UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO
MESTRADO PROFISSIONAL EM SISTEMAS E COMPUTAÇÃO

CARLOS EUGÊNIO PALMA DA PURIFICAÇÃO

UMA PLATAFORMA DE DESENVOLVIMENTO DE SISTEMAS EMPRESARIAIS BASEADA EM MODELOS E REGRAS DE NEGÓCIO

Salvador
2016

CARLOS EUGÊNIO PALMA DA PURIFICAÇÃO

UMA PLATAFORMA DE DESENVOLVIMENTO DE SISTEMAS EMPRESARIAIS BASEADA EM MODELOS E REGRAS DE NEGÓCIO

Dissertação apresentada ao Curso de Mestrado Profissional em Sistemas e Computação, da Universidade Salvador - UNIFACS, Laureate Internacional Universities, como requisito parcial para obtenção do grau de Mestre.

Orientador: Prof. Dr. Paulo Caetano da Silva.

Salvador
2016

Ficha Catalográfica elaborada pelo Sistema de Bibliotecas da Universidade Salvador – UNIFACS, Laureate Internacional Universities.

Purificação, Carlos Eugênio Palma da

Uma Plataforma de Desenvolvimento de Sistemas Empresariais baseada em modelos e regras de negócio. / Carlos Eugênio Palma da Purificação. – Salvador, 2016.

150 f. : il

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação, Universidade Salvador — UNIFACS, Laureate International Universities como requisito parcial para a obtenção do grau de Mestre.

Orientador: Prof. Dr. Paulo Caetano da Silva.

1. Desenvolvimento de software. 2. Interação com o usuário. I. Silva, Paulo Caetano da, orient. II. Título.

CDD: 005.1

CARLOS EUGÊNIO PALMA DA PURIFICAÇÃO

UMA PLATAFORMA DE DESENVOLVIMENTO DE SISTEMAS EMPRESARIAIS BASEADA EM MODELOS E REGRAS DE NEGÓCIO

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, Universidade Salvador - UNIFACS, Laureate International Universities, pela seguinte banca examinadora:

Paulo Caetano da Silva – Orientador _____
Ph.D. in Computer Science, Universidade Federal de Pernambuco - UFPE
UNIFACS Universidade Salvador, Laureate International Universities

Rita Suzana Pitangueira Maciel _____
Doutora em Ciências da Computação pela Universidade Federal de Pernambuco
Universidade Federal da Bahia (UFBA)

Sérgio Martins Fernandes_____
Doutor em Ciências da Computação pela Universidade de São Paulo - USP
UNIFACS Universidade Salvador, Laureate International Universities

Salvador, 30 de setembro de 2016.

Dedico este trabalho à Deus, minha família e todos que me ofereceram de alguma maneira apoio nesta jornada.

AGRADECIMENTOS

À Deus e à minha família que permitiram que essa jornada fosse trilhada. À minha esposa Dejaci e minha filha Maria Sofia que fizeram parte deste empreendimento com o suporte familiar fundamental para esta realização. Meus pais e irmãos pelo incentivo e apoio ao longo do desenvolvimento deste trabalho e todos os outros familiares e amigos.

Ao Prof. Paulo Caetano pelo apoio e confiança, desde o início, sem o qual não seria possível o término deste trabalho. As inúmeras revisões, conselhos e trabalhos criados em conjunto, forneceram insumos imprescindíveis para a conclusão deste trabalho.

Agradeço também ao SERPRO pelo suporte financeiro parcial no início deste trabalho.

RESUMO

Organizações empresariais necessitam de sistemas de informação alinhados com suas regras e processos de negócio. Esses sistemas são necessários na gestão e manutenção dos dados e informações estratégicas para a consecução de seus objetivos e gerar oportunidades competitivas. Entretanto, nesse tipo de organização, os recursos financeiros são escassos na fase inicial de suas operações, o que torna crítico que seus sistemas de informação possam ser construídos de forma ágil e menos custosa. Geralmente, nos sistemas de informação dessas empresas, as regras e processos de negócio estão descritas em linguagem natural em documentos dissociados do sistema, e posteriormente implementadas diretamente no código fonte do software. Esta prática pode levar a um elevado custo de manutenção e dificuldade de adaptação dos sistemas às necessidades de evolução das regras de negócio, pois geralmente envolve algum trabalho de codificação por equipes especializadas na área de tecnologia da informação. Além disto, a tradução das regras de negócio por estes profissionais, pode levar à perda da semântica original da regra, resultando em implementação errônea da intenção original do analista de negócio. Analogamente, o modelo de interação com o usuário geralmente é implementado baseado em técnicas de programação, ficando a cargo dos desenvolvedores a criação e manutenção das diversas telas do sistema, captura de eventos, controle de navegação e sua interação com as regras de negócio das organizações. Este trabalho propõe uma plataforma e um processo de desenvolvimento de software baseado em modelos para a incorporação e manutenção de regras de negócio em sistemas de informação de forma flexível, utilizando a linguagem do próprio negócio, porém baseado em um rigor formal, em oposição a utilização de linguagens natural e sua tradução para linguagens de programação, para permitir que as regras de negócio dos sistemas sejam rapidamente adaptadas e incorporadas aos sistemas pelos interlocutores do negócio. O modelo de interação com o usuário também é definido formalmente, de forma a abranger os conceitos desta interação e permitir sua validação frente às regras de negócio e posterior transformação em código executável. A plataforma envolve ferramental e componentes, aliados a um processo de desenvolvimento baseado em modelos, para auxiliar na definição do modelo conceitual do sistema e geração da aplicação final. Linguagens específicas de domínio, baseadas em padrões formais, foram criadas para permitir a definição do modelo conceitual do sistema sob os pontos de vista estrutural, de negócio e interação com o usuário de forma integrada. Um estudo de caso exemplifica a utilização do modelo de desenvolvimento proposto, em que uma aplicação de exemplo é modelada sob seus aspectos estrutural, de interação com o usuário e regras de negócio de validação de entrada de dados. Subsequentemente é mostrado como estes modelos são submetidos a um processo de transformação para gerar a aplicação final. Os exemplos apresentados no estudo de caso mostram como é possível realizar a validação de entrada de dados do usuário baseado em regras de negócio utilizando a abordagem proposta.

Palavras-chaves: Desenvolvimento Dirigido a Modelos. Regras de Negócio. SBVR. Interação com o usuário. IFML. Linguagens Específicas de Domínio.

ABSTRACT

Organizations need to align information systems with its rules and business processes. These systems are needed in strategic data management and maintenance in order to achieve the business goals and create competitive opportunities. However, in this type of organization, financial resources are scarce in the initial phase of its operations, which makes it critical that their information systems can be built faster and less costly. Generally, in these company's information systems, the business rules and processes are described in natural language documents disassociated from the system and later implemented directly in the software source code. This practice can lead to high maintenance costs and difficulty in adapting systems to the needs of changing business rules, because it usually involves some source-code modification by information technology specialized teams. In addition, the translation of business rules by these professionals, can lead to loss of the original rule semantics, resulting in erroneous implementation of the original business analyst intention. Similarly, user interaction model is usually implemented based on programming techniques, leaving it to developers to create and maintain the various system screens, capture events, navigation control and its interaction with organization business rules. This paper proposes a platform and a model-driven software development process for incorporating and maintaining business rules in a flexible manner, using the business language itself, but based on a formalism, as opposed to the use of a natural language and translating the rules to programming languages constructs, to allow the systems business rules to be quickly adapted and incorporated into the systems by business partners. The user interaction model is also formally defined to encompass the concepts of this interaction, and to allow its validation against business rules and its subsequent transformation into executable code. The proposed platform involves tooling and components, that along with a model-driven development process assists the user on defining the system conceptual model and generating the final application. Domain specific languages based on formal standards were designed to allow the definition of the system conceptual model covering view, structural, business and user interaction aspects in an integrated manner. A use case study illustrates the use of the proposed development model, wherein a sample application is modeled in its user interaction, structure and data input validation rules aspects. Subsequently it is shown how these models are subjected to a transformation process to generate the final application. The examples presented in the use case study show how it is possible validation of user input based on business rules using the proposed approach.

Keywords: Model Driven Development. Business Rules. SBVR. User Interaction. IFML. Domain Specific Languages.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ATL	*ATLAS Transformation Language*
ANTLR	*Another Tool for Language Recognition*
CIM	*Computation Independent Model*
CL	*Controlled Language*
CWM	*Common Warehouse Metamodel*
DSL	*Domain Specific Language*
EIM	*Engen Intermediate Model*
EMF	*Eclipse Modeling Framework*
IDE	*Integrated Development Environment*
IFML	*Interaction Flow Modeling Language*
M2C	*Model-to-Code*
M2M	*Model-to-Model*
M2T	*Model-to-Text*
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MDE	*Model Driven Engineering*
MDSD	*Model Driven Software Development*
MOF	*MetaObject Facility*
MOFM2TMOF	*Model to Text Transformation Language*
MVC	*Model View Controller*
OMG	*Object Management Group*
ORM	*Object-Role Modeling*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
QVT	*Query/View/Transformation*
SBVR	*Semantics of Business Vocabulary and Business Rules*
SBVR-SE	*Semantics of Business Vocabulary and Business Rules Structured English*
SGRM	*SBVR Registry Module*
SI	*Sistemas de Informação*

SQL	*Structured Query Language*
UML	*Unified Modeling Language*
WebE	*Web Engineering*
XMI	*XML Metada Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A informação é um recurso estratégico em empresas, pois pode ser utilizada para medir seu grau de sucesso e apresentar oportunidades de diversificação de produtos e serviços, aumentando sua competitividade. Sistemas de Informação Empresariais desempenham um importante papel nessas organizações, pois eles implementam os processos do negócio e realizam a gestão da informação (LEVI ; POWELL, 2005).

Os sistemas de informação podem contribuir para atingir os objetivos empresariais e necessitam evoluir alinhados com as necessidades do negócio (WAN-KADIR; LOUCOPOULOS, 2004). Essa evolução, geralmente requer um constante esforço na manutenção e evolução desses sistemas, exigindo o empenho de uma grande quantidade de recursos da organização, notadamente, dos especialistas do negócio que detêm o conhecimento das necessidades a serem automatizadas e mantidas pelos sistemas de informação, e dos analistas de sistemas que traduzem as regras descritas, geralmente de maneira informal por meio de linguagem natural, para alguma linguagem de programação.

As necessidades e restrições de um sistema de informação são comumente expressas por meio do que se costuma definir genericamente, na área de Engenharia de Software, como Requisitos de Software (ABRAN et al. 2001). Ao se considerar a criação e evolução de Sistemas de Informação Empresarias é importante ressaltar a necessidade do alinhamento entre os objetivos empresariais e os requisitos definidos para os sistemas. Estes geralmente são expressos por meio de *Regras de Negócio*, que podem ser definidas como requisitos que restringem e controlam o comportamento de um processo de negócio ou definem alguma necessidade em um sistema de informação (ROOVER; VANTHIENEN, 2011).

Pesquisadores e profissionais da área de engenharia de software reconhecem a importância do tratamento de regras de negócio explicitamente durante o processo de desenvolvimento de sistemas de informação para garantir maior agilidade quando mudanças são necessárias (BAJEC; KRISPER, 2005a; JESUS, 2013; MARINOS; KRAUSE, 2009). A agilidade da adaptação das regras de negócio em sistemas empresariais é um elemento-chave para as organizações, pois estas estão sob constante pressão para se adequar às mudanças no ambiente e, portanto, esta capacidade, a adaptabilidade, deveria ser considerada como um requisito importante durante todo processo de desenvolvimento dos sistemas de informação. A falta de tratamento explícito de regras de negócio durante o processo de desenvolvimento, visando propiciar a rápida incorporação de mudanças dessas regras nos sistemas pode trazer diversos problemas, tais como: incapacidade de manter-se coerente com a realidade do

negócio, falta de documentação das regras, dificuldade de localização e manutenção das regras que estão diretamente definidas e espalhadas pelo código-fonte da aplicação sem um repositório adequado (BAJEC; KRISPER, 2005b).

Desta forma, pode-se destacar as seguintes motivações para considerar regras de negócio diretamente no processo de desenvolvimento de um sistema de informação empresarial, e tratá-las como artefatos que podem ser mudados rapidamente e refletidos no sistema:

a) a necessidade do constante alinhamento dos sistemas de informação com os processos de negócio de uma organização;
b) prover o sistema de informação com capacidade de rápida flexibilização de seu comportamento para permitir que se adapte a novas necessidades do negócio;
c) permitir que as regras de negócio possam ser visualizadas, acessadas, manipuladas e geridas diretamente por seus responsáveis (especialistas do negócio) em conjunto com desenvolvedores (especialistas em tecnologia da informação) facilitando sua aquisição, manipulação e gestão;
d) prover um repositório único de acesso às regras de negócio, visando facilitar sua gestão;

Entretanto, considerar as regras de negócio explicitamente nos sistemas de informação e projetá-los de modo a serem flexíveis para estas mudanças é um tópico complexo e esta funcionalidade está ausente na maioria das ferramentas, processos de desenvolvimento e sistemas existentes, tratando-as apenas sob a perspectiva de dados e funções (BAJEC; KRISPER, 2005b). Percebe-se esta deficiência nos sistemas existentes, impedindo mudanças céleres na condução do negócio das organizações para adequar-se a alterações em sua conjuntura, seja tanto por consequência de fatores ou eventos externos, quanto endógenos. Atingir estes objetivos resultaria no aumento da escalabilidade e adaptabilidade dos sistemas de informação, facilitando mudanças e manutenção dos mesmos (BAJEC; KRISPER, 2005b).

Um problema recorrente na expressão de regras de negócio em sistemas de informação é que as regras desenvolvidas por especialistas no negócio são expressas geralmente utilizando *Linguagem Natural*, enquanto que em sistemas, tais regras são expressas por algum modelo formal. Procurando estabelecer um modelo formal para regras de negócio escritas em linguagem natural, recentemente a *Object Management Group* – OMG editou uma nova versão (1.2) de seu padrão Semântico para Vocabulário e Regras de Negócio – SBVR (*Semantics of Business Vocabulary and Business Rules*) (OMG, 2013). O SBVR pode ser utilizado para representar regras de negócio utilizando linguagem natural, e permitindo assim

a interoperabilidade das regras entre organizações, usuários e ferramentas. Segundo a própria OMG, o padrão SBVR é aplicável a todos os domínios de vocabulários e regras de negócios, em todas as atividades e ramos. O suporte a abstrações de lógica de primeira ordem e seu foco específico em usuários do negócio são importantes características para a utilização do padrão para representar os conceitos do negócio (GAILLY, 2013).

Entretanto, não é possível utilizar somente o padrão SBVR para implementar regras de negócio diretamente nos sistemas de informação, pois o mesmo fornece simplesmente meios para descrever regras de negócio sem abordar sua implementação (NJONKO ; EL ABED, 2012; MARINOS ; KRAUSE, 2009). Um dos maiores problemas da adoção da abordagem de desenvolvimento baseado em regras de negócio é a ***falta de ferramental para a geração do código final*** baseado nas suas especificações (BAJEC ; KRISPER, 2005a, grifos nossos). Outro problema apontado pelos mesmos autores, é a tradução de regras de negócio escritas em linguagem natural para uma linguagem formal pelos analistas de sistemas. Como essa tradução nem sempre é fiel ao objetivo original da regra de negócio, geralmente introduzindo distorções entre a intenção original do analista de negócio e a tradução realizada pelos desenvolvedores, é desejável que exista ferramental que facilite e automatize essa tradução. Este trabalho procura prover respostas a este problema, propondo uma plataforma e ferramental que possibilitam a geração do código final baseado em regras de negócio.

Desta forma, para possibilitar que regras de negócio sejam utilizadas diretamente nos sistemas de informação, é necessário que exista um modelo formal, aliado a um ferramental e processos adequados, que permitam:

a) mapear e ***transformar*** os conceitos expressos nas regras de negócio (ex.: *obrigatoriedade, requisição de compra, data de requisição*) para artefatos de software;
b) ***validação*** sintática e semântica das várias regras desenvolvidas perante os artefatos de software;
c) identificação de quais momentos, durante a execução do sistema de informação, uma regra de negócio deve ser ***verificada***;

De acordo com Bajec e Krisper (2005a), o principal objetivo da pesquisa acerca de regras de negócio é achar caminhos e propor meios para apoiar a propagação automática de mudanças no negócio para seus sistemas de informação. Os autores enfatizam que o alinhamento entre a operação do negócio e de seus sistemas de informação é um dos problemas fundamentais em organizações, e propõe cinco áreas que necessitam ser sistematizadas e fundamentadas: escopo, aquisição, especificação, implementação e gestão.

Para esses autores, uma ideia promissora para resolver os problemas apresentados anteriormente seria o uso de uma a abordagem de desenvolvimento baseado em modelos. É importante salientar que, segundo o próprio documento de normatização (OMG, 2013), o padrão SBVR é completamente compatível com o desenvolvimento baseado em modelos. Outros estudos apontam a necessidade de ferramentas adequadas para a coleta e definição de regras de negócio descritas em SBVR de maneira prática e factível, de modo a adicionar valor real aos sistemas de informação nas organizações (FAYOUMI ; YANG, 2012).

Existem diversos tipos de regras de negócio. Um exemplo de regras de negócio em sistemas de informação empresariais são *regras de restrição, ou validação, de entrada de dados*. A entrada de dados é um tipo de regra que está presente em praticamente todos os sistemas empresariais, visto que lidam com uma parte fundamental destes sistemas: a validação da entrada de dados do usuário. A maioria dos dados manipulados por sistemas de informações empresarias é obtida por meio de formulários de entrada de dados nessas aplicações. Tais formulários podem ter, e geralmente possuem, algum tipo de validação de negócio que restringe o conjunto de valores válidos para cada campo do formulário. Consequentemente, este tipo de regra foi escolhido para ser tratada neste trabalho devido à sua aplicabilidade na maioria dos sistemas empresariais. Entretanto, a abordagem apresentada pode ser estendida para outros tipos de regras.

Não obstante, o modelo conceitual de um sistema de informação não depende somente do modelo de sua estrutura e regras de negócio. É necessário que seja especificado como o usuário irá interagir com a aplicação (AQUINO et al. 2011), ou seja, o modelo de interação com o usuário.

Diante do exposto, os problemas advindos da tradução manual de *regras de negócio de entrada de dados* para artefatos de software, sua *verificação* e *validação*, além da *integração* de tais regras com o *modelo de interação com o usuário*, e sua *transformação automática* em artefatos de software, perfazem os principais problemas que este trabalho procura abordar. Para isto, técnicas e ferramentas de desenvolvimento dirigido a modelos são apresentadas como forma de implementar as transformações automáticas e evitar os problemas de tradução manual de regras de negócio em artefatos de software.

Modelo de Interação com o Usuário

De forma a criar um modelo conceitual que permita expressar as necessidades de interação do usuário com a aplicação final, é necessário que, além da estrutura (modelo dos

dados que o sistema irá manipular) e comportamento do sistema, o modelo de interação com o usuário seja definido. (AQUINO et al. 2001).

O *Interaction Flow Modeling Language* – IFML é um padrão de modelagem de interação com o usuário, desenvolvido e lançado recentemente pela OMG (OMG, 2015). O padrão IFML foi concebido para padronizar como expressar o conteúdo, interação com usuário e comportamento do *front-end* de aplicações. Ele permite que analistas de software possam modelar e descrever as principais partes do *front-end* de uma aplicação independentemente de seu domínio (i.e. *Web*, *Desktop* ou Móvel).

O foco da especificação é a descrição da estrutura de uma aplicação, seus dados, comportamento e componentes de negócio, limitados ao que é perceptível, e que tem influência direta na experiência do usuário. Esse padrão define pontos de extensões que podem ser utilizados para definir peculiaridades para cada necessidade, enquanto provê as definições básicas do modelo de interação com o usuário. Em relação ao padrão *Model View Controller* – MVC (GAMMA et al. 1995), o foco do IFML é a parte de visualização – *View*. Ele delega a outras técnicas de modelagem o trabalho de especificar como modelar outras necessidades que não têm relação com a visualização e interação com o usuário.

Desenvolvimento Baseado em Modelos

O desenvolvimento de software tornou-se uma tarefa muito complexa para ser tratada com técnicas centradas em pura codificação (LUCRÉDIO, 2009). Diferentes estudos apontam a necessidade do uso de metodologias e processos que promovam o reúso, para permitir que esta atividade seja realizada com alto grau de eficiência, relação custo-benefício e confiabilidade (MARKIEWICZ; LUCENA; COWAN, 2000; RIEGGER et al. 2010; SCHWABE et al. 2001; LUCRÉDIO, 2009). Reúso é uma técnica que procura aumentar a produtividade em novos projetos de software por meio do emprego de soluções previamente utilizadas e testadas.

Pesquisas na área de reúso de software mostram que para atingir avanços significativos, é necessária uma mudança de paradigma no sentido da definição de famílias de software ao invés de sistemas individuais (CZARNECKI ; HELSEN, 2003). Neste sentido, novas metodologias de desenvolvimento foram definidas como a MDD – *Model Driven Development*, que está se tornando uma abordagem amplamente aceita para o desenvolvimento de software.

O foco da MDD é a utilização de modelos como os principais artefatos no esforço de desenvolvimento de software, aumentando o nível de abstração dos conceitos empregados na solução de um problema, visando a melhora da produtividade e qualidade da aplicação, além da redução da quantidade de código repetitivo (KROISS; KOCH; KNAPP, 2009). Essa técnica, utiliza transformações de modelos, tanto modelo-para-modelo quanto modelo-para-código, e permite simplificar a complexidade do software e aumentar sua manutenibilidade, utilizando na definição dos requisitos do sistema abstrações de alto nível, específicas para o domínio alvo (DIJK, 2009).

Uma das vertentes do MDD é o chamado Desenvolvimento de Software Dirigido a Modelos – MDSD (do inglês *Model-Driven Software Development*). Assim como em toda técnica MDD, o MDSD fornece aos modelos uma maior importância no processo de desenvolvimento de software (MORENO; ROMERO; VALLECILLO, 2008). Porém, o seu foco é criar software mais centrado nos conceitos do domínio que nos conceitos tecnológicos (STAHL; VOELTER; CZARNECKI, 2006). Os conceitos do domínio específico podem ser oriundos de requisitos de usuário ou do negócio de uma organização. Em relação ao processo, se diferencia de outras abordagens por preconizar um estilo ágil de desenvolvimento de software, que deve ser aplicado para permitir entregas rápidas (CADAVID et al. 2009).

A abordagem MDSD geralmente preconiza o uso de Linguagens Específicas de Domínio – DSL (do inglês *Domain Specific Languages*), que são comumente utilizadas para a especificação dos modelos durante o processo de desenvolvimento, com o objetivo de melhorar a expressividade, abstração e consistência dos mesmos com o domínio alvo (STAHL; VOELTER; CZARNECKI, 2006). Linguagens Específicas de Domínio são criadas para um domínio específico e também são conhecidas como linguagens especializadas, orientadas a problemas ou de propósito específico (MERNIK; HEERING; SLOANE, 2005). Uma das vantagens de sua utilização para modelar domínios específicos é sua capacidade de prover um metamodelo do domínio alvo, possibilitando a validação dos modelos especificados baseados nessa formalização.

