01	Mestre	12	-	Alta
P02	Graduando	-	2	Pouca
P03	Graduando	1	-	Pouca
P04	Graduando	-	-	Pouca
P05	Graduando	-	7	Pouca
P06	Graduando	-	1	Média
P07	Graduando	2	-	Média
P08	Graduando	1	-	Muito Pouca
P09	Graduando	-	-	Pouca
P10	Graduando	1	-	Pouca
P11	Graduando	2	-	Alta
P12	Doutorando	3	-	Média
P13	Graduado	25	-	Pouca
P14	Doutor	12	-	Média
P15	Mestrando	9	-	Muito Alta
P16	Mestrando	4	-	Média

Tabela 3 – Número de Participantes por Formação

Número de participantes	Graduando	Graduado	Mestrando	Mestre	Doutorando	Doutor
16	10	1	2	1	1	1

A Tabela 4 e a Tabela 5 contabilizam o número de participantes por área de atuação e experiência na construção e manutenção de planilhas, respectivamente. A maioria dos participantes atua na área de Análise e Desenvolvimento de Sistemas

(11). O restante deles atua na área de Infraestrutura (3) ou não atua profissionalmente (2). Com relação à experiência em planilhas, a maioria deles tem pouca experiência em planilhas (7). Um (1) deles julga ter muito pouca experiência e o restante se julga mais experiente em construção de planilhas, atribuindo nota média (5), alta (2) ou muito alta (1) para este quesito.

Tabela 4 – Número de Participantes por Área de Atuação Profissional

Número de participantes	Não atua profissionalmente	Análise e Desenv. de Sistemas	Infraestrutura
16	2	11	3

Tabela 5 – Número de Participantes por Experiência em Planilhas

Número de participantes	Muito Pouca	Pouca	Média	Alta	Muito Alta
16	1	7	5	2	1

O participante P08 foi considerado um *outlier* por apresentar resultados bem diferentes do restante dos participantes e para que isso não afetasse o resultado final, ele foi retirado das demais análises.

4.4.2 **Erros cometidos**

Conforme apresentado na seção 4.2.2, uma planilha é construída em etapas. São elas: estruturação dos conceitos da planilha, construção das associações entre os conceitos e elaboração das fórmulas. Em cada uma dessas etapas, erros podem ser cometidos. Esses erros podem ser divididos em três categorias: Lapso, Planejamento ou Qualitativo. Por exemplo, um participante pode se esquecer de mapear algum conceito para a planilha, ou seja, um *Lapso* na etapa de *Estruturação*. A Tabela 6 apresenta o número máximo de erros que cada participante poderia cometer, classificados por categoria e etapa. O número que aparece no cruzamento entre a categoria e a etapa é o número potencial de erros nesse cruzamento. Não foram considerados erros *Qualitativos* na fase de *Estruturação*.

Tabela 6 – Número Máximo de Erros por Participante – Etapa x Categoria

Categoria/Etapa	Estrutura	Associação	Fórmulas	Total de Erros
Lapso	29	5	10	**44**
Planejamento	29	5	10	**44**
Qualitativo	-	10	10	**20**
Total de Erros	**58**	**20**	**30**	**108**

A Tabela 7 apresenta o número total de erros cometidos pelos participantes por etapa de construção e por categoria de erro. É importante notar que erros de *Lapso* originam erros nas demais categorias automaticamente, ou seja, se o participante deixou de mapear algum conceito, ele automaticamente cometeu os demais tipos de erros para aquele conceito. Por exemplo, se uma fórmula não foi feita *(Lapso)*, esta fórmula também está errada *(Planejamento)* e não será replicada na inclusão de um novo registro *(Qualitativo)*.

Tabela 7 – Número de Erros – Etapa x Categoria

Categoria/Etapa	Estrutura	Associação	Fórmulas	Total de Erros
Lapso	10	7	34	**51**
Planejamento	14	8	65	**87**
Qualitativo	-	121	93	**214**
Total de Erros	**24**	**136**	**192**	**352**

Podemos observar a partir dessa tabela que o maior número de erros se encontra na etapa de construção de fórmulas. De fato, foi observado que a maioria dos participantes teve dificuldade nesta etapa. Além de deixar de fazer algumas fórmulas, os participantes cometeram alguns erros na sua construção *(Planejamento)*. Dentre os erros mais frequentes estão: intervalo da fórmula não está fixado, intervalo não está correto e operação está errada.

Com relação ao intervalo não fixado, este erro é muito comum para casos onde o usuário deixa de utilizar o recurso de tabelas do Excel. Quando o intervalo não é fixado, utilizando o sinal '$', o usuário tem a falsa impressão de que a próxima linha manterá a referência para o intervalo declarado na linha anterior, gerando um erro difícil de ser detectado.

O erro de intervalo incorreto ocorre com muita frequência por desatenção do usuário. Este erro também é comum em casos onde o usuário não utiliza o conceito de tabelas. Isso acontece porque a identificação da coluna que será referenciada está no formato alfanumérico (p. ex., *A2:B2*), sendo difícil notar que este intervalo se refere à coluna X ou coluna Y da tabela.

No caso específico de operações erradas, isso se aplica normalmente para usuários que não têm conhecimento profundo sobre as fórmulas do Excel e utilizam funções de forma errada ou funções que não se aplicam àquele caso. Por exemplo, ao invés de utilizar a função SUMIFS, utiliza a função SUM. O resultado da função não fará o filtro necessário, resultando em valores errados.

Com relação ao tempo de execução do experimento, em média os participantes levaram de 1h30 a 2h para concluir o estudo, considerando a leitura do material e a construção da planilha.

4.4.2.1 **Erros por Categoria**

A Tabela 8 apresenta algumas medidas importantes sobre o número de erros cometidos por etapa. A tabela mostra a mediana, a média e o desvio padrão do número de erros cometidos por participante, bem como sua representação percentual. Esse percentual é relativo ao número máximo de erros que podem ser cometidos por cada participante (Tabela 6).

Tabela 8 – Contabilização de Erros por Categoria

	Lapso	Planejamento	Qualitativo	Total de Erros
Média	3,4 (7,7%)	5,8 (13,2%)	14,3 (71,3%)	**23,5 (21,8%)**
Desvio Padrão	3,9 (8,9%)	5,0 (11,3%)	4,6 (22,9%)	**11,3 (10,5%)**
Mediana	2,0 (4,5%)	6,0 (13,6%)	15,0 (75,0%)	**23,0 (21,3%)**

Como podemos observar, o número de erros Qualitativos é bem maior que os erros de Lapso e Planejamento. Isto pode ser notado com mais clareza a partir do gráfico de *boxplot* apresentado na Figura 39.

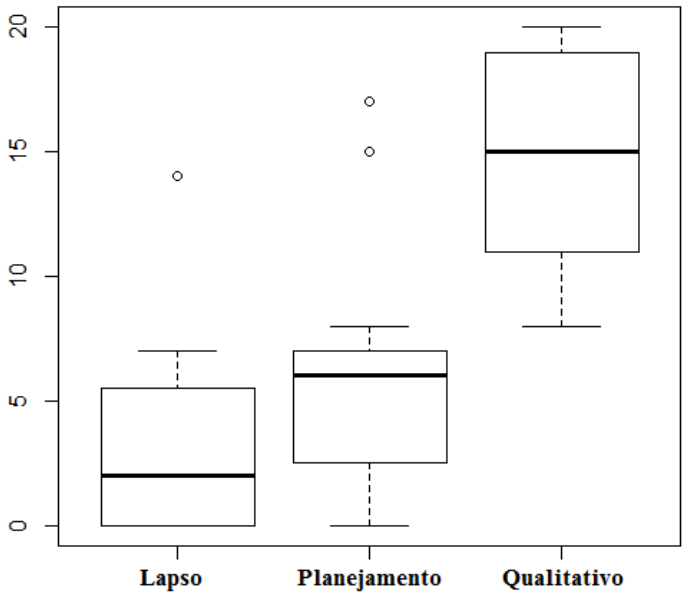


Figura 37 – *Boxplot* do Número de Erros por Categoria

Ainda, quando olhamos pelo ângulo dos valores percentuais que eles representam, os erros Qualitativos se mostram consideravelmente maiores dos que

os demais (Figura 38). Isto se deve pelo fato de que os participantes, em média, deixaram de fazer *(Lapso)* ou erraram *(Planejamento)* em pontos específicos, mas a maioria dos participantes atingiu pontuação quase máxima de erros Qualitativos.

Isso pode ser explicado pelo fato de que participantes não se atentaram ou não julgaram importantes detalhes de boas práticas de planilhas, como a utilização de tabelas e validadores de dados. A utilização de tabelas é de vital importância para ter certeza de que, quando um novo dado for inserido, o contexto de dados de uma determinada coluna será mantido, ou seja, se uma fórmula foi declarada na coluna, ela será replicada no próximo registro. Se a tabela não for utilizada, um erro ocorrerá assim que um novo dado for inserido.

No caso da criação de validadores de dados, eles são importantes para evitar que dados inconsistentes sejam inseridos na planilha. Por exemplo, as categorias de um Departamento podem ser Alimentação, Bebida, Higiene e Limpeza. Se o usuário errar na digitação (*Deslize*), os dados estarão inconsistentes e uma eventual agregação de dados pode causar resultados incorretos.

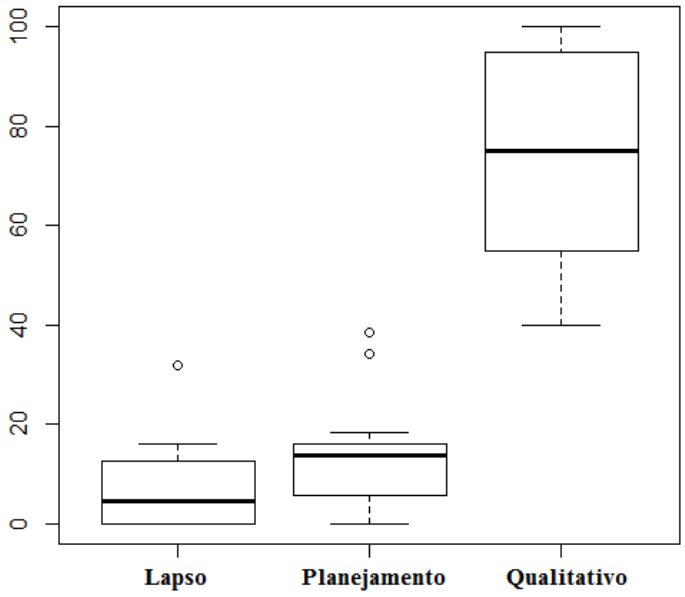


Figura 38 – Boxplot do Percentual do Número de Erros por Categoria

4.4.2.2 **Erros por Etapa**

A Tabela 9 apresenta algumas medidas importantes sobre o número de erros cometidos por etapa. A tabela mostra a mediana, a média e o desvio padrão do número de erros, bem como sua representação percentual. Esse percentual é relativo ao número máximo de erros possíveis, apresentado na Tabela 6.

Tabela 9 – Contabilização de Erros por Etapas

Medida	Estrutura	Associação	Fórmulas	Total
Média	1,6 (2,8%)	9,1 (45,3%)	12,8 (42,7%)	**23,5 (21,8%)**
Desvio Padrão	3,0 (5,2%)	3,0 (15,2%)	8,3 (27,7%)	**11,3 (10,5%)**
Mediana	0,0 (0,0%)	8,0 (40,0%)	16,0 (53,3%)	**23,0 (21,3%)**

Estes valores são importantes para notar, por exemplo, que erros na fase de estruturação não são comuns. Isto pode ser notado com mais clareza a partir do gráfico de *boxplot* apresentado na Figura 39.