Dentro deste contexto, este trabalho procura contribuir com a área de desenvolvimento de software baseado em modelos, modelagem de interação com o usuário, e incorporação de regras de negócio de restrição de entrada de dados em sistemas de informação, utilizando o paradigma de Desenvolvimento de Software Dirigido a Modelos. Para esse fim, uma plataforma composta de linguagens específicas de domínio para modelar a *interação com o usuário* e *regras de negócio de restrição entrada de dados*, além de ferramental e

componentes específicos para a manipulação dos modelos e geração dos artefatos típicos de um sistema de informações empresarial é proposta.

1.1 JUSTIFICATIVA

As organizações que desenvolvem e mantêm sistemas de informações empresariais, necessitam de mecanismos que permitam o desenvolvimento destes sistemas de forma alinhada ao negócio (LEVI ; POWELL, 2005). Para que este alinhamento seja efetivo frente às rápidas mudanças no contexto organizacional, não basta que os sistemas de informações sejam desenvolvidos tendo regras de negócio consideradas durante sua especificação. É necessário que haja um meio flexível, e não somente baseado em codificação e conhecimento técnico em desenvolvimento de software, que permita que especialistas de negócio modifiquem e apliquem rapidamente essas regras nos sistemas de informação, sob pena desses mesmos sistemas tornarem-se obstáculos à consecução dos objetivos empresariais. Além disso, a manutenção, armazenamento e validação das regras de negócio frente ao modelo de navegação do usuário são requisitos que o ferramental de suporte ao desenvolvimento e manutenção destes sistemas deve apresentar para permitir uma interação mais coerente e fácil com o modelo conceitual da aplicação (BAJEC ; KRISPER, 2005a). Conforme estes mesmos autores, um dos principais problemas na especificação e manutenção de regras de negócio em sistemas é a **falta de ferramental para geração de código baseado em sua especificação**, além da necessidade do alinhamento destas regras com o modelo de interação com o usuário para permitir que as regras possam ser verificadas e validadas em consonância com este modelo.

Em sua fase embrionária, é comum que empresas possuam recursos escassos para investimento em desenvolvimento, manutenção ou aquisição de sistemas de informação, o que torna mais crítico o uso efetivo de técnicas e processos que permitam a construção e manutenção desses sistemas mais produtiva e menos custosa. Ter a capacidade de alterar regras de negócio rapidamente nos sistemas de informação, para satisfazer às mudanças no contexto interno e externo de organizações é fator primordial para a continuação do negócio e evitar a perda de competitividade.

1.2 MOTIVAÇÃO

Os problemas da cooperação dos especialistas do negócio diretamente na produção do vocabulário e regras de negócio (FAYOUMI ; YANG, 2012), e da tradução manual destas regras para os sistemas de informação (BAJEC ; KRISPER, 2005a), podem ser resolvidos com a utilização de uma linguagem natural controlada baseada no padrão SBVR, juntamente com técnicas de desenvolvimento dirigido a modelos para a propagação automática das mesmas para os sistemas de informação (BAJEC ; KRISPER, 2005a). Uma das possíveis soluções para o alinhamento dos sistemas de informações empresariais com os requisitos de negócio é o uso da abordagem generativa do *desenvolvimento de software dirigido a modelos*, com o suporte de *linguagens específicas de domínio*, para incorporar *regras de negócio*, descritas em SBVR, no desenvolvimento dos sistemas de informação.

Entretanto, é necessário que a solução possibilite que tanto as regras de negócio descritas em SBVR, quanto o modelo de interação com o usuário da aplicação sejam contemplados, e possam ser **integrados** e **validados** entre si, visto que o SBVR foi concebido para definir **"o quê"** o sistema deve fazer e não **"como"** o sistema funcionará (MARINOS ; KRAUSE, 2009; NJONKO ; EL ABED, 2012, grifos nossos). Além disto, é desejável que a modelagem desta interação com o usuário seja baseada em um modelo que possa promover a interoperabilidade com outros padrões.

Diversos trabalhos sobre SBVR já mostraram o porquê e a importância da necessidade de incorporação de regras de negócio diretamente no processo de desenvolvimento de sistemas de informação (BAJEC ; KRISPER, 2005a; MARINOS; KRAUSE, 2009; JESUS, 2013). Entretanto, como o padrão SBVR não especifica explicitamente como verificar uma regra de negócio em um sistema de informação, é necessário que uma solução para seu uso em sistemas de informação defina como, e quando, dentro do ciclo de vida de uma aplicação de software, estas regras serão verificadas. Esta dissertação propõe uma solução para incorporar *regras de negócio de validação de entrada de dados*, descritas em SBVR nos sistemas de informação, integrá-las a um processo de desenvolvimento dirigido a modelos e de como, e quando, aplicar estas regras nos sistemas, integrando-as aos artefatos de software. Também, foi desenvolvida uma linguagem que permitisse fazer com que as regras de negócio pudessem ser integradas ao modelo de interação com o usuário definido pelo padrão IFML. Para esse fim, o modelo da linguagem implementada no trabalho está fundamentado na especificação formal da IFML, que até recentemente não existia, para especificação de um modelo de interação com o usuário.

No âmbito de regras de negócio descritas em SBVR, este trabalho foca nas problemáticas da *especificação*, *implementação* e *gestão* destas regras. Outrossim, a

propriedade multilinguística definida no padrão SBVR permite que as regras de negócio sejam descritas em qualquer idioma ou que possa ser utilizada qualquer conjunto de palavras-chave para representar os conceitos. Entretanto, dentre as várias propostas analisadas (RAJ; PRABHAKAR; HENDRYX, 2008; RAIMUND et al., 2008; NEMURAITE et al. 2010; MARINOS; GAZZARD; KRAUSE, 2011; SUL et al. 2011; AFREEN ; BAJWA, 2011; NJONKO ; EL ABED, 2012; BACHERLER et al. 2012; SELWAY et al. 2013; JESUS 2013), não foram encontradas respostas completas de como dar suporte a estes aspectos, a não ser com o esforço de traduzir todo o ferramental para o idioma ou linguagem alvo. Neste trabalho, também foi contemplado o aspecto multilíngue do SBVR para permitir que regras de negócio sejam expressas em qualquer idioma.

1.3 OBJETIVOS

O objetivo deste trabalho é apresentar uma solução para a especificação de *Regras de Negócio de Restrição de Entrada de Dados*, utilizando o padrão *Semantics of Business Vocabulary and Rules – SBVR*, aliada à especificação de interações com o usuário baseados na *Interaction Flow Modeling Language – IFML*, que permita modelar sistemas de informação e utilização de técnicas de *desenvolvimento de software dirigido a modelos* para a criação de sistemas de informações empresariais.

Como objetivos específicos almejados para se alcançar o objetivo do trabalho, temos:

a) propor uma solução de transformação sistemática de regras de negócio (neste trabalho serão tratadas somente regras de restrição de entrada de dados), definidas através de uma linguagem específica de domínio, que implementa o padrão *Semantics of Business Vocabulary and Rules – SBVR*, para a criação de artefatos de software baseado em técnicas desenvolvimento dirigido a modelos;
b) propor uma linguagem específica de domínio, que implemente os conceitos do padrão *Interaction Flow Modeling Language* – IFML, para modelar necessidades de interação com o usuário em sistemas de informação;
c) propor uma solução de incorporação de componentes prontos e código criado manualmente no código gerado no processo de desenvolvimento de software dirigido a modelos;

d) propor uma ferramenta para auxiliar especialistas de negócio e de desenvolvimento na especificação de regras de negócio de maneira assistida em um editor personalizado criado para este fim;
e) propor uma linguagem específica de domínio, que permita expressar os conceitos do metamodelo SBVR *em qualquer* idioma;
f) propor um editor para modelagem de interação com o usuário através de uma linguagem específica de domínio baseada no padrão IFML;
g) ilustrar como a abordagem sugerida permite gerar código para diversas plataformas alvo a partir de um modelo especificado usando as linguagens especificas de domínio propostas, baseado em técnicas de desenvolvimento de software dirigido a modelos;
h) realizar um estudo de caso para o domínio de comércio eletrônico, com o objetivo de validar a solução proposta.

1.4 ORGANIZAÇÃO DO TRABALHO

O restante do trabalho é organizado como segue. O Capítulo 2 apresenta as principais bases teóricas sobre regras de negócio e a abordagem de desenvolvimento dirigido a modelos. O Capítulo 3 apresenta trabalhos relacionados. O Capitulo 4 apresenta as contribuições do trabalho mostrando a proposta de desenvolvimento dirigido a modelos sugerida, as decisões de projeto relativas à implementação da solução, a abordagem do uso de regras de negócio no processo de desenvolvimento juntamente com o modelo de interação com o usuário e as ferramentas de apoio à modelagem do sistema e geração de código criadas. O Capítulo 5 apresenta um estudo de caso de desenvolvimento de sistemas de comércio eletrônico utilizando a abordagem de desenvolvimento e ferramentas propostos. Por fim, as conclusões, resultados obtidos, limitações e trabalhos futuros são apresentados no Capítulo 6.

2 REGRAS DE NEGÓCIO E DESENVOLVIMENTO BASEADO EM MODELOS

Neste capítulo serão abordados tópicos importantes para a compreensão do trabalho. Inicialmente serão introduzidos os principais conceitos acerca de requisitos de software e sobre a abordagem baseada em regras – "*Business Rule Approach*". É descrito como o padrão *Semantic of Business Vocabulary and Rules* – SBVR implementa os conceitos dessa abordagem. Subsequentemente mostra-se conceitos sobre desenvolvimento baseado em modelos e suas várias vertentes, sua correlação com o SBVR, linguagens específicas de domínio e o *Interaction Flow Modeling Language* – IFML.

2.1 REQUISITOS DE NEGÓCIO E REGRAS DE NEGÓCIO

Requisitos de Software podem ser definidos como uma capacidade, ou condição necessária que o sistema deve apresentar para permitir que um usuário possa resolver um problema ou atingir um objetivo, ou ainda, satisfazer uma imposição formal como um contrato (LEFFINGWELL ; WIDRIG, 1999). Desta forma, requisitos de software norteiam as características que o software deve apresentar para que satisfaça sua finalidade. Existem vários tipos de requisitos que podem ser definidos para um sistema de informação. Requisitos funcionais representam quais funcionalidades um artefato de software deve apresentar para o usuário, enquanto que requisitos arquiteturais restringem ou especificam quais características a arquitetura de um sistema de informação deve apresentar.

Dentre os diversos tipos de requisitos, os requisitos do negócio constituem um importante artefato em qualquer esforço de desenvolvimento de sistemas empresariais pois expressam sua realidade estrutural e funcional, ou seja, definições sobre sua estrutura ou expectativas funcionais do negócio. Esse tipo de requisito é comumente denominado de **regras de negócio**. O papel das regras de negócio na modelagem de sistemas de informação empresariais é expressar a estrutura do negócio, ou permitir o controle do comportamento de um ou mais de seus processos (BAJWA; LEE; BORDBAR, 2011).

Regras de negócio definem a estrutura (regras estruturais), ou apresentam alguma restrição a algum aspecto comportamental do negócio ou organização (regras operacionais). Algumas características de regras de negócio são mostradas a seguir (MORGAN, 2002; HAY; HEALY, 2000; BAJEC et al. 2000):

a) atomicidade: regras de negócio são atômicas e não podem ser quebradas ou divididas sem perda de significado. Por exemplo, a regra *"É obrigatório que cada ordem de*

compra possua uma data de compra" não pode ser dividida sem deixar de representar a intenção original;

b) precisão e verificação: regras de negócio não podem ter significado ambíguo e devem poder ser verificadas. Regras de negócio não podem ter várias interpretações em um mesmo contexto, por exemplo, *"Uma compra deve ser uma experiência agradável para o cliente"* não pode ser considerada uma regra de negócio;

c) consistência: uma regra de negócio não pode entrar em conflito com outra regra no mesmo contexto;

d) simplicidade: regras de negócio devem ser simples e concisas permitindo sua fácil manipulação por agentes de negócio;

e) expressão: regras de negócio precisam ser formalmente expressas – graficamente ou por meio de alguma linguagem formal;

f) natureza declarativa: regras de negócio são declarativas e não procedurais. Isto significa dizer que elas não indicam como algo deve ser realizado, em termos de passos para mover-se de um estado para outro, mas o que é necessário, obrigatório ou proibido.

É desejável que essas características sejam expressas nas regras de negócio, todavia, não implicam que elas contenham informações estranhas ao negócio. Por exemplo, não é desejável que regras de negócio contenham informações sobre:

a) ***como*** serão avaliadas ou executadas;

b) ***onde*** serão avaliadas ou executadas;

c) ***quem*** é responsável por sua avaliação e execução;

d) ***quando*** elas devem ser avaliadas ou executadas;

e) ***porquê*** elas precisam ser avaliadas ou executadas.

Estas características nortearam a definição de modelos que descrevem regras de negócio com rigor formal, como é o caso do SBVR discutido a seguir.

2.2 SEMANTICS OF BUSINESS VOCABULARY AND BUSINESS RULES – SBVR

Esquemas Conceituais são muito importantes no desenvolvimento de sistemas de informação e são amplamente utilizados na indústria para representar o conhecimento capturado por esses sistemas (KLEINER; ALBERT; BÉZIVIN, 2009). Entretanto, esses esquemas são muito complexos para serem utilizados diretamente pelos interlocutores do negócio para descrever um sistema. O ideal é que analistas de negócio e usuários possam

especificar suas necessidades por meio de um ferramental que suporte facilmente a definição dos requisitos por meio de uma linguagem a que já estão habituados – a linguagem do negócio expressa em *linguagem natural*.

Recentemente a OMG liberou uma nova versão (1.2) de seu padrão Semântico para Vocabulário e Regras de Negócio – SBVR (do inglês *Semantics of Business Vocabulary and Business Rules*) (OMG, 2013) que é um modelo semântico para representar regras de negócio em linguagem natural, e sua interoperabilidade entre organizações e ferramentas. Segundo a OMG, o padrão SBVR é aplicável a todos os domínios de vocabulários e regras de negócios, em todas as atividades e ramos (OMG, 2013).

Devido ao fato de ser baseada em um metamodelo formal e possuir fundamentação matemática, a representação de regras de negócio em SBVR as tornam passíveis de processamento computacional, por poderem ser expressas por meio de uma linguagem natural formal, de fácil compreensão por desenvolvedores e *stakeholders* (AFREEN ; BAJWA, 2011). Desta maneira, é possível a transformação de tais regras em artefatos de software como modelos de objetos, componentes de software e esquemas conceituais. (BAJWA; LEE; BORDBAR, 2011). Entretanto, esse padrão não define como os modelos baseados no SVBR podem ser transformados ou executados. Cabe a cada implementação esta definição.

Além disto, como o padrão SBVR é fortemente baseado em ontologias e modelos conceituais de dados, permitindo que seja definida a estrutura conceitual do negócio, ele (e suas possíveis extensões) pode contribuir para resolver problemas relacionados com a verbalização de regras de negócio e modelos de dados de software (BROCKE et al. 2011).

As seções seguintes irão discutir os principais conceitos presentes no padrão SBVR e que permitem modelar conceitualmente o vocabulário e regras de negócio de um determinado domínio.

2.2.1 Representação dos conceitos

A palavra ***"Semantic…"*** no SBVR indica que o padrão tem como um dos seus elementos chaves o entendimento do **significado**, ou relações entre os significados de um conjunto de símbolos (INSIDER, 2013). No padrão, tais símbolos não são limitados a texto – podem ser frases, imagens, números, ícones, permitindo que a representação dos elementos do vocabulário de negócio e regras possa ser feita de várias formas. É fato que, nas organizações, esses conceitos e regras estão documentados de alguma forma, notadamente em documentos texto. O padrão SBVR sugere a representação dos seus conceitos utilizando uma linguagem

natural formal denominada SBVR *Structured English* – SBVR-SE (OMG, 2013), mas não torna nenhuma linguagem um padrão para a representação do modelo. Entretanto, como mencionado anteriormente, no SBVR a representação dos conceitos e o significado são completamente independentes. Por exemplo, o Anexo J do padrão (OMG, 2013) mostra um exemplo de representação baseado no *Object-Role Modeling* – ORM. Outra representação conhecida é o *RuleSpeak - Business Rule Notation* (ROSS, 2009). Usando o metamodelo SBVR pode-se construir modelos semânticos para o negócio (RAJ; PRABHAKAR; HENDRYX, 2008), consequentemente é viável a representação desse modelo semântico utilizando um diagrama ou outra forma qualquer (LINEHAN 2007; RAJ, PRABHAKAR; HENDRYX, 2008). Não obstante, a forma mais conhecida, e um padrão de fato de representação dos conceitos definidos pelo padrão SBVR continua sendo o SBVR-SE (MARINOS; MOSCHOYIANNIS; KRAUSE, 2010).

O objetivo da especificação SBVR é definir vocabulário e regras para documentar a semântica dos vocabulários de negócio e vocabulário de regras de negócio, visando a troca destas informações entre usuários, organizações e ferramentas (OMG, 2013). Primeiro, o padrão define o que é chamado de *"Vocabulário para Descrever Vocabulários de Negócio"*. O vocabulário é constituído principalmente pelos conceitos de negócio (termos), conceitos verbais e seus significados, que são utilizados por pessoas do negócio, e as comunidades envolvidas no mesmo, para expressar seus conceitos e regras para sua consumação. O vocabulário, desta forma, é utilizado para comunicar os principais conceitos do negócio envolvidos em sua jurisdição. Por *"envolvido em sua jurisdição"*, entende-se que tais conceitos podem ser alterados e revisados pelo negócio (INSIDER, 2013).

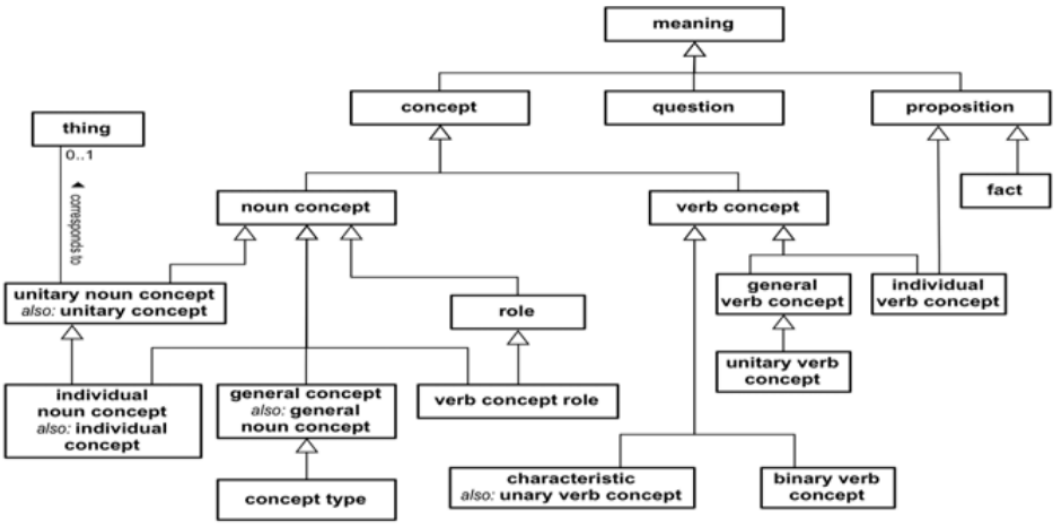
O segundo vocabulário do padrão é o *"Vocabulário para Descrever Regras de Negócio"* e realiza o conceito chamado de *"Business Rule Approach"* (INSIDER, 2006) que lida com a especificação do significado de regras de negócio. Estas, por sua vez são baseadas no *"Vocabulário para Descrever Vocabulários de Negócio"*. Esta relação é a base do que se chama de *"mantra de regras de negócio"*, que resume a definição concebida pelo *Business Rule Group* (HAY ; HEALY, 2000) pela qual *"Regras são baseadas em conceitos verbais e conceitos verbais são baseados em conceitos nominais expressos por termos"* que pode ser melhor entendida pela Figura 1, e detalhado nas próximas seções.

Figura 1 - Modelo do Esquema SBVR

Fonte: Adaptado de Raj, Prabhakar e Hendryx (2008).

O metamodelo da Figura 2 é o modelo definido no padrão SBVR em sua seção 8.1 (SBVR, 2013). Existem vários tipos de conceitos definidos no padrão SBVR, neste trabalho. Três tipos são de interesse: *"General Concept"*, *"Individual Concept"* e *"Verb Concept"*. Esses conceitos são resumidamente mostrados a seguir:

Figura 2 - Metamodelo SBVR



Fonte: SBVR (2013).

- Conceitos ou termos do SBVR

Conceitos (*'concepts'*) no padrão SBVR são definidos como um conhecimento (*'meaning'*) sobre algo com características únicas. No padrão SBVR existem vários tipos de conceitos. As palavras *Conceito* ou *Termo* são utilizadas como uma generalização dos conceitos nominais.

Termos são substantivos utilizados na definições de um conceito, ou entidade do negócio, que fazem parte do vocabulário do negócio que está sendo modelado. Desta forma, os **termos** do vocabulário de negócio representam uma coleção de entidades do negócio e suas instâncias (SUL et al. 2011). Sendo assim, são geralmente representados por um substantivo. O padrão SBVR define alguns tipos de termos para diferenciar o significado dos mesmos. Assim, termos como *cliente* são definidos no padrão como *General Concept*, que designam conceitos baseados em suas características comuns tais como *nome, filiação, local de nascimento e CPF*. Outro tipo definido no padrão são os *Individal Concept*, que designam instâncias dos conceitos comuns no âmbito do negócio como o conceito "*Brasil",* que é uma instância do conceito *país*.

A Figura 3 mostra um exemplo de termos SBVR.

Figura 3 - Exemplo de Termos SBVR

```
- client

- order
    description: A order is an instruction to supply one or more goods
    for a client
```

Fonte: Autoria própria

- Regras de Negócio

Um segundo tipo de elemento que o padrão SBVR engloba são as regras de negócio. As regras de negócio no padrão SBVR representam as regras que estão sob a jurisdição do negócio e definem sua estrutura ou regem seu comportamento (OMG, 2013). As regras de negócio definem a estrutura ou operação do negócio e provêm elementos que guiam suas ações.

As regras no padrão SBVR têm consistência com a lógica formal e são construídas como formulações modais fechadas. As formulações de regras de negócio no padrão estão

baseadas em *conceitos verbais*. Elas representam semânticas de dois tipos: deônticas (representam obrigações, permissões ou proibições) ou aléticas (necessidades, possibilidades ou impossibilidades) sobre algum conceito do negócio. Assim, regras de negócio em SBVR são construídas aplicando obrigação ou necessidade a um ou mais conceitos verbais. Por exemplo, a regra *"É obrigatório que uma compra tenha exatamente um cliente associado"* é baseada no conceito verbal *"compra tem cliente associado"*.

Regras de definição, ou estruturais, são utilizadas para definir como uma organização e seu negócio estão estruturados e denotam uma necessidade. Por exemplo, *"É necessário que cada cliente tenha pelo menos uma conta bancária."*. As regras comportamentais, ou operacionais, definem a conduta de uma entidade em um processo operacional da organização e regulamentam uma obrigação. Por exemplo: *"Cada cliente só pode sacar, no máximo, R$ 1.000,00 por dia"*

O padrão SBVR define um conjunto de *Formulações Lógicas*, de forma a constituir as regras de negócio de forma estruturada e consistente. Estas formulações estão definidas por meio de uma sintaxe, independente de linguagem, para representar o significado das regras de negócio de maneira que possa ser computacionalmente processada. Esta sintaxe é constituída de uma série de estruturas lógicas que definem uma proposição. As formulações atômicas são estruturas que não contém operadores lógicos modais e se baseiam em um conceito verbal. Os conceitos verbais utilizam um verbo para conectar os conceitos do vocabulário de negócio. Ainda existem as quantificações e formulações modais. Enquanto as quantificações definem a cardinalidade de um conceito, as formulações lógicas utilizam um operador lógico para expressar o sentido de uma regra de negócio. Assim a regra de negócio – *"It is obrigatory that each order specifies one payment method"* é baseado em uma formulação atômica, ou conceito verbal: *"order specifies payment method"*, contém duas quantificações *"each"* e *"one"*, e um operador lógico de obrigação *"It is obrigatory that"*.

1.1.2 SBVR Structured-English

O SBVR *Structured-English* é a sintaxe definida no Anexo A da versão 1.2 do padrão (OMG, 2013) que formaliza um padrão em inglês estruturado - SBVR-SE (*SBVR Structured English*), para definição do vocabulário e regras do negócio. De acordo com o esse documento existem diversas formas como esses conceitos (vocabulário e regras de negócio) podem ser representados e combinados. Conforme já mencionado anteriormente, qualquer

representação baseada no padrão pode ser transformada formalmente em termos de um vocabulário e regras SBVR. Isto é possível por meio do metamodelo e pelos conceitos mostrados anteriormente que mapeiam as representações para o vocabulário ou formulações lógicas do SBVR.

Os estilos de representação dos conceitos e formulações no SBVR-SE são o que seguem:

a) **termos**: os termos são sublinhados, e em cor verde e são aplicados a conceitos nominais
b) **nomes próprios**: sublinhado duplo, e em cor verde com a primeira letra em maiúsculo
c) **verbos**: em itálico, e em cor azul para definir conceitos verbais
d) **palavras-chave**: em laranja e negrito, e servem para definir outros símbolos linguísticos de formulações lógicas definidos no padrão SBVR. São utilizados para definir construções da lógica formal, no qual o SBVR é pautado, como por exemplo o conceito de obrigação – "*It is obligatory that***"**, em regras de negócio. Esta representação é também utilizada para demarcar outras construções definidas pelo padrão SBVR como palavras-chave auxiliares como a palavra "*that"*.

2.2.2 SBVR e MDA

O padrão SBVR é inteiramente compatível com a abordagem MDA, pois possui um metamodelo formal de todos seus conceitos. O modelo conceitual do SBVR está situado no nível CIM (*Computation Independent Model*) do MDA (RAJ; PRABHAKAR; HENDRYX, 2008). Quando regras de negócio são escritas utilizando como base o padrão SBVR, um esquema conceitual e um modelo semânticos são definidos. Esse modelo semântico pode ser criado utilizando qualquer linguagem, e servir para mais de um modelo de negócio, pois o metamodelo semântico do SBVR é completamente independente da representação (OMG, 2013).

Devido ao arcabouço formal que embasa o padrão SBVR, especificação de regras de negócio definidas utilizando esse padrão são candidatas à aplicação de técnicas MDD, pois baseiam-se em um metamodelo formal e bem definido. Desta maneira, é possível a transformação de tais regras em artefatos de software como modelos de objetos, componentes e esquemas conceituais (BAJWA; LEE; BORDBAR, 2011). Além disto, o padrão provê um metamodelo para definir e modelar conceitos de negócios e seu vocabulário, de modo que um esquema conceitual pode ser inferido diretamente utilizando uma linguagem mais próxima a

linguagem natural utilizada pelos usuários do negócio, normalmente denominada linguagem controlada – CL (do inglês *Controlled Language*). Esse tipo de linguagem pode ser utilizada diretamente pelos usuários para definir diversas regras que restringem o negócio, e que ao utilizarem o formalismo provido pelo padrão SBVR, permite que as mesmas sejam passíveis de processamento por sistemas computacionais (BAJWA; LEE; BORDBAR, 2011).

2.3 MODEL BASED ENGINEERING - MBE

Sob a égide da Engenharia de Aplicações Baseada em Modelos – MBE (*Model Based Engineering*), existem vários enfoques que são desenvolvidos e aplicados (DUARTE; CARDONA, 2011). Nas seções seguintes são descritos os conceitos mais importantes na área de desenvolvimento baseado em modelos e seus inter-relacionamentos.

2.3.1 Model Driven Engineering

O *Model Driven Engineering* – MDE diz respeito às técnicas de utilização sistemática de modelos como os **elementos chaves** no processo de construção de software e que devem ser utilizados em todo ciclo de vida de um projeto (DUARTE ; CARDONA, 2011). O estudo desta área é abrangente e envolve vários conceitos. Seus principais objetivos e esforços são os de tentar organizar os conhecimentos do desenvolvimento baseado em modelos, de propor um *framework* que defina claramente as metodologias envolvidas na construção de sistemas em qualquer nível de abstração e organizar e automatizar as atividades de testes e validação (FONDEMENT ; SILAGHI, 2004).

Alguns trabalhos estudam os conceitos envolvidos no MDE (STAHL; VOELTER; CZARNECKI, 2006; FAVRE, 2004; FONDEMENT; SILAGHI, 2004; RIVERA; ROMERO; VALLECILLO, 2009; LIDDLE, 2011). Favre (2004) faz a formulação de um modelo que abstrai os próprios conceitos do MDE, enquanto outros se concentram nos processos envolvidos (FONDEMENT; SILAGHI, 2004) e em seus desafios (RIVERA; ROMERO; VALLECILLO, 2009).

Dentre os vários conceitos relacionados com o MDE, o principal é o conceito de modelo. Neste trabalho, é utilizado o conceito de que modelo é uma abstração ou simplificação de um sistema, e seu ambiente, visando melhorar ou facilitar seu entendimento (STAHL; VOELTER; CZARNECKI, 2006). Entretanto, para poder ser corretamente utilizado no contexto do MDE, um modelo precisa ter uma sintaxe e semântica bem definidas,

tornando-se, assim, passível de automação (FAVRE, 2004). O enfoque que a MDE dá aos modelos é de que são artefatos de importância de primeiro nível na construção de um sistema. A ideia promovida pelo MDE é a de que, na construção de um sistema, diversos modelos devem ser utilizados e refinados, cada um com diferentes níveis de abstrações (FONDEMENT ; SILAGHI, 2004). Não obstante, visam capturar e expressar de forma efetiva o domínio da aplicação a ser modelada. Esta visão confronta outras técnicas em que os modelos só servem para documentação (DUARTE ; CARDONA, 2011).

Outro conceito igualmente importante, e relacionado com o conceito de desenvolvimento baseado em modelos, é o conceito de transformação de modelos (FONDEMENT ; SILAGHI, 2004). Apesar de não ser estritamente necessária, ou obrigatória, a construção de uma aplicação baseada em modelos geralmente requer algum tipo de transformação, em alguns casos em várias etapas, para tornar os conceitos expressos no modelo em componentes de software executáveis.

2.3.2 Model Driven Development

O Model Driven Development (MDD) é uma aplicação particular da MDE em que os modelos são descritos de forma a serem executados diretamente ou transformados em código executável em um nível de abstração menor (DUARTE ; CARDONA, 2011). Alguns autores, como Stahl, Voelter e Czarnecki (STAHL; VOELTER; CZARNECKI, 2006), consideram esta denominação sinônima, e menos precisa, do conceito de *Model Driven Software Development* – MDSD, enquanto outros utilizam os termos MDD e MDE com o mesmo significado (LIDDLE, 2011). Como este trabalho não possui um cunho epistemológico dos termos associados ao conceito de MDD, não tratará destas diferenças, e utilizará os conceitos MDD e MDSD como sinônimos, considerando o MDE uma área de estudo mais abrangente, e que aquelas são seus subdomínios.

2.3.3 Model Driven Software Development

Outro padrão que utiliza o conceito de priorização de modelos e seu emprego como artefatos chave durante todas as fases do processo de desenvolvimento de software (especificação e análise, design, implementação e testes), é o *Model Driven Software Development* - MDSD (MORENO; ROMERO; VALLECILLO, 2008).

Esta abordagem preconiza que os modelos devem ser colocados no mesmo nível de importância que o código final da aplicação, sendo que ambos devem ser desenvolvidos e aperfeiçoados durante todo o processo de desenvolvimento, possuindo estreita relação e integração, de forma que seu conjunto, e não só o código da implementação final, representem a solução sendo desenvolvida. Tais modelos são criados e manipulados através de um amplo espectro de técnicas incluindo *model-driven requirements engineering*, *model-driven design*, transformação e geração de código baseado nos modelos, *model-driven testing*, *model-driven software evolution*, dentre outros (STAHL; VOELTER; CZARNECKI, 2006).

O MDSD procura encontrar abstrações específicas de um domínio, torná-las acessíveis na construção de aplicações através de sua especificação formal, utilizando uma ou mais linguagens específicas de domínio – DSL (do inglês *Domain Specific Language*), tornando assim possível a automação da transformação dos modelos baseados neste formalismo. As linguagens podem estar disponíveis em ambientes gráficos, textuais, ou ambos, e é necessário que exista suporte à transformação e execução desses modelos. O objetivo final do MDSD é que o processo de desenvolvimento seja melhorado nos quesitos produtividade e qualidade (STAHL; VOELTER; CZARNECKI, 2006).

A abordagem MDSD visa a constante manutenção dos artefatos criados por parte das equipes de desenvolvimento, tanto de modelo-a-modelo (M2M – do inglês *model-to-model*) e modelo-a-código (M2T – do inglês *model-to-text*). Além do uso de uma DSL, geralmente é necessário o suporte e manutenção dos *templates* arquiteturais, que são utilizados nas transformações M2T, além ferramental para dar amparo a todo processo de desenvolvimento.

No MDSD, que tem suas raízes na engenharia de linhas de produto de software, e que por sua vez tem como premissa a criação de famílias de produtos de software para um segmento específico, os modelos são preferivelmente especificados utilizando uma ou mais Linguagens Específicas de Domínio – DSL (do inglês *Domain Specific Languages*). Estes modelos devem ser então submetidos a transformações M2M e M2T, baseadas em uma arquitetura previamente definida e semânticas formais. Em conjunto com código criado manualmente, estes modelos definem a solução final do software. Eles não são utilizados somente para documentação, mas como uma parte integrante e fundamental da aplicação produzida. É importante notar que partes da aplicação podem ser feitas utilizando código criado manualmente, não sujeitos ao processo de transformação.

Nesta perspectiva, modelos são definidos como uma representação abstrata de alto nível da estrutura do software, função ou comportamento, e compartilham a mesma importância que o código final da aplicação (STAHL; VOELTER; CZARNECKI, 2006).

2.3.4 Model Driven Architecture

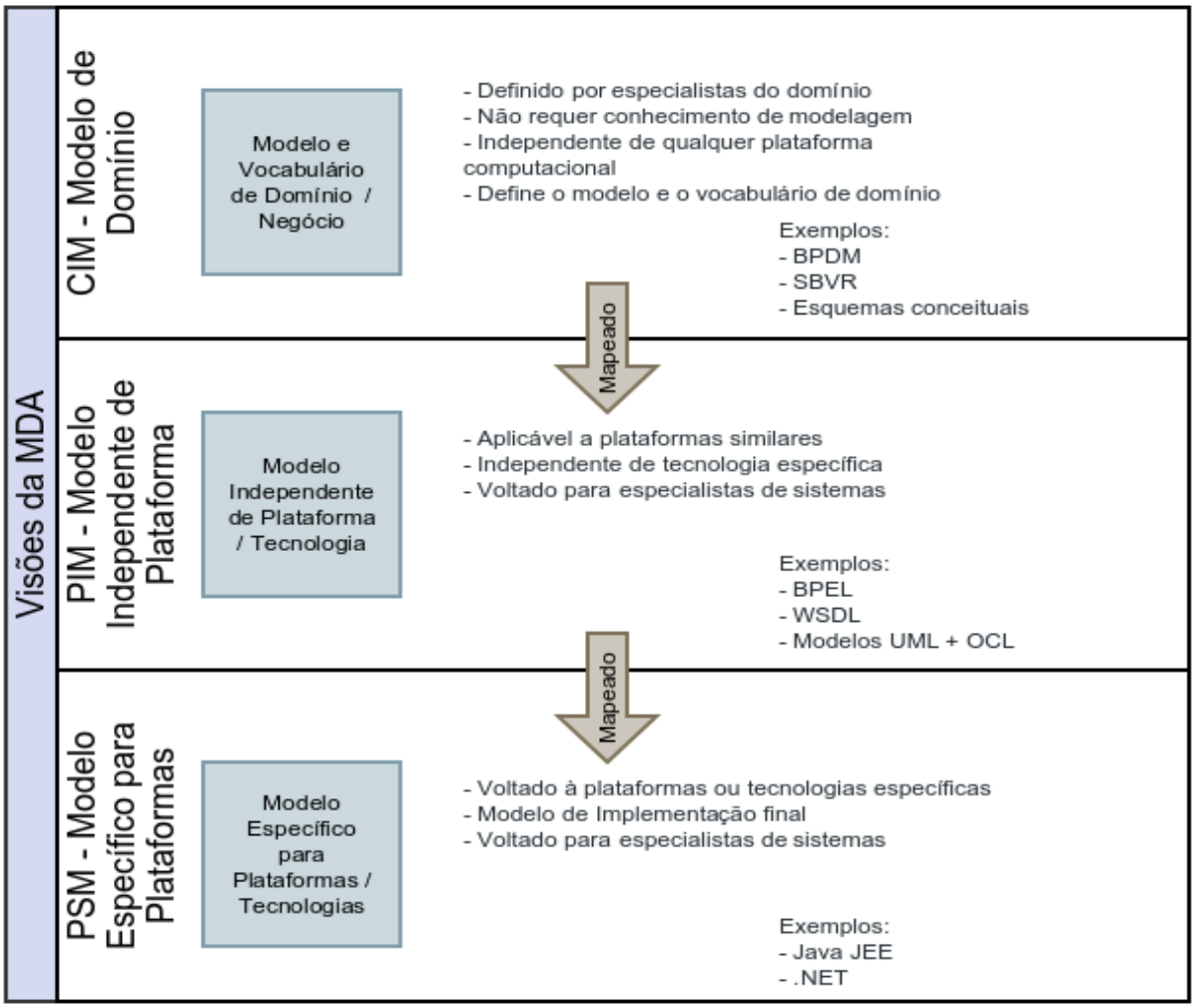
O *Model Driven Architecture* – MDA (OMG, 2014) é a iniciativa da OMG na área de MDD (MORENO; ROMERO; VALLECILLO, 2008). Ele representa um framework conceitual para a criação de aplicações utilizando os conceitos da MDE. Os modelos e artefatos definidos e utilizados neste tipo de abordagem geralmente são produzidos com a UML – *Unified Modeling Language*, utilizando *UML Profiles* para o caso dos modelos (STAHL; VOELTER; CZARNECKI, 2006), e outros padrões da OMG como MOF – *MetaObject Facility*, XMI - *XML Metada Interchange*, e o CWM – *Common Warehouse Metamodel* (LIDDLE, 2011).

Os principais objetivos do padrão MDA são a promoção de interoperabilidade e portabilidade utilizando níveis diferentes de abstrações para modelar todos os aspectos envolvidos na solução. Desta forma, separa-se modelos de negócio e modelos específicos da plataforma alvo do sistema, permitindo que ambos os modelos evoluam independentemente, e possibilitando, através de transformações M2M (*model-to-model*) e M2C (*model-to-code*), a geração da aplicação final (STAHL; VOELTER; CZARNECKI, 2006).

Além disto, a OMG tem endossado fortemente a utilização da abordagem de desenvolvimento baseado em modelos com o sucesso do seu padrão UML e sua conhecida iniciativa MDA (BETTIN, 2004).

O conceito que baliza o MDA é que para se chegar a um sistema pronto através de transformações de modelos é necessário que seja seguida uma abordagem de transformar modelos mais genéricos em modelos mais específicos (*top-down*) até que os últimos implementem tais conceitos de forma executável em alguma plataforma. Desta forma, a MDA define um modelo de visões que é reproduzido na Figura 4 para efeito de completude desta explanação.

Figura 4 - Modelo de Visões da MDA



Fonte: Adaptado de Linehan (2006).

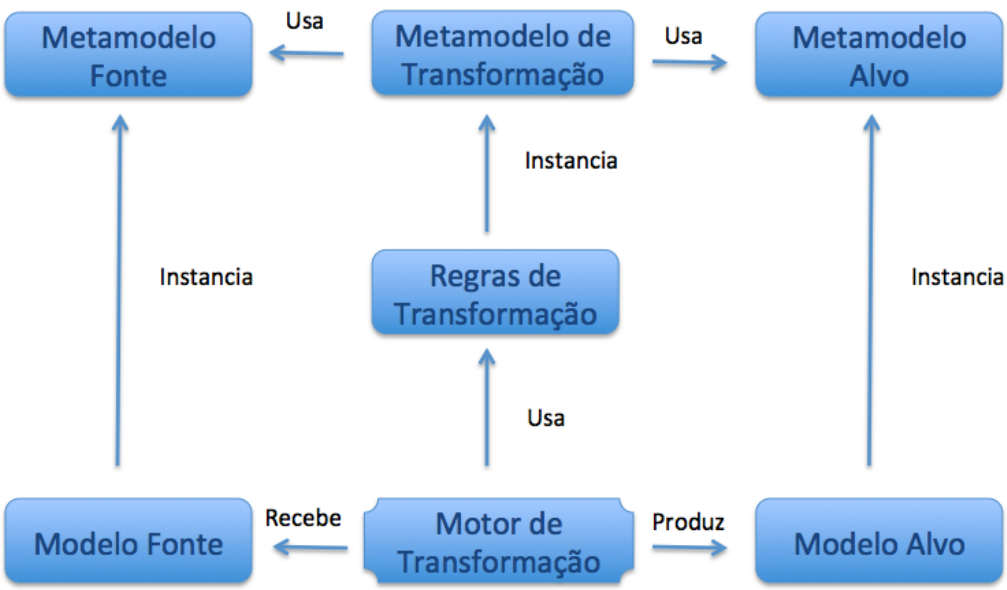
O CIM – *Computation Independent Model*, ou o modelo do domínio, é utilizado para modelar o domínio da aplicação independentemente da plataforma ou arquitetura do sistema. Por não requerer domínio de conceitos de modelagem de sistemas, neste nível, os conceitos geralmente devem ser definidos em uma linguagem próxima ao usuário final da aplicação ou do negócio que está sendo modelado. Desta forma, a validação do modelo torna-se fácil para os *stakeholders* além de permitir a rápida manutenção do modelo. Nesse nível, a intenção é definir um vocabulário de domínio para que o modelo possa ser entendido por todos os envolvidos. Seu público-alvo são os especialistas do domínio. Entretanto deve possuir um mínimo rigor que permita sua tradução para modelos mais específicos. Nesse sentido, modelos CIM permitem reduzir a distância entre os requisitos não formais, geralmente expressos em linguagem natural, e modelos mais específicos para sistemas de informação que necessitam de um maior rigor formal.

Esse modelo então deve ser mapeado e transformado em um PIM – *Platform Independent Model*, que adiciona ao modelo informações arquiteturais sem se comprometer com a plataforma utilizada. Os modelos nesse nível promovem a independência de plataforma, no momento em que relegam para modelos mais específicos decisões necessárias para execução de um sistema em uma plataforma específica. Desta forma, tais modelos podem ser utilizados em diversas plataformas similares, mas de forma neutra em questões específicas de tecnologias empregadas na solução final.

Por fim, este modelo (PIM) é mapeado e transformado em um PSM – *Platform Specific Model*, que adiciona ao modelo do sistema informações inerentes à plataforma em que o sistema será utilizado. Sua função é adaptar o PIM à questões específicas da plataforma alvo do sistema. Para isto, detalhes tecnológicos são adicionados ao modelo permitindo sua tradução para código ou sua direta execução.

Transformações estão previstas em abordagens MDA e podem ser definidas como a geração automática de modelos a partir de outros modelos permitindo o refinamento da solução até a geração do código final. Existem três abordagens para a transformação: baseada em programação, baseada em padrões e baseada em modelos. O processo baseado em modelos está esquematicamente reproduzido na Figura 5.

Figura 5 - Modelo de Transformações MDA



Fonte: Autoria própria.

Neste caso, as transformações devem ser baseadas em um modelo, que está em conformidade com um metamodelo fonte, e resultará em um modelo alvo. Os metamodelos fonte e alvo, definem precisamente os elementos que podem aparecer nos modelos fonte e alvo respectivamente. Assim, uma série de regras podem ser aplicadas, baseadas em um metamodelo de transformação, para que o modelo fonte seja transformado no modelo alvo. Apesar desta abordagem ser a preferida no padrão MDA, utilizamos uma abordagem baseada em codificação pela simplicidade e facilidade de integração com a proposta deste trabalho. Na abordagem baseada em codificação, as regras de transformações estão em componentes codificados para este fim.

Existem várias linguagens e padrões de transformações que podem ser utilizados em uma abordagem de desenvolvimento baseado em modelos. Algumas destas linguagens de transformação de modelos são interpretadas e executadas por um motor de transformação. Por exemplo, o ATL (*ATLAS Transformation Language*) é um tipo de linguagem, na qual as regras são definidas para cada item (elemento ou artefato) do modelo fonte, e que no momento da transformação são aplicadas pelo engenho de transformação no processo de leitura do modelo fonte para produzir algum efeito como a produção de outro modelo ou código. O engenho de transformação infere o tipo de dados do modelo que está sendo considerado (e.g. entidade pessoa, atributo nome) e aplica as regras previamente definidas para cada um destes tipos de entidades. Outras linguagens podem ser utilizadas com o mesmo objetivo, e.g. QVT (*Query/View/Transformation*), MOFM2T (MOF *Model to Text Transformation Language*) e *XTend* (XTend, 2105).