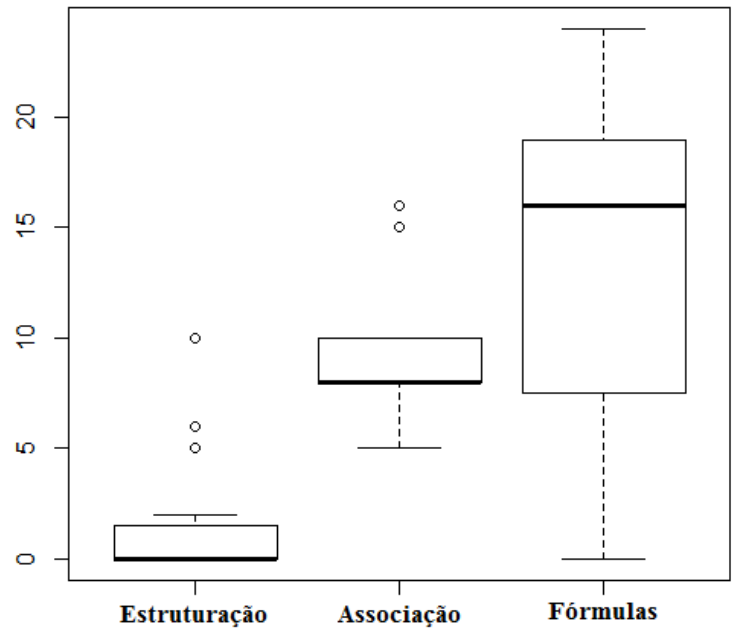


Figura 39 – *Boxplot* do Número de Erros por Etapas

A Tabela 9 também deixa claro que há uma concentração bem maior de erros na etapa de construção de fórmulas. Isto se deve ao fato de que os participantes encontraram mais dificuldade nesta etapa do processo de construção de planilhas. Porém, analisando o percentual que estes valores representam, podemos notar que os erros na etapa de associação também foram bem relevantes. Podemos observar melhor estes dados pela Figura 40, que apresenta o *boxplot* desses percentuais.

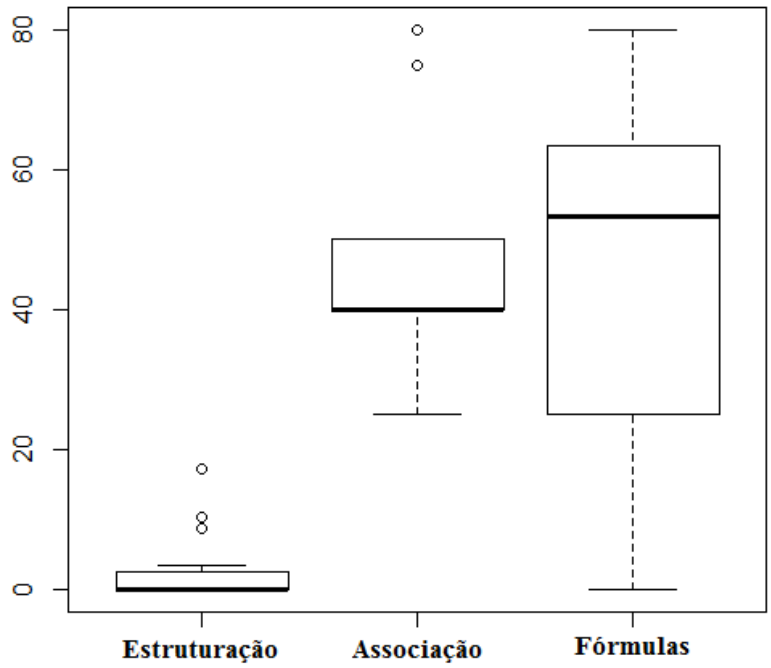


Figura 40 – *Boxplot* do Percentual do Número de Erros por Etapa

Estes valores ratificam que erros na fase de estruturação não são comuns. Além disso, como dito anteriormente, o percentual com o número de erros na fase de associação também é bastante relevante (40,0%) e chega a ficar próximo ao percentual do número de erros das fórmulas (53,3%). Mais uma vez, isto pode ter ocorrido devido ao fato de que a maioria dos participantes não fez validadores de dados ou utilizou tabelas do Excel.

4.4.3 **Correlação**

Nesta seção, será feita uma análise da correlação entre o perfil dos participantes com o número de erros cometidos por etapas e por categoria.

A Tabela 10 apresenta a correlação entre a experiência dos participantes do estudo experimental na construção de planilhas e o número de erros cometidos, seja o número total ou o número por tipo de erro. É possível observar que as correlações são todas negativas, indicando que quanto maior a experiência, menor é o número de erros cometido pelos usuários. A correlação é um pouco menor para o caso dos lapsos, indicando que sua introdução é mais independente da experiência.

Tabela 10 – Correlação entre Experiência e Número de Erros por Categoria

	Número Total de Erros	Número de Lapsos	Número de Erros de Planejamento	Número de Erros Qualitativos
Correlação	-0.26	-0.09	-0.23	-0.27

A Tabela 11 apresenta a correlação entre a experiência dos participantes do estudo experimental na construção de planilhas e o número de erros cometidos por etapa do processo de construção de planilhas. A correlação mais forte identificada nesta análise relaciona a experiência com o número de erros introduzidos em fórmulas: quanto maior a experiência do participante, menor o número de erros introduzidos nesta fase. Aparentemente, o número de erros cometidos das demais fases é quase independente da experiência, apresentando baixa correlação com a mesma.

Tabela 11 – Correlação entre Experiência e Número de Erros por Etapa

	Número Total de Erros	Número de Erros de Estruturação	Número de Erros de Associação	Número de Erros de Fórmulas
Correlação	-0.26	-0.16	-0.01	-0.40

A Tabela 12 apresenta a correlação entre a experiência dos participantes no desenvolvimento de sistemas e o número de erros cometidos por categoria. O número de anos de trabalho com desenvolvimento de sistemas foi utilizado para representar a experiência de cada participante, retirando-se da amostra aqueles que não tinham experiência com desenvolvimento de sistemas.

Como na análise anterior, é possível observar que as correlações são todas negativas, indicando que quanto maior a experiência do participante, menor é o número de erros cometido por ele. Diferente da análise baseada na experiência com a construção de planilhas, o número de erros qualitativos foi quase independente da experiência com o desenvolvimento de sistemas. Este fato pode ser explicado, uma vez que a construção de planilhas envolve o conhecimento de aspectos que não são comuns ao desenvolvimento de sistemas, como a construção de fórmulas e o uso de tabelas do Excel.

Tabela 12 – Correlação entre Experiência e Número de Erros por Categoria

	Número Total de Erros	Número de Lapsos	Número de Erros de Planejamento	Número de Erros Qualitativos
Correlação	-0.29	-0.30	-0.39	-0.02

A Tabela 13 apresenta a correlação entre a experiência dos participantes no desenvolvimento de sistemas e o número de erros cometidos por etapa do processo de construção de planilhas. Como na Tabela 11, o número de erros de estruturação foi independente da experiência, ficando os demais tipos de erro com correlação muito similar com a experiência.

Tabela 13 – Correlação entre Experiência e Número de Erros por Etapa

	Número Total de Erros	Número de Erros de Estruturação	Número de Erros de Associação	Número de Erros de Fórmulas
Correlação	-0.29	-0.06	-0.35	-0.33

4.4.4 **Testes de Inferência Estatística**

As análises apresentadas até aqui visaram caracterizar os erros cometidos pelos participantes em relação à sua categoria e à etapa do processo de construção de planilhas em que foram cometidos, assim como relacionar a frequência dos erros com a experiência dos participantes. No entanto, não se realizou qualquer análise estatística no sentido de identificar se existe um número significativo de erros nas planilhas construídas manualmente, de modo a diferenciá-las da planilha gerada pelo uso da proposta apresentada no Capítulo 4. Este é o objetivo desta seção.

Visando aplicar testes estatísticos com maior poder de conclusão, começamos por analisar a distribuição de frequência dos dados coletados durante a execução do estudo experimental. Dados que sigam a distribuição normal poderão ser submetidos a análises paramétricas (no caso, um *teste T*), enquanto os demais dados terão que ser analisados por testes não paramétricos (no caso, um teste de *Wilcoxon-Mann-Whitney*), que embora tenha menor poder de conclusão, pode ser aplicado em dados que não sigam a distribuição normal.

A Tabela 14 apresenta o *p-value* do teste de *Shapiro-Wilk* para verificação de normalidade nos dados quando aplicado às diversas séries de dados disponíveis: o número total de erros por participante, o número total de erros por participante e etapa e o número total de erros por participante e categoria. Os testes foram realizados no sistema de análise estatística R, versão 3.0.1 (Venables *et al., 2013*). Considerando que *p-values* pequenos descartam a possibilidade de que os dados associados sigam a distribuição normal, podemos afirmar com 95% de certeza que as séries com o número total de erros, os erros cometidos em fórmulas e os erros qualitativos seguem uma distribuição normal e o restante das séries não segue uma distribuição normal.

Tabela 14 – Resultados dos Testes de Análise de Normalidade

	p-Value
Número total de erros	0.570
Número de erros de estruturação	< 0.001
Número de erros de associação	0.020
Número de erros de fórmula	0.127
Número de lapsos	0.009
Número de erros de planejamento	0.041
Número de erros qualitativos	0.054

Com base na Tabela 14, aplicaremos um teste não paramétrico de *Wilcoxon-Mann-Whitney* para todas as séries de dados coletadas no estudo, além de um teste paramétrico (*teste T*) para as séries com o total de erros, erros em fórmulas e erros qualitativos. Nestes testes, compararemos as séries de dados com uma média fixa e igual a zero, visando determinar se os dados nestas séries são significativamente diferentes de zero. A Tabela 15 apresenta os *p-values* destes testes de inferência.

Tabela 15 – Resultados dos Testes de Inferência Estatística

	Wilcoxon	Teste T
Número total de erros	< 0.001	< 0.001
Número de erros de estruturação	0.059	-
Número de erros de associação	< 0.001	-
Número de erros de fórmula	0.003	< 0.001
Número de lapsos	0.006	-
Número de erros de planejamento	0.002	-
Número de erros qualitativos	< 0.001	< 0.001

Considerando que um *p-value* pequeno indica que a hipótese nula deve ser rejeitada, todas as séries avaliadas foram consideradas significativamente diferentes de zero com 95% de certeza. De fato, todas as séries com exceção do número de erros de estruturação de planilhas podem ser consideradas significativamente diferentes de zero com 99% de certeza.

Assim, rejeitamos a hipótese nula e identificamos evidências de que a hipótese alternativa é verdadeira: o número de erros introduzidos na construção manual de planilhas é diferente de zero e um método que permita a construção automática, como o apresentado no Capítulo 4, pode ser útil.

4.5 Considerações Finais

Com os resultados coletados nas execuções do estudo experimental, podemos responder à questão de pesquisa proposta.

> **RQ1:** A construção manual de planilhas eletrônicas implica na introdução de um número de erros significativamente maior do que zero?

A partir de testes estatísticos, temos indícios para afirmar que a construção manual de planilhas eletrônicas implica na introdução de erros significativamente maior do que zero.

A maior parte destes erros se deve a não utilização de tabelas nas planilhas. Isso porque quando a tabela não é utilizada para agrupamento de dados, com a inserção de um novo elemento, ou seja, uma nova linha, uma fórmula de uma coluna não é replicada para esta nova linha. Além disso, os totais das colunas não são automaticamente atualizados. Outro fato que podemos notar é que as validações de dados para as referências às tabelas também não foram feitos pela maioria dos participantes. Além do mais, pudemos observar que a maioria dos erros encontrados não tem uma relação muito forte com a experiência de usuários em planilhas, ou seja, até mesmo usuários mais experientes cometem erros.

Capítulo V – Conclusão

Este trabalho apresentou uma abordagem para geração automática de planilhas eletrônicas a partir de modelos conceituais. Esta abordagem visa a construção de planilhas mais resistentes à introdução de alguns tipos de erro por usuários finais do que planilhas geradas manualmente.

A abordagem proposta utiliza modelos de classes da UML para descrever os conceitos do domínio do problema que se pretende resolver com a planilha e expressões OCL para descrever as operações relacionadas com estes conceitos. Uma série de transformações é aplicada sobre estes modelos para produzir a planilha desejada pelos usuários.

Para fins de validação da solução proposta, foi realizado um estudo experimental que apontou indícios de que o número de erros em planilhas construídas manualmente é significativamente maior do que zero, ou seja, que as planilhas geradas automaticamente tendem a apresentar significativamente menos erros do que as planilhas construídas manualmente.

5.1 Contribuições

Dentre as contribuições deste trabalho é possível destacar:

- Criação de uma abordagem de geração automática de planilhas a partir de modelos conceituais utilizados no desenvolvimento de software;
- Definição de um metamodelo de planilhas, que descreve os principais conceitos utilizados em planilhas eletrônicas e seus relacionamentos;
- Criação de regras de transformação dos elementos de um diagrama de classes da UML em elementos da planilha, em linha com os metamodelos do diagrama de classes, mantido pelo OMG, e de planilhas, desenvolvido no contexto desta Dissertação;

- Criação de regras de transformação de expressões OCL em fórmulas da planilha, em linha com os metamodelos da OCL, mantido pelo OMG, e de planilhas;
- Aplicação de transformações em regras de negócio;
- Realização de um estudo experimental que ajudou a identificar erros comumente cometidos por usuários de planilhas. O estudo envolveu 16 participantes, entre alunos de graduação, pós-graduação e profissionais da indústria;
- Desenvolvimento de uma ferramenta que implementa a abordagem de geração automática de planilhas proposta.