2.3.5 Domain Specific Languages

Como em todas as áreas da ciência e engenharia, sempre existem abordagens genéricas e específicas para resolver determinado problema (DEURSEN; KLINT; VISSER, 2000). A descrição de um problema de um domínio em uma linguagem desenvolvida especificamente para sua solução, tende a ser otimizada e direta, além de possivelmente possuir maior expressividade para a descrição dos conceitos do domínio definido.

Linguagens específicas de domínio são linguagens criadas para um domínio particular e são também chamadas de linguagens especializadas, orientadas a problema ou de propósito específico (MERNIK; HEERING; SLOANE, 2005). Deursen et al (DEURSEN; KLINT; VISSER, 2000), define DSL como linguagens de programação ou linguagens de

especificação executáveis que oferecem um grande poder de expressar, com notações e abstrações focadas e restritas a um domínio, um problema específico. O estudo e utilização de linguagens específicas de domínio não é recente, mas o significado correto do termo ainda é controverso e existem diversas tentativas de formalizá-lo. Entretanto, a procura sistemática de formalização é relativamente nova.

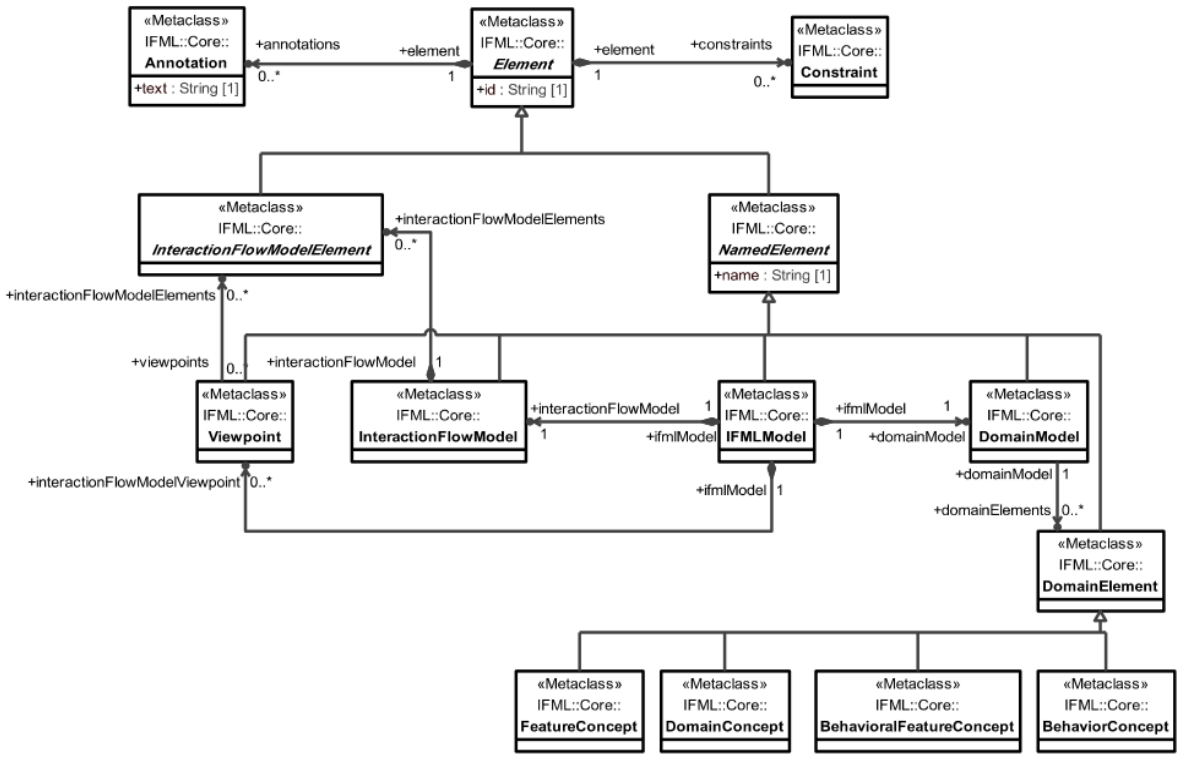
2.4 IFML

O IFML (OMG, 2015) – *Interaction Flow Modeling Language*, é um padrão de modelagem desenvolvido pela OMG para expressar o conteúdo, interação com o usuário e controle do comportamento do *front-end* de sistemas de informação. Ela permite que profissionais de software possam modelar e descrever as principais partes do *front-end* de aplicações. Para isto, a especificação divide o padrão nas seguintes dimensões:

a) *view* - a parte visual da aplicação composta de containers e componentes visuais;
b) regras de negócio e estado - a representação dos objetos que possuem o estado da aplicação e a lógica de negócio a ser executada;
c) dados e conexão com eventos - a conexão entre os componentes visuais e de dados aos eventos;
d) controle de execução de eventos - a lógica responsável por determinar a ordem de execução das ações em resposta a um evento;
e) distribuição da arquitetura - a distribuição dos componentes de controle, dados e lógica de negócio nas camadas da aplicação.

O modelo IFML é ilustrado na Figura 6.

Figura 6 - Metamodelo da IFML



Fonte: IFML OMG (2015).

O elemento *IFMLModel* é o principal módulo que contém os elementos IFML e contém um *InteractionFlowModel* que representa a visualização do usuário de uma aplicação IFML e é composto de: *InteractionFlowModelElements* – uma abstração que representa a generalização de todo elemento de um *InteractionFlowModel* (elementos da *view*); *DomainModel* – elementos do modelo que representa o domínio de negócio e seus dados; *ViewPoints* – representam somente aspectos específicos do sistema para facilitar a compreensão e referência a *InteractionFlowModelElements*.

O *DomainModel* representa o domínio de negócio da aplicação, o conteúdo e comportamento que é referenciado pelo *InteractionFlowModel*. O *DomainModel* contém *DomainElements*. Existem termos especializados dos conceitos de domínio como – *DomainConcept*, propriedades – *FeatureConcept*, comportamento – *BehaviorConcept* e métodos – *BehaviorFeatureConcept*. O *NamedElement* é uma classe abstrata que denota elementos – *Element* (o conceito mais geral da IFML) que possuem um nome. Todos os *Element* possuem *Comments* e *Constraints*. O último é o método de como regras sintáticas definidas pelo IFML podem restringir ainda mais um modelo mais específico.

O *InteractionFlowModel* contém todos os aspectos da interface com o usuário representado por *InteractionFlowModelElements*. *InteractionFlowModelElement* representa

os fundamentos das interações com o usuário. Ele representa todos os elementos que podem participar das conexões com um *InteractionFlow* como *ViewElements*, *Actions* e *Events*. O *InteractionFlow* carrega as informações de um *InteractionFlowEment* (fonte) para outro (alvo), e pode implicar em navegação (*NavigationFlowElement*) entre *InteractionFlowElements*.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo teve como objetivo fornecer o embasamento teórico necessário para o entendimento e acompanhamento da narrativa deste trabalho. Foram discutidos os principais conceitos que permeiam os diversos tópicos relacionados com a abordagem sugerida como requisitos de software, a abordagem "*Business Rule Approach*" para tratar regras de negócio e como o padrão SBVR implementa seus conceitos, conceitos sobre desenvolvimento baseado em modelos, suas várias vertentes e conceitos envolvidos, e o padrão de interação com o usuário IFML.

No capítulo seguinte serão discutidos os trabalhos identificados por meio de uma pesquisa bibliográfica que possuem relevante relação com regras de negócio descritas em SBVR e sua incorporação em sistemas de informação, desenvolvimento baseado em modelos e modelagem de interação com o usuário, que são temas afins aos temas centrais desta dissertação.

3 TRABALHOS CORRELATOS

A seguir são elencados e descritos trabalhos relacionados com esta dissertação. Os trabalhos foram selecionados por meio de uma revisão bibliográfica divididos por tópico de interesse:

a) regras de negócio com SBVR e sua relação com o desenvolvimento dirigido a modelos (MDD) e ferramentas relacionadas;
b) desenvolvimento baseado em modelos, aplicações Web e interação com o usuário.

A metodologia utilizada para a seleção dos trabalhos relacionados foi baseada na afinidade do trabalho pesquisado com a proposta desta dissertação, no tocante às similaridades com os tópicos de interesse descritos anteriormente. Diversos mecanismos de buscas foram utilizados, e.g. *Google*, *Google* Acadêmico e *SiteSeer* para relacionar os artigos de maior relevância sobre os tópicos de interesse. Procurou-se selecionar os periódicos e artigos com maior número de citações para determinar a sua relevância dentro da área de interesse selecionada.

3.1 REGRAS DE NEGÓCIO COM SBVR E MDD

Alguns trabalhos propostos para a utilização do padrão SBVR para definição de regras de negócio e seu processamento em linguagem natural podem ser encontrados na literatura. Nesta seção são discutidos alguns desses trabalhos.

3.1.1 Regras de Negócio e SBVR

Njonko e El Abed (2012) propõem uma metodologia para transformação de regras de negócio, expressas em linguagem natural, utilizando uma abordagem generativa baseada nos conceitos da MDA. Essa proposta visa a criação de modelos executáveis e a definição de representações (e.g. UML, SQL) a partir de regras de negócio descritas em uma linguagem natural e transformadas na notação SBVR a partir de um componente chamado *"NLP Framework"* (*Natural Language Processor Framework*). Nesta solução, posteriormente ocorre uma fase de transformação destas regras de negócio em SBVR, para regras de negócio executáveis por agentes de uma plataforma chamada *Data Excellence Management System* (DEMS). O componente responsável por esta última transformação é chamado de

SBVRtoEXE Framework. Através desse componente, as regras definidas por meio de em linguagem natural são processadas e ambiguidades semânticas são tratadas. Além da análise semântica, são executadas análises léxicas e sintáticas para a identificação de conceitos da SBVR.

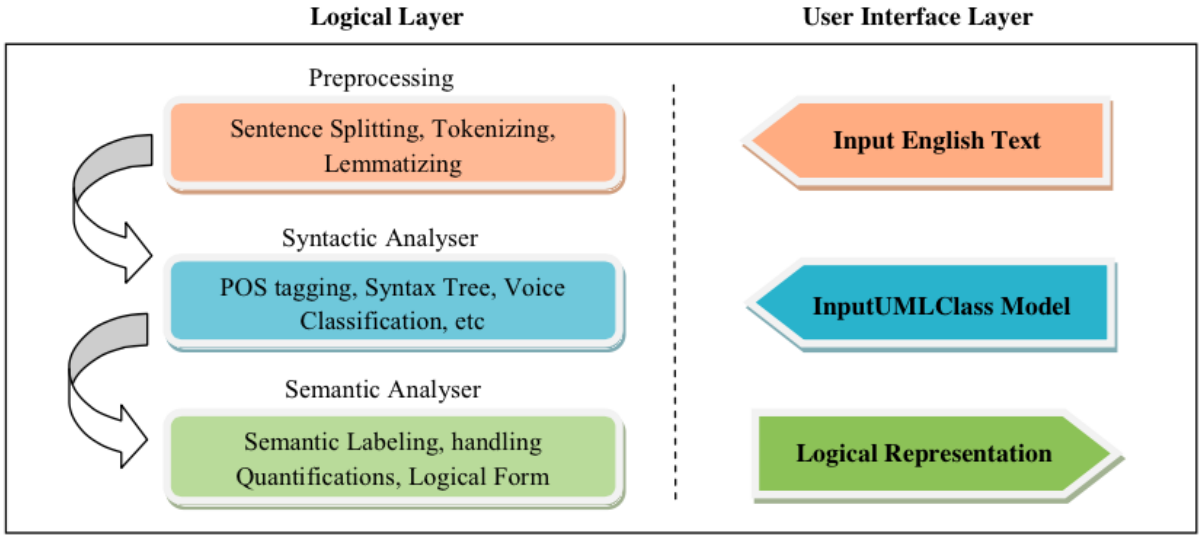
Os seguintes passos são necessários nessa abordagem: segmentação, que consiste em identificar os tokens de uma regra; processamento léxico que relaciona cada token de uma regra a um elemento contextual da regra definida em uma frase como sujeito, verbo ou objeto direto; análise sintática que coloca esses elementos em uma árvore sintática; análise semântica que relaciona os tokens a atores e ações realizadas; extração de conceitos - onde os conceitos nominais, gerais e verbais são extraídos; e, finalmente, a geração da regra SBVR pelo componente NPL. Essas regras de negócio baseadas na notação SBVR são utilizados em um *framework* para exportação de modelos UML. As regras são posteriormente mapeadas para artefatos de software como consultas SQL.

A infraestrutura utilizada é baseada na plataforma *Eclipse Modeling Framework* - EMF, utilizando a linguagem de transformação *Atlas Transforming Language* – ATL que é utilizada para transformar o modelo EMF serializado em XMI, e em scripts SQL. Entretanto essa proposta só aborda parte do problema de criar sistemas de informação baseados em linguagem natural, pois não mostra como outras partes do sistema serão geradas, e.g. modelos de navegação e interação com o usuário e sua integração com as regras de negócio. Além disto, não são discutidos de forma aprofundada temas como a utilização de outros idiomas que não o inglês na definição do vocabulário e regras de negócio.

Bajwa possui uma série de trabalhos (BAJWA; BORDBAR; LEE, 2010; BAJWA; LEE; BORDBAR, 2011; BAJWA; BORDBAR; LEE, 2011; BAJWA, 2012) mostrando uma abordagem para transformar expressões definidas por meio de linguagem natural em notação SBVR, e regras de restrições SBVR para a linguagem OCL – *Object Constraint Language* (OMG, 2014). A principal característica dessa abordagem é a utilização de linguagem natural na definição dos conceitos e seu posterior processamento e transformação em SBVR, chamado de NLP (*Natural Language Processing*).

Conforme o autor, os principais desafios dessa abordagem são: a análise das restrições da linguagem natural para gerar uma representação lógica do vocabulário SBVR e o mapeamento da representação lógica para regras em SBVR, além da representação em outros idiomas. A análise das restrições da linguagem natural envolve uma sequência de passos: pré-processamento, análise sintática e análise semântica. A Figura 7 mostra o *framework* utilizado pelo autor (BAJWA, 2012) para a análise das restrições da linguagem natural.

Figura 7 - Bajwa - Framework para análise de restrições da linguagem natural



Fonte: Bajwa (2012).

São duas camadas definidas no *framework*: *Logical Layer* e *User Interface Layer*. A primeira realiza tarefas como pré-processamento, análises sintática e semânticas, enquanto que a segunda é responsável por lidar com a entrada e saída de dados. As principais tarefas na fase de pré-processamento estão na divisão das frases, separação das palavras e lematização - desconsideração do gênero e número. Para a maioria destas funcionalidades, o componente – *"Stanford POS Tagger"* se encarrega de dividir e categorizar as partes do discurso – POS (do inglês *Part of Speech*). Por exemplo, na análise sintática, o *POS Tagger* separa as palavras e dá a elas uma sigla definindo que parte do discurso ela representa como substantivo (NN), plural (NNS), verbo (VB), ou outro elemento gramatical. Entretanto, estes componentes não são a prova de falhas e podem marcar elementos inconsistentemente e produzir uma árvore inválida. Desta forma, a abordagem do autor é utilizar um modelo UML como validador do resultado do parser e de decisão quando houver dúvida sobre a verdadeira função de uma palavra no discurso. Assim, todas as *tags* geradas pelo *POS Tagger* são mapeadas para classes UML. A Figura 8 mostra como este mapeamento é feito para termos do inglês.

Figura 8 - Bajwa - Mapeamento de elementos do inglês para elementos de um modelo de classes UML

UML class model elements		English language elements
Class names	→	Common Nouns
Object names	→	Proper Nouns
Attribute names	→	Generative Nouns, Adjectives
Method names	→	Action Verbs
Associations	→	Action Verbs

Fonte: Bajwa (2012).

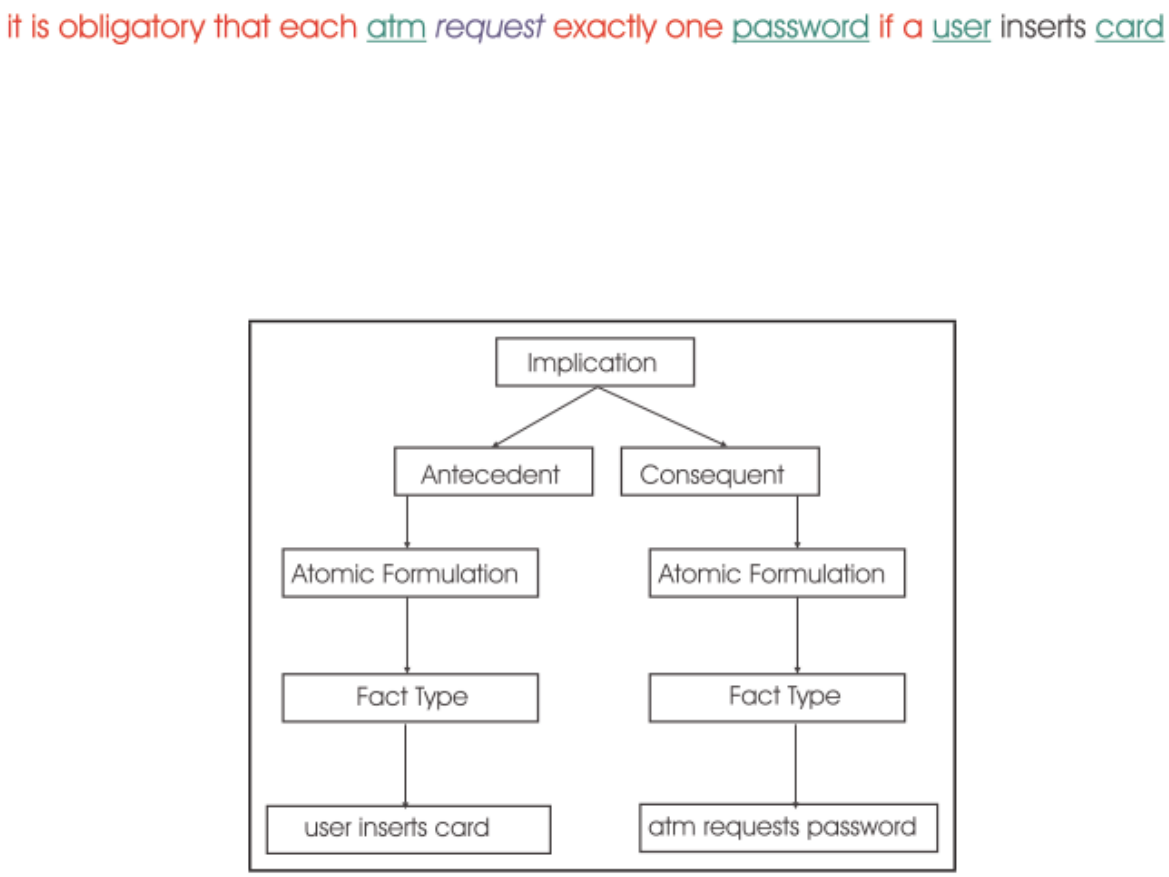
Essa abordagem é mais abrangente do que a tratada neste trabalho, no que diz respeito ao processamento da linguagem natural. Enquanto essa abordagem faz um mapeamento de NL para SBVR ou OCL, neste trabalho, parte-se do processamento de uma regra já definida em SBVR. Isto facilita em muito o processamento, pois restrições de linguagens naturais não precisam ser consideradas. Como em outras abordagens, um potencial problema dessa abordagem é a internacionalização, visto que o reconhecimento das partes do discurso é dependente do idioma original. Na abordagem adotada nesta dissertação, o idioma original, no qual estão sendo descritas as regras e o vocabulário, não interfere no reconhecimento do editor e na geração do modelo. Além disto, outros aspectos de um sistema de informação tradicional como navegação, *layout* e visualização não são abordados.

Em Raj, Prabhakar e Hendryx (2008) é proposto um modelo de transformação de vocabulários e regras SBVR em modelos e diagramas UML. O modelo utiliza uma abordagem MDA para mesclar um modelo SBVR com outros modelos e automatizar as regras de negócio. Nessa abordagem, a verificação da regra ocorre através da automação. De acordo com essa pesquisa, é necessário que uma política de verificação de regras de negócio no sistema de informação seja aplicada a cada regra para a definição de como, quando e em qual lugar no sistema a regra será verificada e aplicada. Esse trabalho se dedica a definir quais são essas políticas de verificação e aplicação de regras nos sistemas de informação. Desta forma, uma metodologia é apresentada para transformar essas regras de negócio e seu vocabulário em diagramas UML de Classes, Atividades e Sequência.

Essa abordagem utiliza as formulações lógicas da SBVR para definir as regras que serão utilizadas nos diagramas e suas relações. As regras são categorizadas por um algum atributo personalizado, como por exemplo, o chamado de *"IT Support"* que pode assumir os valores *"automated"* ou *"supported"*. Apenas as regras marcadas como "*automated*" serão utilizadas, por exemplo, para a criação do diagrama de atividades. As regras automatizáveis

são analisadas como construções "*if-then*", que no padrão SBVR correspondem a implicações. Os antecedentes e consequências das implicações são mapeadas para pré-condições e atividades respectivamente. A Figura 9 mostra esta correlação.

Figura 9 - Raj - Formulação Lógica de Regras *If-Then-Else*



Fonte: Raj, Prabhakar e Hendryx (2008).

Somente os verbos transitivos serão considerados como atividades, como na proposição *"user inserts card"*. Verbos não estruturais como *"has"* e *"is"* não são considerados no diagrama de atividades pois representam estrutura. No diagrama de atividades o mapeamento segue as seguintes regras: um nó pré-definido com o nome "start" é criado; todos os conceitos verbais (fatos) com verbos transitivos serão considerados como atividades; as restrições de transições serão tipos de fatos que são booleanos (i.e.: "*It is obligatory that each atm requests one password if a user inserts a card" – insert_card* é um atributo booleano de guarda da atividade); ações serão operações que serão executadas durante a transição entre as atividades (i.e.: se o *"user inserts card"* é verdadeiro, a transição executará *"request_password()"*).

Os diagramas de sequência são definidos através do sequenciamento das formulações lógicas das regras verificando a relação entre os antecedentes e consequências das regras conforme mencionado anteriormente. As regras para definição do diagrama de classes se baseiam no mapeamento de termos que começam com letras minúsculas para nome de classes, *Individual Concepts* são mapeados para instâncias de objetos. Fatos binários com o verbo "has" serão mapeados como classe-propriedade. A hierarquia de classes será determinada pelos verbos *"specializes/generalizes"*. As associações são determinadas pelos conceitos verbais e a cardinalidade pelas regras que incluem tal informação (i.e.: *"It is necessary that each card uses at most one password" – card* tem um *password*).

Jesus (2013) apresenta uma abordagem para integração de regras de negócio em sistemas de informação baseado em eventos. Para a verificação de uma regra de negócio em um sistema de informação, um dos aspectos que tem que ser analisado são os eventos que indicam que uma regra deve ser verificada. Como o padrão SBVR não trata explicitamente estes eventos, esse trabalho propõe uma abordagem de checagem dos eventos que geram alguma mudança às instâncias dos elementos associados às regras de negócio. Para identificar estes eventos, o trabalho sugere o mapeamento do modelo de representação semântica do SBVR para um formalismo de eventos baseado em álgebra de processos π – *calculus* (MILNER, 1999). O autor considera que a extração e a definição da relação de cada evento com os elementos e variáveis das regras de negócio, são implementados em processos paralelos e concorrentes. As regras estarão em execução em um motor de regras de negócio, e necessitam se comunicar com outros elementos da arquitetura da aplicação, fazendo-se necessário um protocolo de comunicação que, na proposta, é baseado no π – *calculus.*

Esse trabalho mostra uma abordagem para o mapeamento dos eventos envolvidos na verificação de regras de negócio, mas deixa questões abertas: a verificação da regra só ocorre após sua violação; a tradução de regras para o padrão SBVR em outros idiomas, além do inglês; a transformação dos elementos semânticos e mapeamento de eventos de forma automatizada – já que utiliza um processo manual. Além disto, a abordagem sugerida por esse trabalho pode criar uma maior complexidade na implementação das soluções, por pressupor a utilização de motores de regras de negócio, o que requer a construção de um adaptador específico para cada implementação de motor de regras e o compartilhamento da base de dados para a intercomunicação destes componentes da arquitetura.

3.1.2 Ferramentas SBVR

O projeto OPAALS (RAIMUND et al., 2008) foi um dos trabalhos pioneiros que discute a possibilidade da utilização de modelos SBVR em sistemas de informação. O projeto visou estudar várias abordagens e o referencial teórico envolvendo o uso do SBVR em sistemas de informação. Apesar da extensa pesquisa na área de linguística e linguagem natural em sistemas de informação, esta parte do projeto não parece ter continuado após a criação do editor SBVR denominado – *SBeaVer*. A abordagem usa linguagem natural para a definição de vocabulário e regras SBVR e do *workflow* dos sistemas de informação por meio dos relacionamentos entre as regras detectadas. Através de uma série de transformações, as regras SBVR são analisadas e seus relacionamentos e interdependências inferidos. A partir dessas informações um modelo de *workflow* é produzido e uma aplicação é produzida através de novas transformações de modelo-para-código. Outras versões do mesmo projeto foram produzidas para demonstrar a viabilidade de reconfiguração de *workflows* utilizando regras definidas em SBVR.

VetisTool (NEMURAITE ; SKERSYS, 2010) foi um esforço subsequente ao projeto descrito anteriormente, para prover um editor SBVR para regras e vocabulário de negócio. Ele foi baseado no *SBeaVer*, e adicionou algumas funcionalidades como a opção de exportar SBVR 1.0 *XMI Schema* e integração com a ferramenta UML *MagicDraw* com regras representadas através de restrições OCL. A ferramenta usa traços "-" para separar conceitos verbais, para facilitar o processamento e reconhecimento destes conceitos. Por exemplo, o seguinte conceito deve ser representado com traços para definir o conceito verbal que um estudante deve estar registrado para um curso: *"student must-be-registered-for curse"*. Esta limitação impõe um vocabulário e regras de negócio que se assemelham a identificadores utilizados em linguagens de programação, ou marcação, por utilizarem um hífen para conectar os conceitos e verbos da regra.

O trabalho de Marinos, Gazzard e Krause (MARINOS; GAZZARD; KRAUSE, 2011) mostrou um editor SBVR com capacidade de auto-complemento de conceitos e marcação de sintaxe. A ferramenta adiciona suporte a mais conceitos que as antecessoras (*VetisTool* e *SBeaVer*), mas não permite outras representações senão o SBVR-SE. Adicionalmente, como os autores enfatizam, a ferramenta não consegue marcar e identificar conceitos nas definições de atributos de conceitos do SBVR como *"Synonym"* ou *"Definition"*.

RuleXpress (RULEARTS, 2008) é uma ferramenta que permite criar vocabulários de negócio, juntamente com regras, no formato *RuleSpeak* e também pode ser utilizado para criar

vocabulários e regras SBVR. A ferramenta possui facilidades de visualização e manutenção de regras, mas o suporte a idiomas diferentes do inglês só é possível com um esforço de tradução da ferramenta. Além disto não há suporte ao modelo de interação com o usuário ou transformações de modelos.

3.1.3 SBVR e MDA

Pode-se identificar várias abordagens na literatura visando adaptar regras de negócio definidas em linguagem natural à notação SBVR para posterior transformação ou execução. Njonko e EL ABED (2012) propõem uma metodologia para transformação de regras de negócio, expressas em linguagem natural, utilizando uma abordagem generativa baseada nos conceitos da MDA. Esta proposta visa a criação de modelos executáveis e representações (UML, SQL, etc.) a partir de linguagem natural e regras de negócio, posteriormente transformados em formulações baseadas na notação SBVR para sua utilização em um *framework* de exportação de modelos. Bajwa (BAJWA; LEE; BORDBAR, 2011) mostra uma abordagem para transformar expressões definidas em linguagem natural, em notação SBVR para posterior geração de restrições do tipo OCL (*Object Constraint Language*), mostrando a implementação uma ferramenta denominada *OCL-Builder* para este fim. Em Linehan (LINEHAN, 2006), regras de negócio expressas utilizando SBVR são expostas a técnicas de Desenvolvimento Baseado em Transformação de Regras de Negócio – MDBT (*Model-Driven Business Transformation*) utilizando a abordagem MDA para a criação de modelos UML com restrições do tipo OCL.

Esses trabalhos dão uma visão do que pode ser alcançado na adoção de técnicas MDD para a transformação de regras de negócio, expressas em linguagem natural, para modelos formalmente enunciados de acordo com as semânticas definidas no padrão SBVR. Em sua maioria, definem em sua arquitetura, previamente ao processamento do modelo SBVR, um processador de linguagem natural. Isto tende a tornar a solução mais complexa de ser implementada, uma vez que utilizam a premissa de que a linguagem natural do negócio, em sua forma mais direta e, portanto, sem nenhum formalismo, será utilizada na definição das regras de negócio. Outras abordagens utilizam o modelo SBVR como fonte de entrada das regras de negócio e transformá-las em um modelo intermediário, como o OCL, e posterior transformação em artefatos de software.

3.2 MDD, WEB E INTERAÇÃO COM O USUÁRIO

Diversos trabalhos já foram propostos para modelagem e especificação de sistemas *Web* e interação com o usuário. *WebML* (CERI et al. 2000), *WebSA* (MELIA et al. 2003), *FrameWeb* (ESTÊVÃO; FALBO; GUIZZARDI, 2007) e *WebDSL* (VISSER, 2007) são exemplos. Este trabalho parte destas abordagens utilizando explicitamente a forma textual e declarativa de descrever os diversos aspectos envolvidos neste tipo de aplicação, porém se baseia no padrão IFML.

A *WebML* (CERI et al. 2000) também propõe um modelo que utiliza outros trabalhos como base para a definição de aplicações *Web* utilizando UML e XML nas dimensões de dados (modelo estrutural), páginas (modelo de composição), topologia dos *links* entre as páginas (modelo de navegação), os requisitos de *layout* e gráficos para as páginas (modelo de apresentação) e as necessidades de personalização para a entrega de conteúdo para o usuário (modelo de personalização). Os autores inclusive integram o comitê de manutenção da IFML. A *WebSA* (MELIA et al. 2003) propõe a extensão e enriquecimento destes modelos utilizando a abordagem MDA (OMG, 2014) para abranger aspectos arquiteturais de aplicações *Web*.

Distante et al. (2007) descreve uma abordagem utilizando o framework UWA (*Ubiquitous Web Application*), com o uso do padrão arquitetural MVC (*Model View Controller*), a tecnologia JSF – *JavaServer Faces* e o processo de desenvolvimento dirigido a modelos – MDD (*Model Driven Development*). Apesar de haver semelhanças entre a proposta desta dissertação na utilização da metodologia MDD e o padrão arquitetural MVC, tal abordagem está focada em uma tecnologia específica de aplicações *Web*, na plataforma Java, para responder aos requisitos de apresentação do JSF.

A proposta da linguagem apresentada por esse trabalho, abstrai um modelo de domínio de alto nível para especificação de aplicações *Web*, tendo como base o MVC, porém não especifica qual tecnologia de implementação será utilizada. Assim, o arquiteto de software, analista ou designer, pode escolher implementar os templates arquiteturais da solução para que gerem artefatos coerentes com quaisquer soluções tecnológicas e *frameworks* existentes para aplicações *Web* – JSF, *Struts*, *Spring*, *Spring Web*, .NET ou quaisquer outras.

Kroiss, Koch e Knapp (2009) propõem um método de integração e validação da integração entre os diversos modelos (navegação, aspectos, conteúdo, lógica de negócio e apresentação) para sistemas *Web* chamado UWE4JSF, que é baseado na metodologia de *Web Engineering* com UML (UWE). Este método descreve seus modelos baseados em UML

Profiles e utiliza um processo MDD para a geração da aplicação final baseado em templates para Java EE e JSF.

A maioria das propostas similares avaliadas nesta dissertação baseia-se principalmente no uso de UML – Unified Modeling Language (uma DSL visual) ou XML para a especificação dos modelos. Outras propostas mais recentes, incluindo a descrita nesta dissertação, se baseiam em algum tipo de DSL textual para a especificação dos conceitos de aplicações *Web*. Por exemplo, Visser (VISSER, 2007) apresenta a *WebDSL*, na qual define uma linguagem expressiva para definir aplicações *Web*, abrangendo seus aspectos estruturais, de navegação e transacionais. Entretanto, não utiliza padrões para as definições de regras de negócio ou modelo de interação com o usuário.

3.3 CONSIDERAÇÕES FINAIS

Comparativamente com a proposta deste trabalho, Njonko e El Abed (2012) apresentam uma abordagem interessante para processamento de modelos UML e MDA para SBVR e SQL. Entretanto, como essa abordagem foi criada especificamente para criar agentes capazes de serem utilizados por um *framework* específico (DEMS), só abordam parte do problema de criação de sistemas de informação com regras de negócio definidas em SBVR, pois não mostram como outras partes do sistema serão gerados, como modelos de navegação e interação com o usuário, além de sua validação perante as regras de negócio, cujos aspectos são contemplados neste trabalho. Além disto, a possibilidade da definição de regras em múltiplos idiomas é apresentada como uma característica que o modelo SBVR dá suporte, sem mostrar como isso seria implementado.

A abordagem proposta por Bajwa (BAJWA, 2012) procura mapear o processamento da linguagem natural para SBVR e restrições OCL. A abordagem sugerida por esta dissertação, parte do processamento do vocabulário e regras em SBVR, o que facilita a implementação da proposta, visto que não há o problema de interpretação direta da linguagem natural na qual estão sendo descritas as regras e o vocabulário. Além disto, outros aspectos de um sistema de informação tradicional como navegação, *layout* e visualização também não são abordados nesse trabalho.

Jesus (2013) fornece uma abordagem de mapeamento dos eventos que devem resultar em uma verificação de regras de negócio definidas em SBVR, utilizando um modelo baseado no formalismo do π – *calculus*. Esse trabalho preconiza o uso deste modelo para dar suporte a um mecanismo de comunicação paralelo entre as regras em execução em motores de regras de

negócio e o resto dos elementos da arquitetura. Entretanto, não aborda os problemas de suporte a múltiplos idiomas na formalização das regras de negócio em SBVR, ou de intercomunicação com o modelo de interação com o usuário, somente com elementos do modelo estrutural definidos pelo vocabulário e regras de negócio definidas em SBVR. Além disto, como a abordagem pressupõe o uso de motores de regras, é necessária a implementação de um adaptador específico para cada motor de regra utilizado.

Outros estudos como o de Raj, Prabhakar e Hendryx (2008) e Linehan (2006), apresentam uma metodologia de transformar regras e vocabulário de negócio modelos intermediários como diagramas UML de Classes, Atividades e Sequência e restrições OCL. Tais abordagens, e outras que englobam ferramental para lidar com a SBVR, diferem da apresentada neste trabalho, que abrange a geração final da aplicação, além da integração dos modelos de interação com o usuário e o de regras de negócio. Outrossim, mostra como questões relacionadas ao suporte de múltiplos idiomas pode ser resolvida com a solução. Esses trabalhos apresentam problemas na representação das regras de forma textual, por exemplo, na solução apresentada por Raj, Prabhakar e Hendryx (2008), a representação textual de conceitos verbais das regras devem conter um "*underscore*" separando conceitos com mais de uma palavra como como em "*atm is_role_of machine*".

No âmbito de aplicações *Web* e aplicação de técnicas MDD, trabalhos como os de Kroiss, Koch e Knapp (2009), Ceri et al. (2000), Distante et al. (2007), Visser (2007) desenvolveram uma solução semelhante ao deste trabalho, mas não focam em como resolver problemas relacionados as regras de negócio em linguagem natural, utilizando SBVR, e sua integração com o modelo de interação com o usuário e o desenvolvimento baseado em modelos. Essas abordagens utilizam modelos não padronizados para a descrição e incorporação de regras de negócio e interação com o usuário no modelo conceitual dos sistemas de informação dificultando a interoperabilidade. Também se preocupam com a geração de aplicações *Web* sem abordar o problema da possível perda de semântica na interpretação das regras de negócio, originalmente definidas por analistas de negócio em linguagem natural. A integração com código manual não é mencionado em nenhum destes trabalhos.

Conforme discutido, nenhuma das propostas analisadas consegue tratar explicitamente regras de negócio, em um processo de desenvolvimento dirigido a modelos, por meio de uma linguagem controlada, multilíngue, e que proveja um modelo de interação com o usuário associado. Portanto, a partir das características identificadas nos trabalhos correlatos, pode-se definir os seguintes requisitos para a solução apresentada nesta dissertação:

a) permitir a definição de regras e o vocabulário de negócio através de uma linguagem controlada como a SBVR;
b) permitir a definição de um modelo estrutural e de interação com o usuário;
c) permitir que as regras de negócio e o modelo de interação estejam integrados e validados entre si;
d) aderente ao desenvolvimento dirigido a modelos;
e) permitir a inclusão de código gerado manualmente;
f) permitir a geração de aplicações *Web*;
g) suporte a múltiplos idiomas;
h) não necessitar interpolação de caracteres especiais nas regras de negócio SBVR para seu processamento;
i) utilização de padrões para os elementos do modelo conceitual nas regras de negócio e modelo de interação com o usuário.

Nos Quadros 1 e 2 são mostrados os requisitos analisados e se são atendidos por cada uma das soluções analisadas.

Quadro 1 – Comparação entre as abordagens analisadas (a)

Característica / Solução	***SBVR-TO-EXE***	**Bajwa**	**Raj, Prabhakar, e Hendryx**	**Jesus**	**OPAALS**	***VetisTool***	**Marinos SBVR Editor**
Processamento de Regras em SBVR ou outra linguagem controlada	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Definição dos modelos estruturais e de interação com o usuário	Não	Não	Não	Não	Parcial	Não	Não
Integração e validação entre os modelos de interação com o usuário e regras de negócio	Não	Não	Não	Não	Não	Não	Não
Aderente ao Desenvolvimento Dirigido a Modelos	Parcial	Parcial	Parcial	Não	Parcial	Não	Não
Integra código manual	Não	Não	Não	Não	Parcial	Não	Não
Geração de Aplicações *Web*	Sim	Não	Não	Não	Sim	Não	Não
Suporte a múltiplos idiomas na definição de regras de negócio em SBVR	Não	Não	Não	Não	Não	Não	Não
Necessidade de caracteres especiais nas regras de negócio SBVR	Não	Não	Sim	Não	Sim	Sim	Sim
Uso de padrões para o modelo conceitual	Sim	Sim	Sim	Sim	Sim	Sim	Sim

Fonte: Autoria própria.

Quadro 2 - Comparação entre as abordagens analisadas (b)

Característica / Solução	*RuleXpress*	*OCL-Builder*	*Linehan*	*WebML*	*WebSA*	UWA	UWE4JSF	*WebDSL*
Processamento de Regras em SBVR ou outra linguagem controlada	Sim	Sim	Sim	Não	Não	Não	Não	Não
Definição dos modelos estruturais e de interação com o usuário	Não	Não	Não	Sim	Sim	Sim	Sim	Sim
Integração e validação entre os modelos de interação com o usuário e regras de negócio	Não	Não	Não	Sim	Parcial	Sim	Não	Sim
Aderente ao Desenvolvimento Dirigido a Modelos	Não	Parcial	Não	Sim	Sim	Sim	Sim	Sim
Integra código manual	Não	Não	Não	Não	Não	Não	Não	Sim
Geração de Aplicações *Web*	Não	Não		Sim	Sim	Sim	Sim	
Suporte a múltiplos idiomas na definição de regras de negócio em SBVR	Parcial	Não	Não	Não	Não	Não	Não	Não
Necessidade de caracteres especiais nas regras de negócio SBVR	Não	Não	Não	Não	Não	Sim	Não	Não
Uso de padrões para o modelo conceitual	Sim	Sim	Sim	Não	Não	Não	Sim	Não

Fonte: Autoria própria.

4 PLATAFORMA *ENGEN*

O objetivo deste trabalho é apresentar uma solução para o desenvolvimento de sistemas empresariais, baseados na aplicação de técnicas MDSD (*Model Driven Software Development*), com utilização de regras de negócio, para permitir uma melhor sinergia e integração entre equipes de tecnologia da informação e do negócio. Para isto, foram propostas ferramentas de apoio à especificação, modelagem, transformação entre modelos, geração de código e uma arquitetura que permitem a criação de artefatos de software utilizando técnicas MDSD para a construção de sistemas empresariais. O conjunto dos componentes que fazem parte da solução recebe o nome de *Plataforma Engen* (***EN****terprise System* ***G****enerator* ***EN****gine – Motor de Geração de Aplicações Empresariais*). Neste capítulo são discutidos os componentes, técnicas, ferramentas e arquitetura propostas. Sua organização ocorre da seguinte forma: a Seção 4.1 enumera os requisitos e premissas definidos para a solução; a Seção 4.2 apresenta a arquitetura da Plataforma *Engen*, das ferramentas e componentes produzidos; na Seção 4.3 são mostrados os elementos do processo de desenvolvimento sugerido; na Seção 4.4 a *EngenDSL* é descrita, juntamente com seus elementos; na Seção 4.5 a *EngenSBVR* é detalhada; a Seção 4.6 mostra os Editores da Plataforma e a Seção 4.7 trata das transformações entre modelos e código.

Para possibilitar um uso efetivo de técnicas MDSD (*Model Driven Software Development*) com regras de negócio, é necessário que se utilize ferramentas de apoio, arquitetura e processos desenvolvidos para este fim. Desta forma, foram considerados no desenvolvimento deste trabalho, algumas premissas e requisitos que são discutidas na próxima seção.

4.1 PREMISSAS E REQUISITOS DO MODELO ARQUITETURAL

A abordagem MDSD nesta proposta inclui sugestões de como utilizar o código gerado por meio das transformações, e incorporá-lo à aplicação final, opcionalmente em conjunto com o código criado manualmente (ver Seção 4.7 para mais detalhes).

A abordagem desta dissertação utilizou as seguintes premissas para a implementação:

a) o modelo conceitual poderá ser construído utilizando uma linguagem específica de domínio textual criada para a modelagem do vocabulário e regras de negócio de entrada de dados, sem estar atrelada a nenhum idioma específico, juntamente com uma

linguagem específica de domínio textual criada para a modelagem da interação com o usuário, ou qualquer outro editor de texto disponível;

b) deve ser prático de manipular o modelo conceitual na solução proposta;

c) as transformações dos modelos de domínio para os modelos intermediários e código-fonte podem ser executadas em qualquer fase do ciclo de desenvolvimento;

d) a aplicação final deve utilizar uma estrutura padronizada, capturada pelos *templates* arquiteturais utilizados;

e) o número de instruções deve ser o menor possível, sem prejuízo da expressão das semânticas necessárias para a modelagem dos conceitos de interação em um sistema de informação;

f) as regras de transformação estão escritas em bibliotecas de código escritas nativamente, ou que possam ser transformadas na linguagem JAVA, e estão disponíveis para serem instanciadas pelas ferramentas de transformação, respeitando as interfaces definidas pelos componentes de transformação de M2M e M2T;

g) as linguagens devem evitar o uso de construções típicas de linguagens genéricas de alto nível como condicionais, laços (loops), definição ou chamada métodos com parâmetros;

h) linguagens e frameworks open-source, além de padrões publicados por instituições reconhecidas, serão ser utilizados sempre que possível para a arquitetura da solução proposta;

A plataforma proposta foi concebida de acordo com estas restrições, com o intuito de modelar a interação com o usuário e regras de negócio em um sistema de informação baseado em técnicas MDSD. Os seguintes requisitos foram definidos a partir daqueles discutidos na Seção 3.3 e utilizados para nortear a elaboração da plataforma:

a) devem ser criadas linguagens textuais que permitam a modelagem do modelo conceitual, sob os aspectos de **regras de negócio**, semelhante à linguagem utilizada pelos especialistas do negócio, e **interação com o usuário**;

b) a linguagem para especificação de regras de negócio deve suportar qualquer idioma;

c) as linguagens devem poder ser validadas entre si, de modo a dar suporte à validação das regras de negócio de entrada de dados frente ao modelo de interação com o usuário;

d) a solução deve permitir, além da modelagem da interação com o usuário, a estruturação de elementos visuais de um sistema de informação;

e) priorizar construções declarativas em vez de procedurais na definição dos conceitos da aplicação;

f) ferramentas devem ser disponibilizadas para a aplicação de técnicas MDSD aos elementos do modelo conceitual do sistema, permitindo que o processo possa ser repetível;

g) permitir a modelagem da interação com o usuário em conformidade com o conceito "*view"* do modelo arquitetural MVC – Model View *Controller* (GAMMA et al. 1995);

h) as abstrações devem ser definidas de forma a contemplar o maior número de arquiteturas possíveis, sem se comprometer com nenhuma especificamente;

i) as linguagens não precisam ser expressivas suficientemente para definir todos os conceitos da aplicação, ou seja, outras informações podem ser necessárias para a definição completa das funcionalidades da aplicação;

j) os componentes definidos externamente à linguagem devem poder ser integradas ao sistema em desenvolvimento, sem prejuízo da semântica e sintaxe da DSL.

k) prover editores para as linguagens capazes de fornecer facilidades para a construção, armazenamento, visualização, recuperação e referência às regras de negócio e que permitam a definição de conceitos de interação com o usuário, incluindo *layouts* e componentes de tela e sua integração com as regras, bem como facilidades de usabilidade tais como validação sintática e semântica ao digitar, propostas automáticas para o complemento de regras e formatação visual;

l) a solução não deve prescindir destes editores para a manipulação destes modelos;

m) definir um *parser* para tipos específicos de formulações lógicas da SBVR para permitir que regras de negócio possam ser interpretadas;

n) definir um visualizador do resultado do *parser* para as formulações lógicas das regras;

o) criar um modelo intermediário capaz de carregar os conceitos definidos nas linguagens para posterior geração de código;

p) criar um Motor de Transformação capaz de transformar o modelo EMF gerado pelas linguagens no modelo intermediário;

q) criar um Motor e Templates Arquiteturais em Java/EE que permitem a transformação do modelo intermediário para o código final da aplicação, integrados ao editor que possibilite e apoie o processo de geração de código a partir dos modelos definidos pelo analista, baseados no metamodelo das linguagens;

r) criar ferramentas, componentes e bibliotecas que funcionem independente do editor, para as mesmas funcionalidades de geração de código, permitindo que o processo de desenvolvimento não esteja associado a uma IDE (*Integrated Development Environment*) específica.