5.2 Limitações

Podemos destacar algumas limitações da abordagem proposta com relação a três aspectos: metamodelo UML, metamodelo OCL e transformações.

Este trabalho não engloba todos os aspectos definidos no metamodelo UML referentes à construção de diagramas de classes. Construções como generalização e especialização, associações binárias com multiplicidade N:N, associações que não sejam binárias, atributos multivalorados, agregação/composição e interfaces não foram incluídas no escopo deste trabalho e não são consideradas nas regras de transformação que convertem um diagrama de classes em elementos do metamodelo de planilhas.

Este trabalho também não engloba alguns tipos de restrições da OCL, como invariantes, derivação de atributos e associações, valor inicial de atributos, pré e pós-condições. Além disso, nem todas as operações disponíveis na especificação do OCL são tratadas, dentre elas operações sobre inteiros (abs, max, min, mod e div), reais (floor e round), strings (concat, toLower, toUpper e subString), operações relacionais (xor, not e implies) e operações aplicáveis a coleções (isEmpty(), notEmpty(), exists() e forAll()).

Além disso, as transformações não tratam expressões OCL que contenham expressões de navegação dentro de operações de seleção (*select*) ou coleta (*collect*), assim como não tratam navegações com multiplicidade ‘1’.

Um problema que podemos encontrar na proposta é o fato de que ela está direcionada a princípio a pessoas com conhecimento em diagramas de classes e expressões OCL, e não aos usuários finais.

5.3 Trabalhos Futuros

Como trabalhos futuros pretende-se cobrir todas as transformações que não foram consideradas no escopo desta proposta (seção 5.2), bem como implementá-las na ferramenta desenvolvida. Além disso, pretendemos avaliar a utilização de outros tipos de modelos conceituais, como os modelos entidade relacionamento (MER) ou ontologias. Extensões deste trabalho também podem contemplar a geração de planilhas para diferentes softwares, uma vez que a implementação atual foi realizada apenas para o Microsoft Excel. Ainda podemos considerar a geração de planilhas com diferentes formatos visuais, utilizando os diversos recursos de interação com o usuário para adequar o formato da planilha ao que mais lhe convém.

O experimento ratificou a existência e relevância dos erros cometidos pelos usuários na construção da planilha de forma manual. Um estudo relevante poderia ser avaliar a construção dos modelos que são utilizados na proposta de solução.

Podemos ainda pensar em utilizar os artefatos utilizados para gerar as planilhas, como diagramas de classes e expressões OCL para geração de casos de teste de planilhas.

Referências Bibliográficas

ABREU, R.; RIBOIRA, A.; WOTAWA, F. (2012) "Constraint-based Debugging of Spreadsheets", XV Ibero-American Conference on Software Engineering (CibSE12), Buenos Aires, Argentina, April.

BASILI, V.; CALDIERA, G.; ROMBACH, H.D. (1994) "Goal Question Metric Approach". Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc.

BOOKBINDER, D. J. (1989) "The Lotus guide to 1-2-3: Release 3". Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

BELO, O.; CUNHA, J.; FERNANDES, J. P.; MENDES, J.; PEREIRA, R.; SARAIVA, S. (2013) "QuerySheet: A Bidirectional Query Environment for Model-Driven Spreadsheets", In the Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '13), September 15-19. San Jose, CA, USA.

BERRY, D. M.; KAMSTIES, E. (2004) "Ambiguity in Requirements Specification", In Leite, J. C. S. P. and Doorn, J. H. (eds), Perspectives On Software Requirements, 1 ed., chapter 2, Kluwer Academic Publishers, 2004.

BOOCH, G. (1994) "Object Oriented Analysis and Design with Applications". 2 ed. California, Addison-Wesley.

BOOCH, G. (1999) "UML in action", Communications of the ACM, v. 42, pp. 26-28.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. (2005) "The Unified Modeling Language User Guide". 2 ed. Massachusetts, Addison-Wesley.

BURNETT, M.; SHERETOV, A.; REN, B.; ROTHERMEL, G. (2002) "Testing homogeneous spreadsheet grids with the "what you see is what you test"

methodology", Software Engineering, IEEE Transactions on , vol.28, no.6, pp.576,594, Jun.

CAMPBELL, M. V. (1985) "Using Excel". Que Corp., Indianapolis, IN, USA.

CLERMONT, M. (2004) "A Scalable Approach to Spreadsheet Visualization". PhD thesis, Universitaet Klagenfurt.

CLERMONT, M.; HANIN, C.; MITTERMEIR, R. T. (2008) "A Spreadsheet Auditing Tool Evaluated in an Industrial Context", In ACM Computing Research Repository (CoRR).

CORREA, A. L., (2006) "Reestruturando Especificações de Restrições de Modelos Elaboradas em OCL", Tese de Doutorado, Maio.

CROLL, G. J. (2007) "The importance and criticality of spreadsheets in the city of London". CoRR, abs/0709.4063.

CROLL, G. J. (2009) "Spreadsheets and the financial collapse". CoRR, abs/0908.4420.

CUNHA, J.; SARAIVA, J.; VISSER, J. (2009a) "Discovery-based edit assistance for spreadsheets", Visual Languages and Human-Centric Computing (VL/HCC 2009), IEEE Symposium on , vol., no., pp.233,237, 20-24 September.

CUNHA, J., SARAIVA, J., VISSER, J. (2009b) "From spreadsheets to relational databases and back", In Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, pp. 179–188. ACM, New York, NY, USA .

CUNHA, J.; ERWIG, M.; SARAIVA, S. (2010) "Automatically Inferring ClassSheet Models from Spreadsheets", In proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2010), Madrid, Spain, IEEE Computer Society, September.

CUNHA, J., (2011) "Model-based Spreadsheet Engineering", *PhD thesis*, March.