As Seções 4.1.1 e 4.1.2, discutem as decisões tomadas a respeito das premissas e requisitos elencados para a solução proposta.

4.1.1 Representação Textual das DSLs

Alguns trabalhos (CERI; FRATERNALI; BONGIO, 2000) provêm um meio gráfico para modelar aplicações *Web*. Apesar de que os modelos visuais podem proporcionar um rápido entendimento da solução, desenvolvedores de software estão acostumados a utilizar ferramentas de edição textual de código-fonte. A abordagem textual de descrição dos modelos dos sistemas, apresentada neste trabalho, prioriza a representação textual dos modelos, dentre outros motivos, por esta forma proporcionar a maneira mais direta de modelagem associada às atividades de desenvolvimento software. Linguagens específicas de domínio (DSL) textuais possuem diversas vantagens, como a integração com as ferramentas existentes para desenvolvedores, facilidade na evolução da linguagem, integração com ferramentas de versionamento de código e facilidade da leitura e pesquisa do conteúdo do modelo, além da facilidade de armazenamento proporcionado pela representação textual.

O versionamento do código é uma atividade importante durante o desenvolvimento de um projeto de software pois evita que seja perdida a memória do que foi feito durante as sucessivas iterações do projeto. Particularmente o uso de ferramentas de versionamento de código é substancialmente dificultado em formas binárias de representação dos modelos utilizando diagramas UML (KÖVESDÁN; ASZTALOS; LENGYEL, 2014). Mesmo que tais diagramas sejam serializados em formato XML, o tratamento de conflito é uma tarefa difícil, quando modelos UML são atualizados por diferentes pessoas, trabalhando em paralelo, e o resultado da serialização do modelo é versionado (BROSCH et al., 2011).

O formato textual apresentado neste trabalho permite que as técnicas e ferramentas já preconizadas, e regularmente utilizadas por desenvolvedores, para versionamento de código como CVS (CVS, 2015) e GIT (GIT, 2015), e de tratamento de conflitos como KDIFF3 (EIBL, 2014), possam ser aplicadas aos modelos descritos nas linguagens sem o uso de nenhuma técnica adicional. Além disto, devido ao seu formato textual, os especialistas podem

manipular os documentos de regras diretamente em qualquer editor de texto. Desta forma, tanto os requisitos de definição de regras de negócio em uma linguagem semelhante à utilizada pelos especialistas do negócio, quanto os requisitos de armazenamento e manutenção são atendidos com a utilização de uma DSL textual.

4.1.2 Framework XText

Para a construção dos editores e linguagens específicas de domínio, propostas nesta dissertação, o framework escolhido foi o Projeto *XText* (XTEXT, 2016). A escolha do *XText* deve-se à facilidade de criação de linguagens específicas de domínio e editores, baseados na plataforma *Eclipse,* manutenção das linguagens e o fato de ser um projeto de código aberto e livre, conforme os requisitos e premissas da plataforma elencados na introdução deste capítulo. Como a atualização dos editores é automatizada, e baseada em geração de código conforme o metamodelo da linguagem evolui, o esforço envolvido na manutenção das linguagens é diminuído, se comparado com outras abordagens analisadas como o *AntLR* (ANTLR, 2015). O *XText Framework* (XTEXT, 2016), permite a criação de um metamodelo definido pela linguagem, que por sua vez é baseado no EMF. Ao se definir uma linguagem utilizando o *XText,* o *framework* provê vários elementos que fornecem a base para a criação de editores baseados no *Eclipse* e EMF:

a) um editor padrão para a linguagem criada, com serviços de complemento de termos e visualização básica do modelo;
b) infraestrutura de um *lexer* e um *parser* para a linguagem, que são componentes que juntos são capazes de converter sequência de caracteres em um modelo semântico, bem como serviços de *linking,* que é a descoberta de referências textuais entre os modelos;
c) uma base de serviços para a personalização do editor, complemento de termos, visualização do modelo, validação de sintaxe e semântica dentre outros.

Apesar do *XText* permitir a criação automática de um editor básico para as linguagens, uma quantidade considerável de trabalho é necessária para que os editores e linguagens possam ser utilizados de forma eficaz por usuários. A construção semântica do modelo, validações e verificações devem ser personalizadas por linguagem, de forma a tornar o modelo passível de processamento.

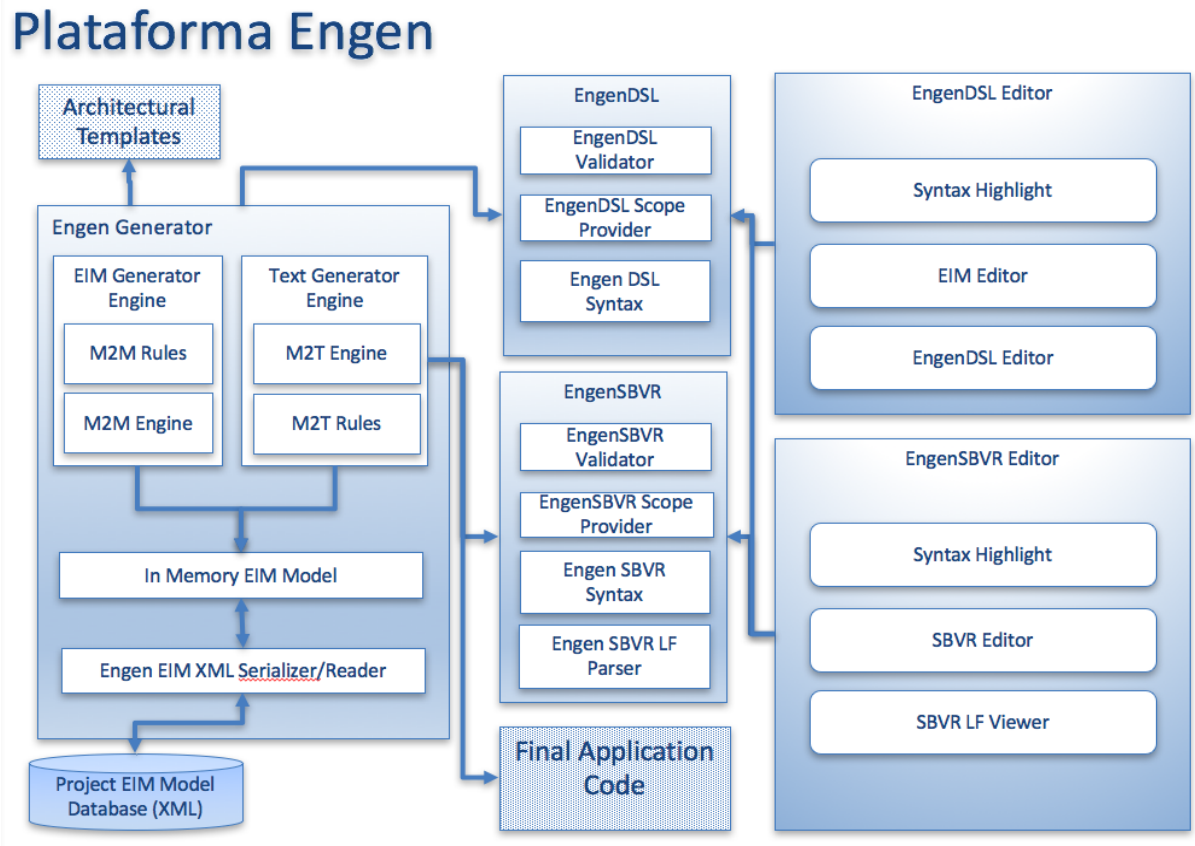
A *Plataforma Engen*, seus componentes e relacionamentos são discutidos nas seções seguintes.

4.2 PLATAFORMA ENGEN DE DESENVOLVIMENTO

Para permitir que DSL textuais fossem utilizadas, juntamente com editores e facilitadores para a modelagem dos sistemas de informações empresariais que integrem a especificação do modelo de interação com o usuário baseado na IFML, e de regras de negócio baseadas no SBVR, é proposta a *Plataforma Engen*, composta pelos componentes ilustrados na Figura 10. Os componentes visuais, e as linguagens específicas de domínio, tem como base a plataforma *Eclipse* (ECLIPSE, 2014), que é uma plataforma difundida na comunidade de software e que possui uma base para o desenvolvimento de editores e visualizadores de modelos, principalmente baseados no projeto *EMF – Eclipse Modeling Framework*, conforme discutido no Capítulo 2.

Destarte, foi necessária a implementação de diversos componentes na *Plataforma Engen*, de forma a permitir que seus objetivos fossem alcançados. Além das linguagens específicas de domínio, um motor de transformação entre modelos baseado em técnicas MDSD e *templates arquiteturais* para a geração de código foram construídos, que funcionam independente dos editores, e juntos compõem a *Plataforma Engen*. Tais componentes são mostrados esquematicamente nos elementos da Figura 10, e são detalhados nas seções subsequentes.

Figura 10 – Editores e Componentes



Fonte: Autoria própria.

A arquitetura da solução envolve diversos componentes, a saber:

a) ***EngenGenerator*** – conjunto de componentes (motores de transformação) encarregados das transformações modelo-para-modelo e modelo-para-texto (*M2M Engine* e *M2T Engine* respectivamente). Para cada tipo de transformação existem regras associadas (M2M *Rules* e M2T *Rules*) e templates para transformação do modelo intermediário (*Engen Intermediate Model* - EIM) em código, além de um conjunto de testes para verificação da correção das regras de geração;

b) ***EngenDSL*** – uma linguagem específica de domínio criada para a modelagem da interação com o usuário e definição de *layouts* para sistemas de informação. É composta pela sua sintaxe definida em um arquivo do *XText Framework* próprio para este fim, de validadores semânticos que são acionados pelo editor, ou pelo *EngenGenerator*, e o provedor de escopo, que consiste em um analisador que determina quais são os elementos referenciáveis válidos em um determinado contexto;

c) ***EngenDSL Editor*** – um editor baseado no *XText Framework* para facilitar a manipulação da *EngenDSL* e do EIM, e é composto de componentes relacionados com a tarefa de edição dos arquivos de código fonte na linguagem *EngenDSL*, como

componentes para coloração e formatação da fonte utilizada no editor, componentes para visualização do modelo, dentre outros;

d) ***Engen Intermediate Model (EIM)*** – Modelo no nível PIM da MDA que é gerado a partir dos modelos codificados utilizando a *EngenSBVR* e *EngenDSL* após a primeira fase de transformação, que permite modificações finas no esquema conceitual da aplicação;

e) ***EngenSBVR*** – uma linguagem específica de domínio criada para a modelagem do vocabulário e regras de negócio. É composta pela sua sintaxe definida em um arquivo do *XText Framework* próprio para este fim, um parser para as formulações lógicas de regras de negócio descritas em SBVR, de validadores semânticos que são acionados pelo editor ou pelo *EngenGenerator*, e o provedor de escopo, que consiste em um analisador que determina quais são os elementos referenciáveis válidos em um determinado contexto;

f) ***EngenSBVR Editor*** – um editor baseado no *XText Framework* para facilitar a manipulação da *EngenSBVR* e é composto de componentes relacionados com a tarefa de edição dos arquivos de código fonte da linguagem, similarmente ao *EngenDSL Editor* um componente para a visualização da árvore de formulações lógicas das regras de negócio (*SBVR LF Viewer*).

As próximas seções explicam o funcionamento e a estrutura dos componentes da Figura 10, seus relacionamentos, bem como justificam algumas escolhas arquiteturais na criação dos componentes citados nesta figura.

4.2.1 Lexer e Parser das Linguagens

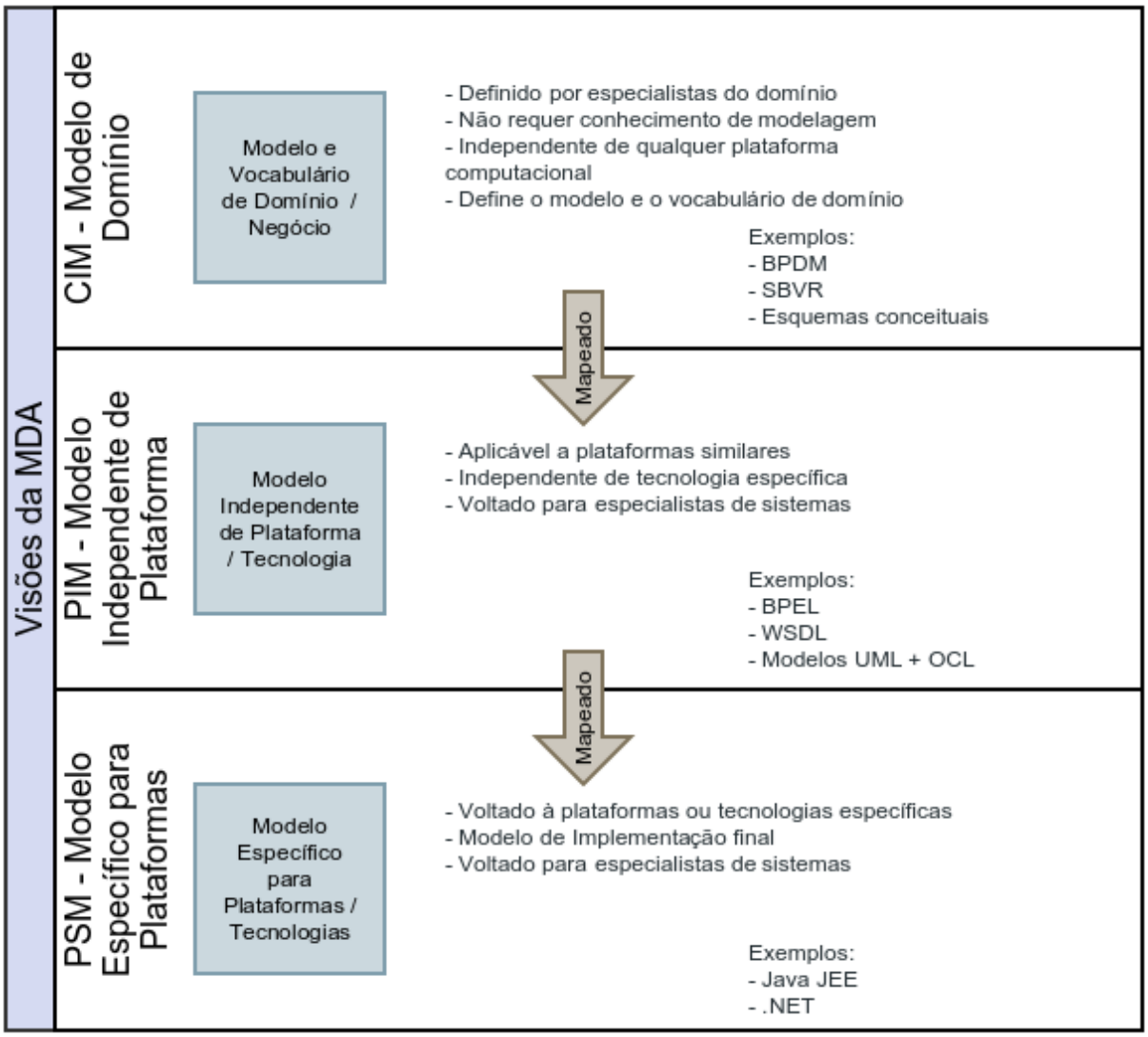
O *Lexer*, gerado automaticamente pelo *XText* para a linguagem que está sendo definida na ferramenta, é um conjunto de classes *Java* que definem os tokens válidos para a linguagem em uma sequência de caracteres. Por exemplo, para os componentes da Figura 10 que representam a sintaxe das linguagens *EngenDSL* e *EngenSBVR*, o *XText* gera um componente que faz a leitura de um texto e reconhece entradas como *"entity"*, *"controller"*, *"TermSet"* e *"view"*, que estão previamente definidos como elementos da sintaxe das linguagens. O *Parser* também é um conjunto de classes que são encarregadas de verificar se estes tokens estão sendo utilizados de forma correta, ou seja, seguindo a semântica definida no metamodelo da linguagem alvo, *EngenDSL* ou *EngenSBVR*, neste caso. Isto é de extrema importância pois, criar o *Lexer* e *Parser* manualmente seria difícil, mesmo utilizando o gerador do *AntLR* (PARR, 2014), que é utilizado internamente pelo *XText*. Além das

checagens criadas automaticamente pelo *XText*, o *framework XText* permite que semânticas adicionais sejam adicionadas ao modelo, como validação de referências válidas em determinado contexto. Quando instanciado, o metamodelo proverá a coerência sintáxica e semântica para a criação do modelo final da aplicação, que deve ser personalizado pelo desenvolvedor para prover funcionalidades além das básicas, definidas pelo *framework*. Por exemplo, pode-se definir que a medida em que entradas do modelo são criadas pelo usuário da linguagem no editor, as validações ocorram automaticamente mostrando as possíveis inconsistências. Este é o papel dos componentes de validação das linguagens, criados pela solução apresentada e mostrados na Figura 10: o *EngenDSL Validator* e o *EngenSBVR Validator*. O leitor interessado em mais detalhes de como personalizar os componentes básicos de validação gerados pelo *XText*, deve se referir à página do projeto *XText* (XTEXT, 2016).

4.3 PROCESSO DE DESENVOLVIMENTO UTILIZANDO A PLATAFORMA ENGEN

Para a utilização da *Plataforma Engen*, é necessária a adoção de um processo de desenvolvimento de software baseado em modelos. O processo de desenvolvimento baseado em modelos mais conhecido e adotado atualmente é o MDA da OMG, discutido no Capítulo 2. No padrão MDA existem três níveis de modelos que são definidos, os quais passam por transformações sucessivas até chegarem a um modelo que possa ser transformado em código ou executado na plataforma desejada: CIM (*Computational Independent Model*), PIM (*Platform Independent Model*) e PSM (*Platform Specific Model*) conforme mostrado na Figura 11.

Figura 11- Modelo de Visões da MDA



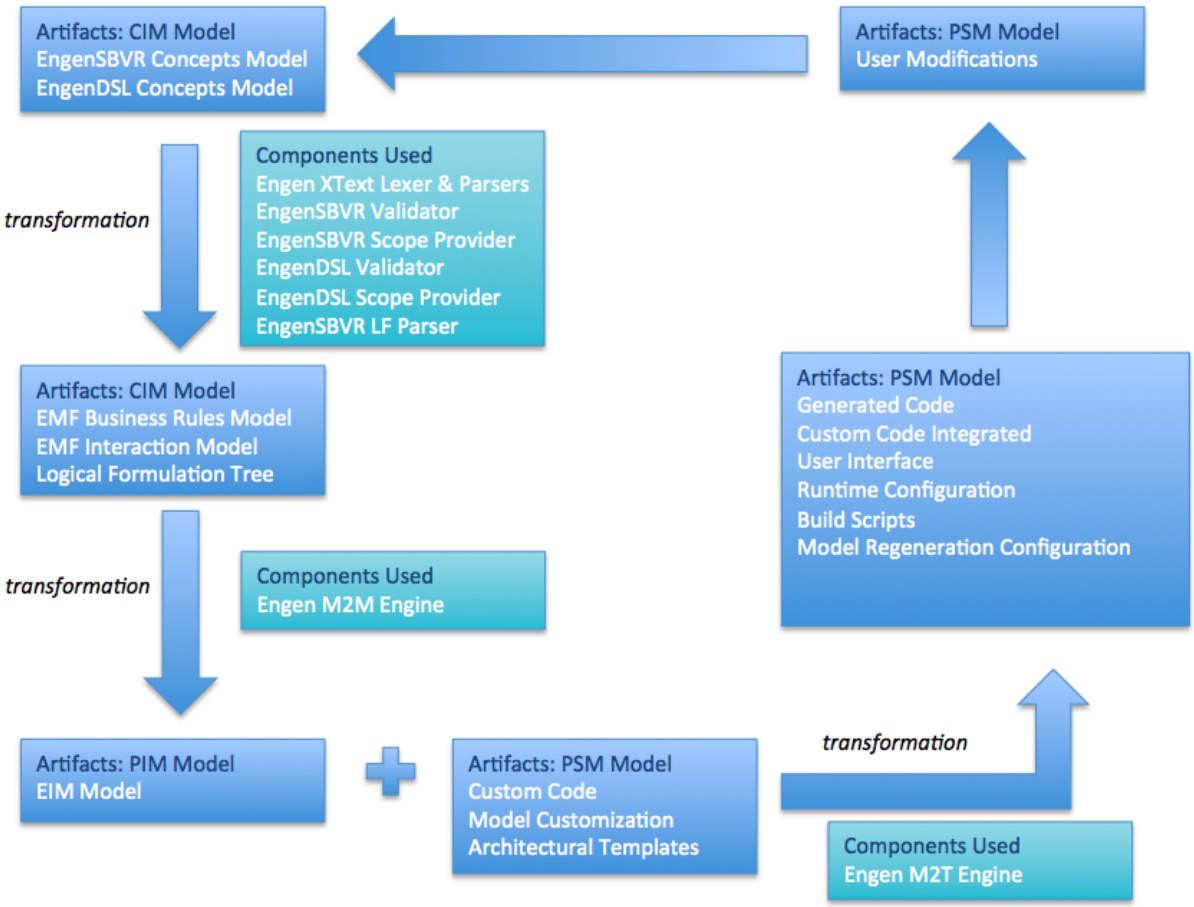
Fonte: Adaptado de Linehan (2006).

O Modelo CIM, ou o modelo do domínio, é utilizado para modelar o domínio da aplicação independentemente da plataforma ou arquitetura do sistema. Este modelo deve ser mapeado e transformado em um PIM, que adiciona ao modelo CIM inicial, informações arquiteturais, sem se comprometer com a plataforma utilizada. Por fim, este modelo (PIM) é transformado em um PSM, que adiciona ao modelo do sistema informações inerentes à plataforma em que o sistema será utilizado.

A arquitetura e o processo de desenvolvimento utilizados nesta dissertação objetivam mapear e transformar modelos de negócio no nível CIM, descritos em SBVR utilizando a sintaxe da DSL *EngenSBVR*, em conjunto com modelos de interação com o usuário descritos por meio da *EngenDSL*, para um modelo PIM chamado EIM, que pode ser utilizado na geração de sistemas de informações empresariais para a Web ou dispositivos móveis, por

meio de transformações para um modelo PSM, ou código final. Porém, conforme mencionado no Capítulo 1, o processo escolhido é baseado no MDSD, no qual, não somente os modelos transformados são utilizados, mas código criado manualmente é produzido e integrado à solução final. As transformações, segundo o processo sugerido, devem ocorrer constantemente durante o processo de desenvolvimento. A Figura 12 mostra alguns componentes da plataforma sugerida alocados ao processo de desenvolvimento, juntamente com os artefatos de entrada e saída de cada fase.