CUNHA, J.; VISSER, J.; ALVES,T.;SARAIVA, S. (2011a) "Type-safe Evolution of Spreadsheets", In proceedings of the Fundamental Approaches to Software Engineering - FASE 2011, Saarbrücken, Germany, volume 6603 of LNCS, pages 186-201, Springer. March.

CUNHA, J.; BECKWITH, L.;FERNANDES, J. P.;SARAIVA, S. (2011b) "End Users Productivity in Model-based Spreadsheets: An Empirical Study", In

proceedings of the Third International Symposium on End-User Development - IS-EUD 2011, Torre Canne (Brindisi), Italy, June.

CUNHA, J.; BECKWITH, L.; FERNANDES, J. P.;SARAIVA, S. (2011c) "An Empirical Study on End-users Productivity Using Model-based Spreadsheets", In proceedings of the EuSpRIG'11, London, UK, July.

CUNHA, J.; MENDES, J., FERNANDES, J. P.; SARAIVA, S. (2011d) "Embedding and Evolution of Spreadsheet Models in Spreadsheet Systems", In proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2011), Pittsburgh, USA, pages 179-186, IEEE Computer Society, September.

CUNHA, J.; FERNANDES, J. P.; SARAIVA, S. (2012a) "From Relational ClassSheets to UML+OCL", 27th Annual ACM Symposium on Applied Computing (SAC 2012), track on Software Engineering, Trento, Italy, pages 1151-1158, March.

CUNHA, J.; FERNANDES, J. P.; MENDES, J.; SARAIVA, S.; PACHECO, H. (2012b) "Bidirectional Transformation of Model-Driven Spreadsheets", 5th International Conference on Model Transformation (ICMT 2012), Prague, Czech Republic, pages 105-120, May.

CUNHA, J.; FERNANDES, J. P.; MENDES, J.; SARAIVA, S. (2012c) "MDSheet: A Framework for Model-driven Spreadsheet Engineering", 34th Internactional Conference on Software Engineering (ICSE 2012), Zurich, Switzerland, pages 1395-1398, June.

CUNHA, J.; FERNANDES, J. P.; MENDES J.; SARAIVA, S. (2012d) "Towards an Evaluation of Bidirectional Model-driven Spreadsheets", User evaluation for Software Engineering Researchers (USER 2012), an ICSE 2012 Workshop, Zurich, Switzerland, pages 25-28, June 5.

CUNHA, J.; FERNANDES, J. P., MARTINS,P.; MENDES, J.; SARAIVA, S. (2012e) "SmellSheet Detective: A Tool for Detecting Bad Smells in Spreadsheets", In proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012) (tool demo), Innsbruck, Austria, pages 243–244, September/October 2012.

CUNHA J.; SARAIVA, S.; VISSER, J. (2012f) "Model-based Programming Environments for Spreadsheets", 16th Brazilian Symposium on Programming Languages (SBLP 2012), pages 117-133, Natal, Brazil, September 2012.

CUNHA, J.; FERNANDES, J. P.; PEIXOTO, C.; SARAIVA, S. (2012g) "A QualityModel for Spreadsheets", In the Proceedings of the 8th International Conference on the Quality of Information and Communications Technology, Quality in ICT Evolution Track, pages 231-236, Lisbon, Portugal, 3 to 6 September.

CUNHA, J., FERNANDES, J.P., MENDES, J.; SARAIVA, S. (2012h) "Extension and Implementation of ClassSheet Models", 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012), Innsbruck, Austria, pages 19–22, September/October.

CUNHA, J.; FERNANDES, J.P.; RIBEIRO, H.; SARAIVA, J. (2012i) "Towards a catalog of spreadsheet smells", In Proceedings of the 12th international conference on Computational Science and Its Applications (ICCSA'12). Vol. Part IV, 202-216. Springer-Verlag, Berlin, Heidelberg.

CUNHA, J.; FERNANDES, J.P.; MENDES, J.; SARAIVA, S. (2013a) "Complexity Metrics for Spreadsheet Models", In proceedings of the The 13th International Conference on Computational Science and Its Applications (ICCSA 2013), Ho Chi Minh City, Vietnam, June 24-27.

CUNHA, J.; FERNANDES, J. P.; MENDES, J.; PEREIRA, R.; SARAIVA, S. (2013b) "Querying Model-Driven Spreadsheets", In the Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '13), September 15-19. San Jose, CA, USA.

DAVIS, J. S. (1996) "Tools for spreadsheet auditing". International Journal of Human Computer Studies, 45(4):429–442.

ELMASRI, R.; NAVATHE, S. B. (2009) "Fundamentals of Database Systems". Addison/Wesley, Redwood City, California, 5rd edition.

ENGELS G.; ERWIG, M. (2005) "ClassSheets: Automatic generation of spreadsheet applications from object-oriented specifications", In ASE '05: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, pages 124-133, New York, NY, USA.

EuSpRIG. European Spreadsheet Risks Interest Group. http://www.eusprig.org/, 2011. pages 3, 5 and 133.

FOWLER, M. (1997) "Analysis Patterns: Reusable Object Models". Addison-Wesley, Reading, Massachusetts, 1997.

FRANCE, R.; RUMPE, B. (2007) "Model-driven Development of Complex Software: A Research Roadmap", In 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, 37-54, 2007.

GREENFIELD, J.; SHORT., K. (2004) "Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools". Wiley Publishing, Inc., Indianapolis, IN.

HERMANS, F.; PINZGER, M.; VAN DEURSEN, A. (2010) "Automatically extracting class diagrams from spreadsheets", In Proceedings of the European Conference on Object Oriented Programming (ECOOP), pages 52-75.

HERMANS, F.; PINZGER, M; VAN DEURSEN, A. (2011) "Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams", In Proceedings of the International Conference on Software Engineering (ICSE), pages 451-460.

HEUSER, C. A. (2009) "Projeto de Banco de Dados". Vol. 4, 6ª ed., Editora Bookman.

JACOBSON, I., (1992) "Object Oriented Software Engineering: a Use Case Driven Approach". 1 ed., Addison-Wesley.

KO, A. J.; ABRAHAM, R.; BECKWITH, L.; BLACKWELL, A.; BURNETT, M.; ERWIG, M.; SCAFFIDI, C.; LAWRANCE, J.; LIEBERMAN, H.; MYERS, B.; ROSSON, M. B.; ROTHERMEL, G.; SHAW, M.; WIEDENBECK, S. (2011) "The state of the art in end-user software engineering". Journal ACM Computing Surveys (CSUR), vol. 43, issue 3, article 21, 61 pages.