Figura 12 - Modelo do Processo de Desenvolvimento MDSD *Engen*



Fonte: Autoria própria.

Na Figura 12, são mostradas três atividades de transformação, os componentes utilizados em cada fase, além dos artefatos utilizados, ou produzidos, em um possível ciclo de desenvolvimento baseado no processo sugerido. Estes artefatos, servem de entrada para o componente *M2M Engen Generator*. As transformações propostas são detalhadas na Seção 4.7.

Inicialmente, os modelos no nível CIM da MDA, baseados na *EngenSBVR* e *EngenDSL*, são criados pelos desenvolvedores e analistas de negócio (nomeados de

EngenSBVR e *EngenDSL Concepts Model* na Figura 12). Para realizar esta tarefa, qualquer editor de texto pode ser utilizado para a criação do modelo, ou preferencialmente, os editores criados na solução proposta, devido ao seu suporte à validação entre os modelos de negócio e de interação com o usuário: *EngenDSL Editor* e *EngenSBVR Editor*. Estes modelos são posteriormente transformados no modelo EMF das duas linguagens, além da árvore de formulações lógicas do SBVR. Os modelos EMF são produzidos pelos *Lexer* e *Parser* das linguagens, que validam o modelo conceitual utilizando os validadores e os provedores de escopo das linguagens (*EngenSBVR e EngenDSL Validators & Scope Providers* da Figura 12). Os editores das linguagens, ou o *EngenGenerator* (no caso da utilização de editores comuns), irão instanciar os *Lexer* e *Parser,* explicados na Seção 4.2.1, produzidos pelo *XText* e personalizados na solução proposta, e caso os modelos textuais estejam de acordo com as validações sintáticas e semânticas definidas pelos validadores (*EngenDSL Validator* e *EngenSBVR Validator*), um modelo EMF será criado contendo o modelo conceitual da aplicação definidos utilizando a semântica das linguagens. Neste contexto, durante a validação, os componentes que implementam o modelo de provedor de escopo do *XText – Scope Providers,* que são elementos da arquitetura de validação da linguagem para identificar quais *tokens,* ou símbolos, são válidos em determinado contexto *(EngenDSL Scope Provider e EngenSBVR Scope Provider* na Figura 12*),* são bastante utilizados para permitir que ambos modelos possam referenciar elementos entre si, e entre os diversos arquivos de código fonte. Sua função é de permitir que estas referências sejam resolvidas e validadas.

O componente *EngenSBVR LF Parser* (LF é uma abreviação do conceito de Formulações Lógicas – do inglês *Logical Formulation*, da SBVR apresentadas no Capítulo) é utilizado para a transformação de regras descritas utilizando o modelo da *EngenSBVR*, detalhado na Seção 4.5 deste trabalho, em uma árvore de formulações lógicas de acordo com a especificação do SBVR. Esta árvore é a base para identificação semântica dos conceitos das regras de negócio descritas em SBVR.

A partir dos modelos EMF e da árvore de formulações lógicas, é gerado um **modelo conceitual da aplicação** chamado *Engen Intermediate Model* – EIM, mostrado na Figura 12, e que logicamente está situado no nível PIM da MDA, que é constituído pelos modelo de domínio ou estrutural, regras e vocabulário de negócio, *layout* e o modelo de interação com o usuário do sistema. É papel do componente *M2M Engine* do *EngenGenerator*, gerar este modelo.

O EIM gerado, pode ser personalizado e é fisicamente materializado em um arquivo XML. É papel do *EngenDSL Editor*, permitir a personalização visual deste modelo. Tal

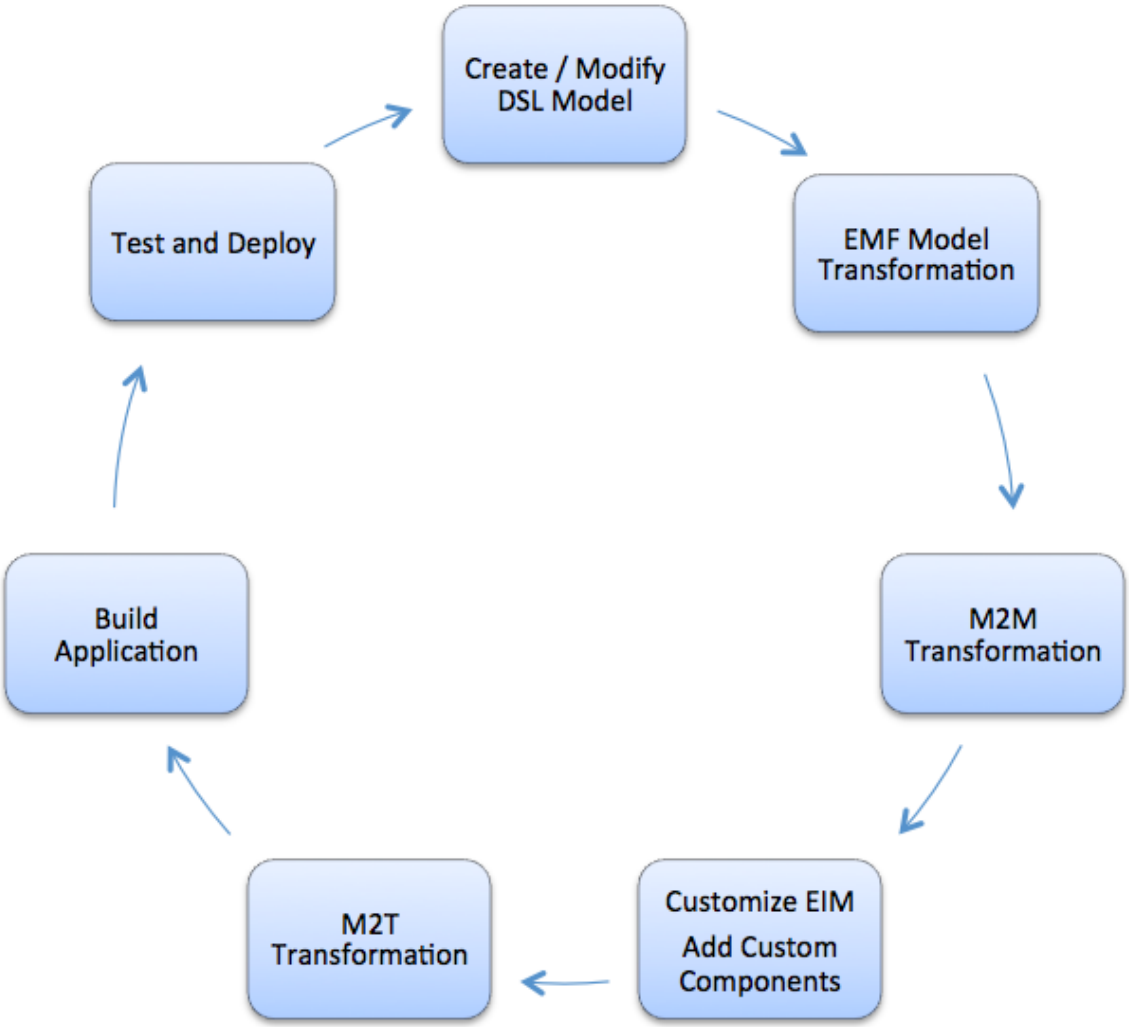
personalização, quando realizada no editor, ocorre através de uma técnica chamada de model *weaving* (STAHL; VOELTER; CZARNECKI, 2006) e está representada na Figura 12 com o nome "*Model Customization*". A implementação deste conceito, consiste em adicionar, remover ou alterar informações do modelo EIM, definindo tal personalização em arquivos de configuração separados do modelo original. A justificativa para a utilização desta técnica é a necessidade de que em sucessivas transformações e serializações dos modelos EMF para o EIM, as personalizações anteriores não sejam perdidas. Isto é feito automaticamente pelo *EngenDSL Editor*, que armazena em arquivos separados as informações de personalização, e que serão utilizados posteriormente pelo motor de transformação *M2T Engine* mostrado na Figura 12.

Na terceira etapa de transformação, o componente *M2T Engine* irá mesclar os dados do modelo EIM, com os dados de personalização criados ou alterados pelo desenvolvedor. Além disto, nesta fase, é possível adicionar código manual à solução (*Custom Code* na Figura 12), que deve ser criado preferivelmente em um diretório separado do código gerado (um exemplo será mostrado no Capítulo 5 do exemplo de aplicação), que será integrado à solução final pelo componente *M2T Engine*. Nesta fase, o *SBVR Global Concepts Registry Module* – SGRM, que contém os conceitos advindos do modelo da *EngenSBVR*, é também utilizado para enriquecer o EIM com informações de regras de negócio. O M2T *Engine* irá utilizar as informações do modelo EIM personalizado, e disponibilizá-las para serem aplicados aos *templates* da arquitetura (*Architectural Templates* na Figura 12) definida para a aplicação final. Por exemplo, estes *templates* podem estar configurados para a geração de código em uma plataforma específica – PSM, como *Java/Web* ou .NET/*Web*, além de outros artefatos necessários na aplicação final, como scripts de construção e instalação da aplicação.

O resultado desta terceira fase de transformação é o código gerado, juntamente com o código provido pelo desenvolvedor, código de testes, além de arquivos da interface com o usuário, scripts de geração de código para o projeto, scripts de construção da aplicação e implantação e configurações de execução da aplicação. Com modificações do usuário ao modelo CIM inicial, o processo então é repetido até obter-se a aplicação de software final, objetivo do esforço de desenvolvimento.

Na Figura 13, pode-se visualizar um resumo deste processo onde as sucessivas atividades são representadas de maneira cíclica.

Figura13 – Atividades do Processo

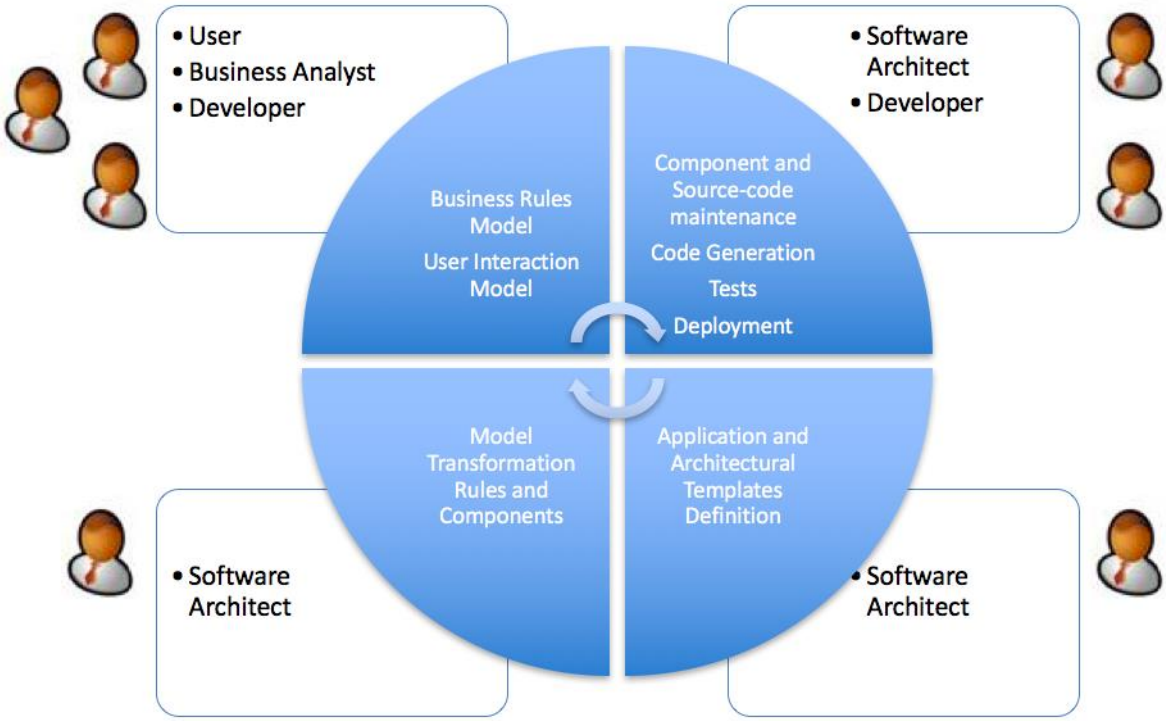


Fonte: Autoria própria.

4.3.1 Papéis no Ciclo de Desenvolvimento

No processo MDSD proposto neste trabalho, os papéis do usuário (conhecedor do negócio), desenvolvedor (analista de sistemas que está em contato constante com o cliente e traduz as intenções do negócio para o correto formato esperado nas DSLs) e o arquiteto de software (especialista na tecnologia e arquitetura escolhida para os projetos) são alocadas nas principais tarefas em relação ao processo de desenvolvimento. A Figura 14 mostra esquematicamente os papéis, e respectivas tarefas à qual estão alocados no processo de desenvolvimento sugerido.

Figura 14 - Modelo de Responsabilidade dos Papéis na Arquitetura



Fonte: Autoria própria.

Os três papéis na parte superior esquerda – usuário, analista de negócio e desenvolvedor, são comuns em processos de desenvolvimento. Eles são responsáveis por produzir e manter os modelos no nível – CIM, mais próximos dos conceitos de negócio e independente de plataforma. O código manualmente produzido e componentes externos que devem ser integrados à solução, a execução das transformações, a execução de testes e implantação dos artefatos em servidores de aplicação, são tarefas executadas tipicamente pelos arquitetos de sistema e desenvolvedores do projeto. Os arquitetos e desenvolvedores que detém maior conhecimento da arquitetura completa do produto e de como personalizar as ferramentas sugeridas neste trabalho, devem ser responsáveis por realizar a manutenção dos *templates* arquiteturais quando necessário. Além disto, a evolução das ferramentas e DSLs devem ser definidas com apoio dos arquitetos.

A próxima seção apresenta a linguagem *EngenDSL*, a estratégia utilizada em sua construção e exemplos de sua utilização.

4.4 ENGENDSL

Esta seção explica as decisões arquiteturais tomadas no projeto de criação da *EngenDSL*, que é um dos componentes da *Plataforma Engen*, conforme mostrado na Figura

10. A linguagem é descrita a partir de exemplos de utilização para permitir melhor compreensão dos conceitos.

4.4.1 Motivação para a criação da EngenDSL

Pesquisas na área de reúso de software mostram que para atingir avanços significativos, é necessária uma mudança de paradigma no sentido da definição de famílias de software ao invés de sistemas individuais (CZARNECKI ; HELSEN, 2003). O MDSD é uma das propostas que preconizam esta prática. Assim como em toda técnica MDD, o MDSD fornece aos modelos uma maior importância no processo de desenvolvimento de software (MORENO; ROMERO; VALLECILLO, 2008). Porém, o foco do MDSD é criar software mais centrado nos conceitos do domínio do que nos conceitos tecnológicos (STAHL; VOELTER; CZARNECKI, 2006).

A abordagem MDSD geralmente preconiza o uso de linguagens específicas de domínio, que são comumente utilizadas para a especificação dos modelos durante o processo de desenvolvimento, com o objetivo de melhorar a expressividade, abstração e consistência dos mesmos com o domínio alvo (STAHL; VOELTER; CZARNECKI, 2006). Uma das vantagens da utilização de DSL para modelar domínios específicos é sua capacidade de prover um metamodelo do domínio alvo, possibilitando a validação dos modelos especificados baseados nesta formalização.

Especialmente na área de desenvolvimento de soluções para a *Web*, estudos indicam que a utilização deste tipo de abordagem, para o desenvolvimento das soluções de software, aumenta significativamente a qualidade da aplicação final (ROUSSEV, 2003). Pesquisas já mostraram que a Engenharia da Web – *WebE*, é uma área na qual os conceitos e técnicas da MDSD podem ser aplicadas com sucesso (MORENO; ROMERO; VALLECILLO, 2008). Por exemplo, em (KROISS; KOCH; KNAPP, 2009) é apresentada uma abordagem para desenvolvimento baseado em modelos para a *Web*. Entretanto, ela se baseia em uma arquitetura pré-definida (*Java*/JSF). Tais estudos (MORENO; ROMERO; VALLECILLO, 2008) mostraram resultados expressivos no que se refere à utilização de técnicas de MDSD no processo de desenvolvimento de software e na busca do aumento de produtividade e qualidade do software produzido. Entretanto, não abordam o problema da **integração** e **validação** dos **requisitos de negócio** perante a **modelagem conceitual, estrutural e de interação com o usuário do sistema**, de forma que em tempo de desenvolvimento sejam descobertos problemas do projeto e inconsistências em relação aos requisitos do usuário.

O suporte de ferramentas é um aspecto crucial na adoção de métodos e tecnologias de desenvolvimento, portanto, a adoção de uma plataforma já conhecida e utilizada por uma grande gama de desenvolvedores é desejável. Entretanto, isto não deve ser uma restrição à utilização da solução proposta neste trabalho, conforme os requisitos apresentados na introdução deste capítulo, que deve prover meios de utilização da proposta sem utilização de editores específicos. Desta forma, a solução proposta neste trabalho permite que o desenvolvedor utilize tanto ferramentas que lhes proporcionem grande produtividade baseado na plataforma *Eclipse*, quanto, por exemplo, em ambientes como um terminal remoto utilizando um editor como o VIM (MOOLENAAR, 1991). Destarte, todos os componentes integrantes do *EngenGenerator* mostrados na Figura 10, permitem que qualquer arquivo texto, ou sequência de caracteres lidos de uma fonte qualquer, possa ser processado como um modelo válido para as linguagens definidas na solução.

4.4.2 Interação com o usuário

Um dos maiores problemas da adoção da abordagem de desenvolvimento baseado em regras de negócio é a falta de ferramental para a geração do código final baseado nas especificações dessas regras. As ferramentas para transformação de regras em código do sistema geralmente são específicas para uma tecnologia, ou geram código para um motor de regras específico (BAJEC ; KRISPER, 2005a). Além disto, outros aspectos como a interação com o usuário, navegação, *layout* e integração com componentes precisam ser modelados para que a solução final seja completa. É necessário que uma possível solução para este problema possibilite que tanto as regras de negócio definidas em SBVR, quanto o modelo de interação com o usuário da aplicação sejam contemplados, e possam ser integrados e validados entre si, ou seja, que permita que regras de negócio definam parte do comportamento do sistema de informação, e que outros aspectos, e.g. modelo de navegação e interação com o usuário possam ser modelados. Outrossim, é desejável que estes modelos sejam baseados em um padrão que possa permitir interoperabilidade com o metamodelo do SBVR.

Para a modelagem da interação com o usuário, o padrão IFML foi adotado neste trabalho por prover interoperabilidade com outras aplicações, ferramentas e outros padrões da OMG, já que é baseado no padrão desta entidade para a descrição de linguagens. O padrão de definição de linguagens da OMG define que os modelos devem ser baseados em um metamodelo UML, e que este deve estar de acordo com o metamodelo MOF (*Meta-Object*

Facility). Essa conformidade entre os modelos é comum a todas as especificações da OMG. O metamodelo da IFML não é diferente. Isto permite que o modelo da linguagem possa ser exportado para outras ferramentas, que não as sugeridas neste trabalho, portanto, que entendam o padrão MOF. Além destes aspectos, a abordagem desta dissertação, contribui permitindo a modelagem do modelo conceitual do *layout* de componentes de apresentação ao usuário, não abordado originalmente pela IFML.

4.4.3 A Linguagem EngenDSL

A necessidade da integração da modelagem de interação com o usuário, juntamente com a especificação de regras de negócio, motivou a proposta de uma DSL textual, chamada ***EngenDSL***, para modelar a interação com o usuário de forma a dar suporte a aspectos estranhos às regras de negócio da aplicação. O modelo proposto neste trabalho, permite a modelagem dos aspectos inerentes à interação com o usuário, em consonância com o padrão IFML, utilizando uma linguagem específica de domínio, criada exclusivamente para este fim, chamada *EngenDSL*. A conformidade com o padrão IFML da OMG visa dar maior estabilidade e confiabilidade ao metamodelo da linguagem por já ter sido analisado por vários colaboradores do padrão (SIEGEL, 2005).

O metamodelo da IFML é estendido pela *EngenDSL* e segue as diretrizes desse padrão. A *EngenDSL* adiciona conceitos não abordados pela IFML, notadamente quanto ao ***layout*** das interfaces com o usuário e o encadeamento de ações representadas na linguagem.

Abstrações de interação com o usuário como telas, menus, componentes visuais e navegação não fazem parte do negócio das organizações, e, portanto, não devem ser modelados como regras de negócio. Entretanto, para uma efetiva utilização de técnicas MDSD em projetos de sistemas de informações empresariais, é importante que estes conceitos estejam presentes no modelo de forma a aumentar a qualidade, minimizar o retrabalho e o esforço envolvido na definição dos artefatos de software da aplicação.

A forma textual da linguagem e o número reduzido de instruções permitem que o desenvolvedor possa criar modelos de aplicações de uma forma ágil e precisa, visto que as instruções são constantemente validadas pelo editor, descrito na Seção 4.6 deste Capítulo, em consonância com as técnicas MDSD.

O principal objetivo da linguagem é o de auxiliar e facilitar o desenvolvimento de sistemas de informações empresariais para o profissional desenvolvedor de software,

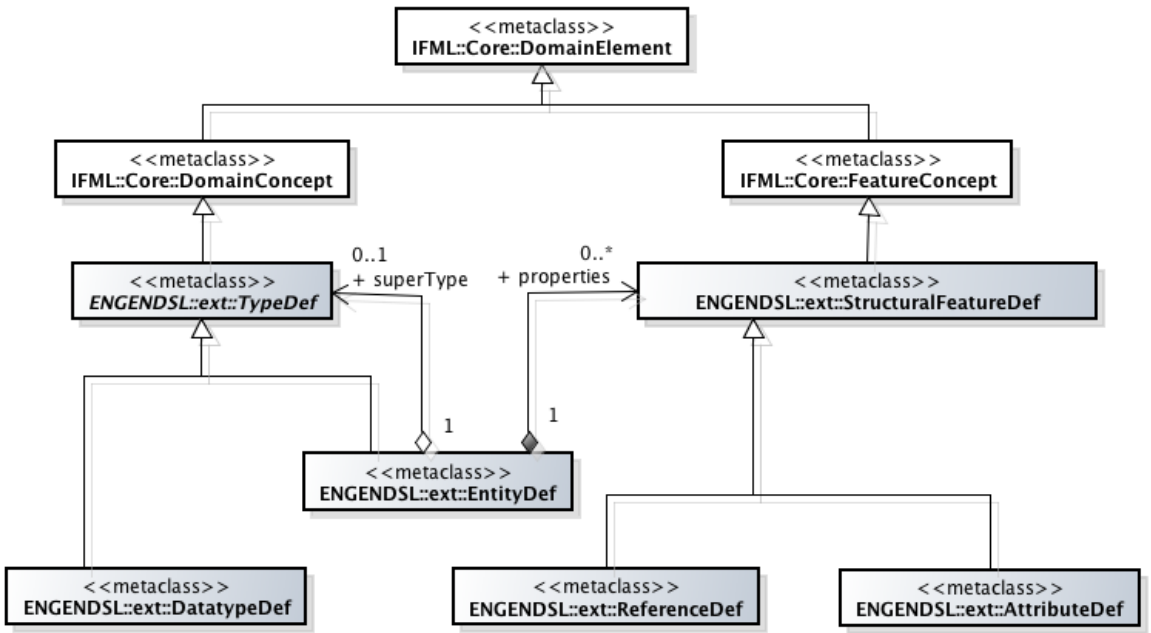
utilizando abstrações para os conceitos de interação do usuário baseados no padrão IFML. A integração com os conceitos de regras de negócio descritas em SBVR é feita pelos componentes de transformação de modelos do *EngenGenerator*.