LEMIEUX, V. (2002) "Competitive Viability, Accountability and Record Keeping: A Theoretical and Empirical Exploration Using a Case Study of Jamaican Commercial Bank Failures". PhD thesis, University College London, 2002.

LEMIEUX, V. (2008) "Archiving: The overlooked spreadsheet risk". CoRR, abs/0803.3231.

MENDES, J. (2011) "ClassSheet-driven Spreadsheet Environments", In proceedings of the Graduate Consortium of the 2011 IEEE Symposium on Visual Languages

and Human-Centric Computing (VL/HCC 2011), Pittsburgh, USA, pages 235-236, IEEE Computer Society, September.

MENDES, J. (2012) "Coupled Evolution of Model-driven Spreadsheets", *Extended Abstrac*t for the Student Research Competition at 34th International Conference on Software Engineering (ICSE 2012), Zurich, Switzerland, pages 1616–1618, June.

MENDES, J. (2012) "Model-Driven Spreadsheets in a Multi-User Environment". In proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012) (Graduate Consortium), Innsbruck, Austria, pages 231–232, September/October.

MEYER, B. (1992) "Applying Design by Contract". IEEE Computer, v. 25, n. 10, pp. 40-51.

O'DONOVAN, T. M. (1984) "VisiCalc Made Simple". John Wiley & Sons, Inc., New York, NY, USA.

O'LEARY, L. (2008) "Microsoft Office Excel 2007 Introduction". McGraw-Hill, Inc., New York, NY, USA, 2008.

OMG, (1999) "Object Management Group - Unified Modeling Language (UML) 1.3 specification". In: http://www.omg.org/cgi-bin/doc?formal/00-03-01, Accessed in 08/2013.

OMG, (2002a) "Object Management Group - MOF 1.4 specification". In: http://www.omg.org/cgi-bin/doc?formal/2002-04-03, Accessed in 08/2013.

OMG, (2002b) "QVT-Merge Group 1.8 Revised submission for MOF 2.0 Query/Views/Transformations". RFP (ad/2002-04-10). Technical report.

OMG, (2003a) "Object Management Group - UML 2.0 OCL Specification". In: http://www.omg.org/cgi-bin/doc?ptc/2003-10-14, Accessed in 08/2013.

OMG, (2003b) "Object Management Group – OMG/MDA Guide. Version 1.0.1". In: http://www.omg.org, 2003, Accessed in 08/2013.

PANKO R. (2000) "Spreadsheet errors: what we know. what we think we can do". Spreadsheet Risk Symposium, July.

PANKO, R.R.; HALVERSON JR., R. P. (2001) "An Experiment in Collaborative Spreadsheet Development". Journal of the Association for Information Systems 2(4), July.

PANKO, R. R.; AURIGEMMA, S. (2010) "Revising the Panko-Halverson taxonomy of spreadsheet errors". Decision Support Systems, 49(2):235–244.

PARNAS, D. L. (1972) "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, v. 15, n. 12, pp. 1053-1058.

PFLEEGER, S. H., (2001) "Software Engineering: Theory and Practice". 2 ed. New Jersey, Prentice Hall.

POWELL, S.; BAKER, K. (2003) "The Art of Modeling with Spreadsheets". John Wiley & Sons, Inc. New York, NY, USA.

RAJALINGHAM, K.; CHADWICK, D.; KNIGHT, B; EDWARDS, D. (2000) "Quality Control in Spreadsheets", In Proceedings of the 33rd Hawaii International Conference on System Sciences, 10 pages.

RILEY, J. A. (2009) "Introduction to OpenOffice.org". Prentice Hall Press, Upper Saddle River, NJ, USA.

RONEN, B.; PALLEY, M.; LUCAS JR., HENRY. (1989) "Spreadsheet analysis and design", Communications of the ACM, 32(1):84–93, January 1989.

ROSS, D. T., GOODENOUGH, J. B., IRVINE, C. A., (1975) "Software Engineering: Process, Principles and Goals", IEEE Computer, v. 8, n. 5, pp. 17-27.

ROTHERMEL, G.; LI, L.; DUPUIS, C.; BURNETT, M. (1998) "What you see is what you test: a methodology for testing form-based visual programs", Proceedings of the 1998 International Conference on Software Engineering, 1998, vol., no., pp.198,207, 19-25 April.

ROTHERMEL, K. J.; COOK, C. R.; BURNETT, M. M.; SCHONFELD, J.; GREEN,T. R. G.; ROTHERMEL, G. (2000) "WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation". In Proceedings of the 22nd international conference on Software engineering (ICSE '00). ACM, New York, NY, USA, 230-239.

ROTHERMEL, G.; BURNETT, M.; LI, L.; DUPUIS, C.; SHERETOV, A (2001) "A methodology for testing spreadsheets", ACM Transactions Software Engineering Methodology 10, 1, January, 110-147.

RUMBAUGH, J. (1990) “Object Oriented Modeling and Design”. 1 ed., Prentice Hall.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. (2004) “The Unified Modeling Language Reference Manual”. 2 ed. Massachusetts, Addison-Wesley.

SCAFFIDI, C.; SHAW, M.; MYERS. B. A. (2005) “Estimating the numbers of end users and end user programmers”, In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 207-214.

SOLEY, R.; FRANKEL, D.; MUKERJI, J.; CASTAIN, E. (2001) “Model Driven Architecture - The Architecture of Choice For a Changing World”. OMG Technical report.

SZTIPANOVITS, J.; KARSAI, G. “Model-integrated computing” (1997). Computer, 30(4):110–111.

VENABLES, W. N.; SMITH, D. M.; R CORE TEAM (2013) “An Introduction to R”, Version 3.0.1, Maio, http://cran.r-project.org/manuals.html <acessado em 21/08/2013>.

WARMER, J.; KLEPPE, A. (2003) “The Object Constraint Language. Getting Your Models Ready for MDA”. 2 ed. Reading, Mass, Addison Wesley.

WOHLIN, C., RUNESON, P., HOST, M. et al. (2000) “Experimentation in Software Engineering. An Introduction”, Massachusetts, Kluwer Academic Publishers Group.

Anexo A – Estudo Experimental
Questionário de Caracterização

Nome:_____

Determine seu nível de formação.

[ ] Graduando [ ] Graduado
[ ] Mestrando [ ] Mestre
[ ] Doutorando [ ] Doutor

Determine sua área de atuação (marque mais de uma alternativa, se julgar necessário).

[ ] Não atuo profissionalmente
[ ] Análise e Desenvolvimento de Sistemas (____ anos)
[ ] Finanças (____ anos)
[ ] Outros: Especifique: _____ (____ anos)

Determine a sua experiência em construção de planilhas.

[ ] Eu nunca construí uma planilha
[ ] Eu já construí planilhas simples para uso próprio
[ ] Eu já construí planilhas de média complexidade
[ ] Eu já construí planilhas de alta complexidade
[ ] Eu trabalho com construção de planilhas de alta complexidade

Anexo B – Estudo Experimental
Descrição Textual do Minimundo

Interessado em controlar as entradas e saídas dos seus produtos e aumentar a lucratividade do seu negócio, um comerciante dono de uma rede de minimercados, chamada Minimundo, decide montar uma planilha para fazer o controle de estoque dos produtos.

O comerciante deseja armazenar o nome, o código e a localização de cada minimercado (loja). Para cada loja, ele necessita totalizar as compras e vendas realizadas (em reais) e armazenar o lucro que obteve através do cálculo da diferença entre o total de vendas e o total de compras durante todo o período.

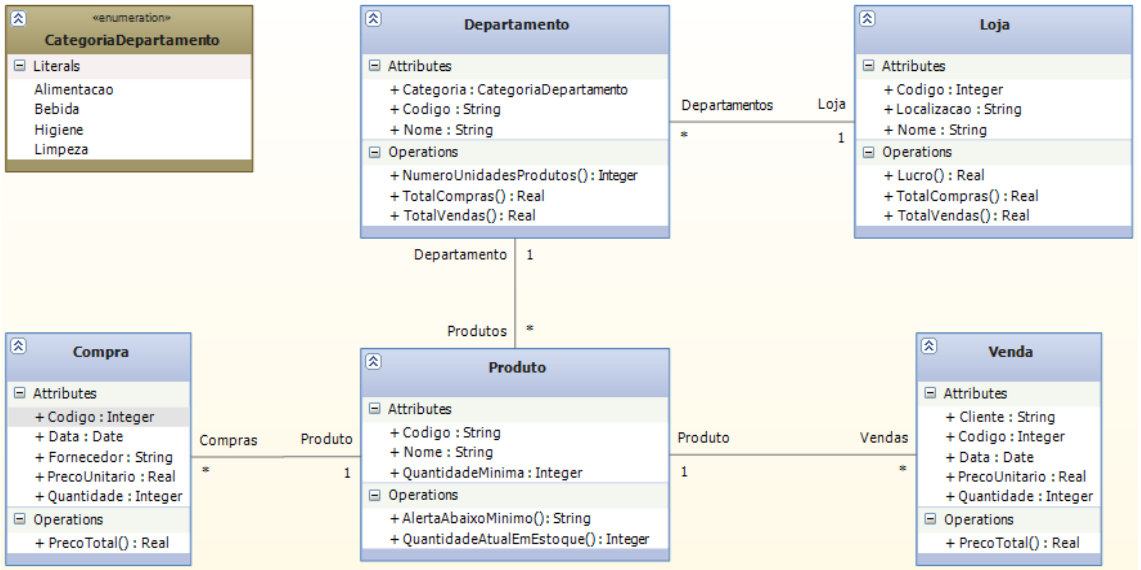
As lojas são divididas em departamentos. Cada departamento tem seus produtos, que não podem pertencer a dois departamentos distintos. O comerciante deseja armazenar o código, o nome e a categoria de cada departamento. Para que não haja erro na inclusão da categoria do departamento, ele deseja manter uma lista com as categorias existentes, que são: alimentação, bebida, higiene e limpeza. Além disso, ele deseja calcular a quantidade de produtos em cada departamento (número de unidades), o total gasto com a compra dos produtos (em reais) e o total de vendas (em reais).

Os produtos são comprados de um determinado fornecedor e revendidos nas lojas. Para cada produto, ele deseja armazenar o código, nome e quantidade mínima que deve ser mantida em estoque. Para cada produto, ele deseja manter a quantidade atual em estoque, que é calculada através da diferença entre a quantidade de compras e a quantidade de vendas do produto. Além disso, um campo de alerta deve ser criado para avisar quando um produto está com o estoque abaixo do mínimo permitido. Para que isso seja feito é necessário criar uma verificação entre a quantidade de estoque atual e a quantidade mínima de cada produto.

O comerciante deseja armazenar dados sobre as compras dos produtos, que são: identificador da transação, nome do fornecedor, data da compra, preço unitário (em reais) e quantidade comprada (número de unidades). Deseja também manter

informações sobre as vendas dos produtos, que são: identificador da transação, nome do cliente, data da venda, preço unitário (em reais) e quantidade (número de unidades). Para simplificar, assumiremos que cada compra ou venda conterá apenas um produto. Tanto para as compras quanto para as vendas, o comerciante precisa calcular o preço total de cada transação (em reais), a partir da multiplicação do preço unitário pela quantidade do produto comprado ou vendido.

Anexo C – Estudo Experimental
Diagrama de Classes do Minimundo



Anexo D – Estudo Experimental
Links dos vídeos de funcionalidades do Excel

Duração total dos vídeos: aprox. 36min

Excel 2007: Organize data using an Excel Table (4:47)
http://www.youtube.com/watch?v=RVZ7VKIN-vk

Excel 2007: Use simple formulas to do the math (7:33)
http://www.youtube.com/watch?v=XvkpGPjEgQI

Excel 2007: Create a Drop Down Menu (Data Validation Selection List) (3:44)
http://www.youtube.com/watch?v=arMjwrQg_rw

Excel 2010: INDEX and MATCH Functions (9:12)
http://www.youtube.com/watch?v=GjDSclhCv0M

Excel 2007: SUMIFS & COUNTIFS (7:17)
http://www.youtube.com/watch?v=_cWg850PMys

Excel 2007: Nesting IF and AND functions (3:58)
http://www.youtube.com/watch?v=YT1abztXZxc