As seções seguintes detalham o modelo, ou metamodelo, da *EngenDSL*, seus principais conceitos, sintaxe e semântica.

- EngenDSL - Detalhamento do Modelo

O metamodelo da *EngenDSL*, assim como o metamodelo da IFML, é dividido em visões, que é o particionamento do modelo em conjuntos de elementos logicamente conectados e que modelam um aspecto da interação com o usuário. Na Figura 15 é mostrado um diagrama UML do metamodelo conceitual da *EngenDSL* para a visão do metamodelo de domínio.

Figura 15 - Metamodelo EngenDSL - Visão de Domínio



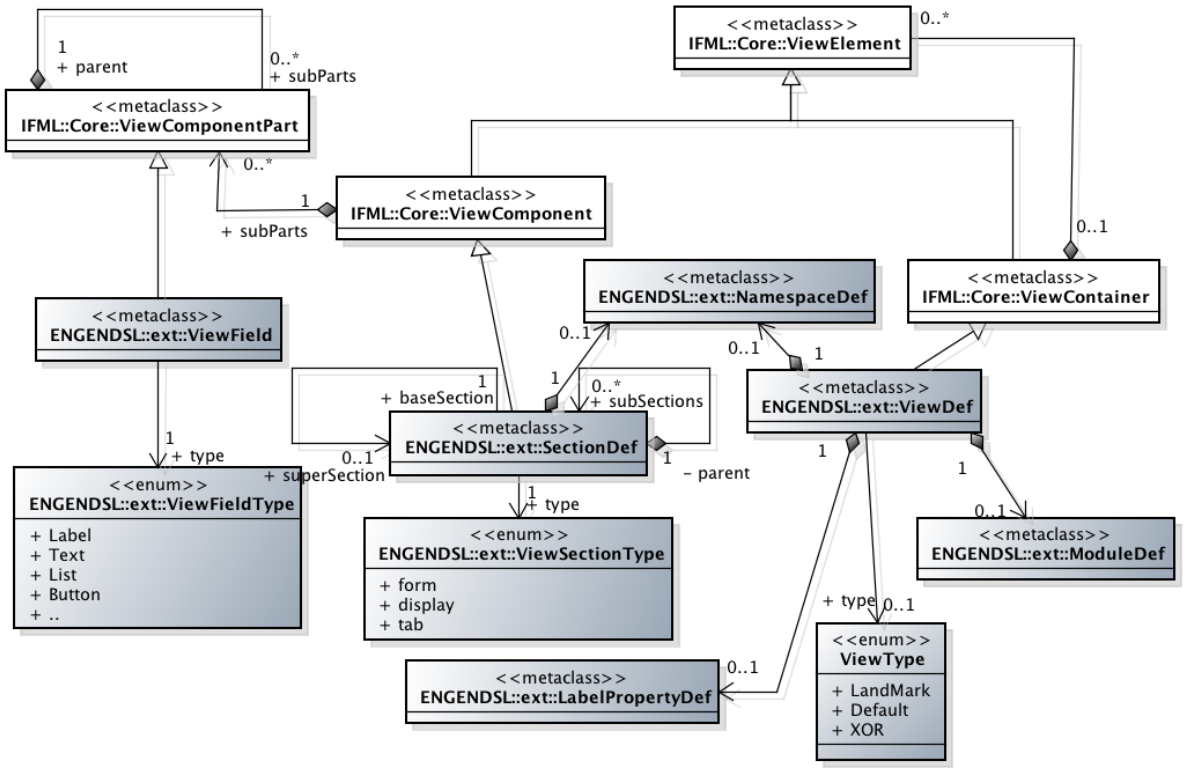
Fonte: Autoria própria.

No modelo da Figura 15 são mostrados os elementos do modelo IFML que representam conceitos de domínio da aplicação (*DomainElement*, *DomainConcept* e *FeatureConcept*) além dos elementos do modelo conceitual da *EngenDSL*, os elementos específicos da *EngenDSL* estão destacados, que estendem estes conceitos de tipos (*TypeDef* e *DataTypeDef)*, entidades do sistema (*EntityDef)*, e os dados que manipulam

(*StructuralFeatureDef*, *ReferenceDef* e *AttributeDef*), os quais serão detalhados nas seções seguintes.

Na Figura 16 é mostrado o metamodelo de visualização da *EngenDSL* e sua relação com os conceitos do padrão IFML.

Figura 16- Metamodelo de Visualização da EngenDSL



Fonte: Autoria própria.

Tanto na visão de domínio quanto na visão de visualização, os elementos específicos da *EngenDSL* estão destacados. Na visão de visualização os elementos da EngenDSL permitem a modelagem da interação com o usuário e telas do sistema (*ViewDef*), dos campos (*ViewField*) e seções de cada tela (*SectionDef*). A função e semântica de cada elemento dos modelos das Figura 15 e Figura 16 serão detalhadas a seguir:

- *TypeDef*

A metaclasse *TypeDef*, representa um conceito de domínio do IFML e serve para abstrair o tipo da entidade, ou conectar o modelo conceitual da aplicação a um tipo externo qualquer definido pelo usuário. Esta definição, permite que componentes criados

externamente possam ser utilizados em conjunto com tipos definidos internamente na linguagem.

- *EntityDef*

Na modelagem de domínio de sistemas de informação o modelo de entidades representa um conjunto de conceitos que estruturam dados, dos quais se pretende ter alguma memória, mas é independente da forma de armazenamento dos mesmos. Tais entidades são representadas na *EngenDSL* pela meta-classe *EntityDef,* e são instanciadas através da palavra-chave *"entity"*.

Cada entidade na linguagem proposta possui os seguintes papéis:

a) representar uma entidade, possivelmente com dados de negócio a serem persistidos;

b) definir os relacionamentos entre a entidade e outras entidades do modelo;

c) definir os atributos simples da entidade e que não representam relacionamentos.

A definição formal de uma entidade na linguagem, especificando a sintaxe da declaração da mesma em uma aplicação é mostrada na Figura 17. Ela serve como exemplificação da sintaxe definida na *EngenDSL*.

Figura 17- Definição na EngenDSL da sintaxe da declaração de uma Entidade do modelo da aplicação

```
EntityDef:
    (abstract?="abstract")? "entity" name=ID (COLON extends=[TypeDef])? "{"
        (namespace=NamespaceDef)?
        (
            (properties+=StructuralFeatureDef)*
        )
        (constraints=StructuralFeatureConstraintsDef)?
        (businessKeys+=BusinessKeyConstraintDef*)?
    "}"
;
```

Fonte: Autoria própria.

Conforme esta definição, entidades devem iniciar com a palavra-chave ***entity***, seguida do nome da entidade, podem ser abstratas, se precedidas da palavra-chave ***abstract***, ou até estender outras entidades. Esta informação é definida na linguagem com o símbolo ":", seguido do nome da entidade da qual se quer herdar – *TypeDef*. Apesar de ser uma

funcionalidade típica de linguagens gerais, neste caso, a declaração de herança em si não fere os requisitos funcionais da linguagem por tratar-se de uma simples definição da extensão de um conceito com semântica própria. Na linguagem, o *NamespaceDef* identifica o pacote ao qual algum elemento da linguagem está associado. O construtor *StructuralFeatureDef* define os vários atributos de uma entidade. Tal definição permite que atributos sejam de tipos simples como sequência de caracteres, inteiros ou datas, até referências para outras entidades do modelo.

- *ControllerDef*

Para satisfazer o requisito de abstrair na *EngenDSL* os conceitos do padrão MVC, a linguagem utiliza o conceito de *controllers* que representam, de acordo com esse padrão, componentes que são responsáveis por controlar a entrada e saída de dados da aplicação, bem como o redirecionamento para a *View* correta, ou outro *controller*. Os *controllers* da *EngenDSL* estão diretamente relacionados com o conceito de *Actions* da IFML.

Em algumas implementações de sistemas utilizando o MVC, encontra-se diferentes comportamentos para a implementação desse conceito, principalmente no que se refere a como o "*controller*" é acionado. A especificação da linguagem, em seu nível conceitual, não aborda como o "*controller*" deve ser acionado, ela define que existe um controlador que receberá uma requisição provinda de uma *View* ou outro *Controller*, que executará algum componente, e acionará uma *View*, ou outro *Controller*, ao final do processamento, os quais realizam a interface *LogicalPath* que será discutida adiante. A Figura 18 mostra como um *controller* é definido na *EngenDSL* utilizando o *XText*.

Figura 18 - Definição da sintaxe da declaração de *Controllers* na *EngenDSL*

```
ControllerDef:
    "controller" name=ID controllerBodyDef=ControllerBodyDef;

ControllerBodyDef:
    (superType=ControllerSuperTypeDef)? "{"
        controllerBodyElements=ControllerBodyElements
    "}"
;

ControllerBodyElements:
        namespaceDeclaration=NamespaceDeclaration?
        moduleDeclaration=ModuleDeclaration?
        (
            //These entries are unordered
            (target=ControllerTargetDeclaration)
            (label=LabelPropertyDef)?
            (showOnMenu=ShowOnMenuDef)?
            (before=ControllerBeforeDef)?
            (preProcessor=PreProcessorDef)?
            (postProcessor=PostProcessorDef)?
            (successPath=SuccessPathDeclaration)
            (failurePath=FailurePathDeclaration)
            (cancelPath=CancelPathDeclaration)?
            (otherPaths+=LogicalPathDeclaration)*
        )
        (constraints=FieldConstraintsDef)?
;
```

Fonte: Autoria própria.

Observa-se que *controllers* devem ser definidos na *EngenDSL* com a palavra-chave "*controller"*, os quais podem ser de um tipo pré-definido, indicado pela definição *ControllerSuperTypeDef* que define tipos de símbolos na linguagem para especificar comportamentos pré-definidos para um *controller* (e.g.: *Search*, *List*, *Edit*, *Create*, e etc.).

Vários outros atributos compõem a definição de um *Controller* na *EngenDSL*. Por exemplo, o *target* que define sobre qual entidade do modelo de domínio o *Controller* deve operar, os *preProcessor* e *postProcessor* que permitem definir comportamento específico antes e após a execução de um *Controller, respectivamente*.

- *ViewDef*

O padrão IFML define *ViewContainers* como componentes que contêm outros elementos da *view* - *ViewElements*, que podem ser *ViewContainers* ou *ViewComponents*. Na *EngenDSL*, essa mesma semântica se aplica ao conceito *View*, ou seja, representam

ViewContainers. *Views* são utilizadas para apresentar, ou capturar, algum dado do usuário segundo o padrão MVC. Na linguagem, as *Views* são entidades que representam este conceito de visualização. A apresentação de uma *View* ao usuário do sistema é resultado de uma ação de algum *Controller* (*Action* na IFML). Estruturalmente, as *Views* podem possuir subseções (ver subseção: *Sections*). Tais seções também representam uma parte da *View* e podem ser de diversos tipos como uma tabulação de elementos na tela, divisões ou formulários que contém dados que devem ser enviados para a aplicação. É importante notar que, no conceito da linguagem, seções de uma *View* podem conter subseções. Este conceito é perfeitamente compatível com o design de páginas *Web*, XML e outras formas de visualização, pois permitem utilizar um padrão de composição para apresentar conteúdo aninhado.

Na Figura 19 é mostrado um trecho da definição da linguagem para *Views*. Alguns esclarecimentos sobre os conceitos apresentados nesta definição são discutidos a seguir.

Figura 19 - Definição da sintaxe da declaração de uma *View* na *EngenDSL*

```
ViewDef:
    "view" name=ID "{"
        (
            ((namespace=NamespaceDef)?
            (label=LabelPropertyDef)?
            (module=ModuleDef)?)
            (layoutSection=ViewLayoutSectionReference)?
            (scope=ViewScopeDef)?
            (style=StylePropertyDef)?
        )
        (views+=ViewDef)*
        (sections+=SectionDef)*
    "}"
;
```

Fonte: Autoria própria.

A definição de uma *View* engloba seus aspectos visuais, estruturais e funcionais. A definição da propriedade *"module"* da *View* especifica a qual módulo funcional da aplicação esta *View* faz parte. A propriedade *label* indica um rótulo para ser utilizado na *View*. Como será discutido sobre o componente de *Layouts* da *EngenDSL*, uma *view* deve estar associada a uma seção da definição do *Layout* da aplicação. O atributo "*layoutSection*" define a qual elemento *LayoutSection* a *View* deve ser associada. Esta propriedade não está presente na especificação IFML, e pode ser definida como um dos pontos em que a *EngenDSL* estende o padrão IFML para acrescentar características necessárias para a definição de *Layouts* em sistemas de informações empresariais.

- *SectionDef*

Como já visto, estruturalmente *Views* são constituídas de seções. *Sections* são componentes que compõem as *Views* e apresentam conteúdo para o usuário. Definições de seções podem ser compostas e empilhadas em definições de *Views*, ou seja, *Views* podem conter várias seções e estas podem conter outras subseções. A definição de uma *Section* – *ViewSection*, na *EngenDSL* corresponde ao conceito *ViewComponent* da IFML. Assim, seções são utilizadas para capturar ou apresentar qualquer dado do usuário, o que a torna compatível com o padrão IFML. Seções possuem um nome, um tipo, um nome do módulo ao qual pertencem e podem conter outras seções ou campos – *Fields* (que é um tipo de subseção na *EngenDSL*). A Figura 20 mostra a especificação da linguagem para uma seção. Nela estão definidas as principais características estruturais da seção.

Figura 20 - Definição da sintaxe da declaração de uma seção na *EngenDSL*

```
SectionDef:
    "section" name=ID
        // One colon for section type
        (COLON type=[SectionTypeStatemetDef])?
        // Optionally followed by two colon for super types
        (COLON COLON super=[SectionDef])? "{"
        (
            (namespace=NamespaceDef)?
            ("target" (COLON)? target=SectionFormTargetDef )?
            (subSection+=Subsection)*
        )
    "}"
;

Subsection:
  SectionDef | ViewField | LabelPropertyDefinition | PutFieldsPropertyDefinition | StylePropertyDef
;
```

Fonte: Autoria própria.

Conforme esta definição, seções de uma *View* começam com a palavra-chave "*section"*, seguida de um nome e, opcionalmente, uma seção base e um tipo de seção, ambos conceitos semelhantes ao conceito de herança em linguagens com suporte a orientação a objetos. É importante notar a definição *Subsection*. Ela se refere às seções que uma seção pode conter, conforme a definição mostrada na parte inferior da Figura 20.

Entre as definições suportadas pelo conceito de *Subsection* estão:

a) *SectionDef*: chamada recursiva ao conceito de uma seção de uma *View*. Significa dizer que uma seção pode conter outras seções.

b) *ViewField*: define um campo de uma *Section*, podendo ser campos de entrada de texto, imagens, datas, listas de opções, tabelas, botões e rótulos.

c) *LabelPropertyDefinition*: definição de um rótulo para a seção.

d) *PutFieldsPropertyDefinition*: seção auxiliar ao modelo de domínio que permite definir que entidade está sendo visualizada. Esta definição colocará na *View* todos os campos definidos pela definição da entidade do modelo especificado.

➢ *ViewField*

O conceito *ViewField* corresponde ao conceito *ViewComponentPart* da IFML e representa um campo em uma seção de uma *View* e estão definidos na linguagem como vários tipos: botões, listas, caixas de texto, rótulos, etc. Estes campos por sua vez contêm propriedades e eventos associados. Estas propriedades definem, por exemplo, o identificador único do campo na *View*, a ordem em que o campo aparecerá na seção que o contém, o alvo (*target*) caso um evento ocorra (um *controller* ou uma *view* por exemplo), um estilo, um validador, dentre outros. O tipo do *ViewField* define semanticamente seu comportamento e características de apresentação como, por exemplo, os tipos *button* ou *select*. O modo como os eventos são acionados é dependente da arquitetura escolhida pelos *templates* na transformação do modelo alvo. Todo campo possui um evento padrão associado. Por exemplo, um botão possui um evento padrão: *click*. A propriedade *target* deste campo então poderá indicar qual *Controller* ou *View* será acionado quando o evento ocorrer. A Figura 21 mostra a definição de um *ViewField* na *EngenDSL*.

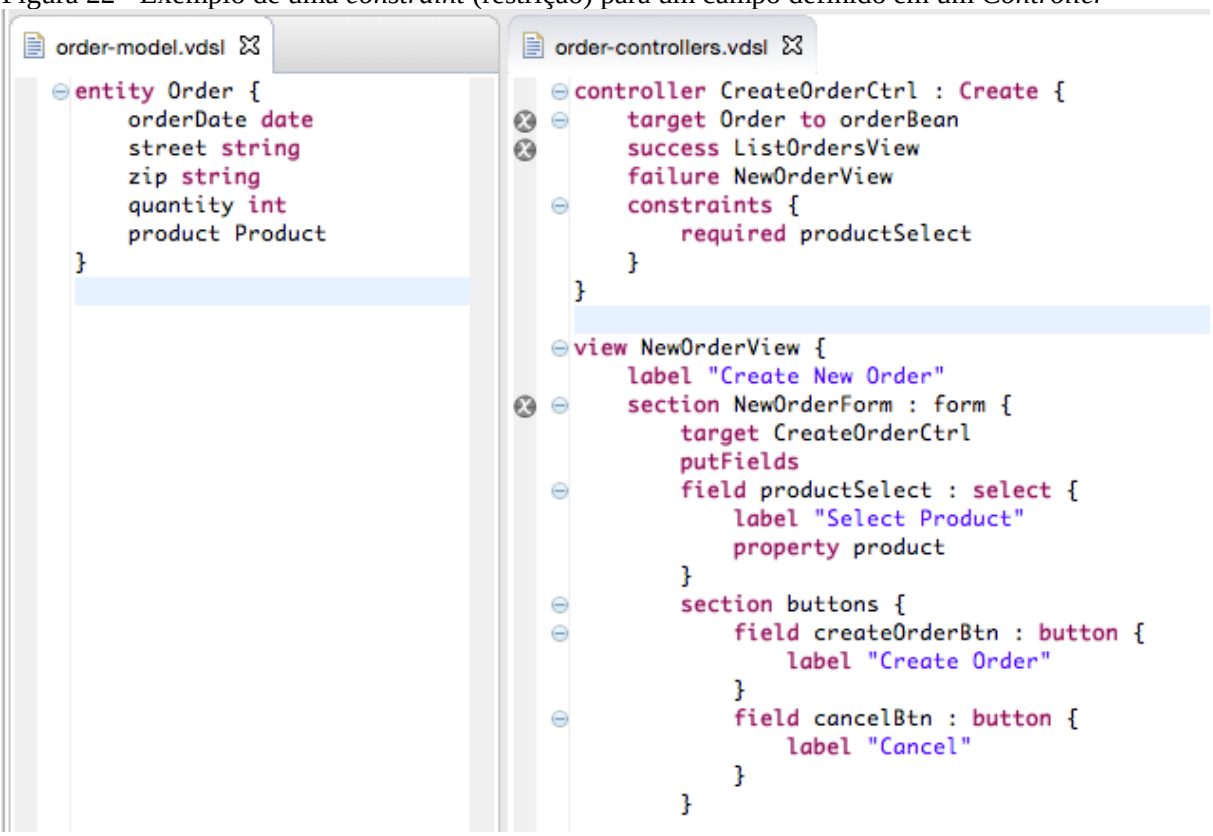
Figura 21- Sintaxe de um *ViewField* na *EngenDSL*

```
ViewDef:
    "view" name=ID "{"
        (
            ((namespace=NamespaceDef)?
            (label=LabelPropertyDef)?
            (module=ModuleDef)?)
            (layoutSection=ViewLayoutSectionReference)?
            (scope=ViewScopeDef)?
            (style=StylePropertyDef)?
        )
        (views+=ViewDef)*
        (sections+=SectionDef)*
    "}"
;
```

Fonte: Autoria própria.

Os *ViewFields* possuem várias propriedades. São exemplos um identificador único para o campo – *IdPropertyDef*, um label – *LabelPropertyDef*, a propriedade do componente de domínio – *FieldPropertyDef*, ou valor que irão apresentar – *SourcePropertyDef* e restrições – *FieldConstraintsPropertyDef*. As últimas representam validações que devem ocorrer quando o valor deste campo for manipulado ou submetido como campo de um formulário. A Figura 22 mostra um exemplo da linguagem instanciada no editor da plataforma e mostra como o campo *productSelect*, definido na *View NewOrderView*, está definido como obrigatório (*required*) na criação de uma ordem conforme definido pelo controller – *CreateOrderCtrl*.

Figura 22 - Exemplo de uma *constraint* (restrição) para um campo definido em um *Controller*



Fonte: Autoria própria.

- *LogicalPath*

O conceito de *LogicalPath* está diretamente associado ao conceito de navegação entre *Views* e *Controller*s. Um *Controller*, ao final do processamento, redireciona o usuário para um caminho. Definir este caminho é a função de um *LogicalPath*, que não é nada mais que

uma abstração para o conceito de um redirecionamento. Entretanto, como é possível que para a consecução de uma tarefa um *Controller* possa resolver acionar outro *Controller*, *Views* e *Controller*s representam, e são extensões de *LogicalPath* dentro da linguagem. Assim, um *Controller* poderá definir, de acordo com sua lógica de execução, redirecionar o usuário para um caminho – *LogicalPath*, que no final irá produzir uma *View*, ou encaminhará o fluxo da aplicação para outro *Controller*. Note-se que, em algum momento, o último *LogicalPath* de um determinado caminho lógico da aplicação será uma *View*. Para suportar o conceito de serviços que atendem demandas assíncronas de componentes *Web*, opcionalmente os *controllers* podem simplesmente retornar valores para serem apresentados pela aplicação em uma *View*. Cabe à implementação dos *templates* arquiteturais da aplicação dar suporte a tais requisitos.

Para fins de concisão, e atender aos requisitos da linguagem, conforme definido na introdução da Seção 3, é necessário que a definição de um *Controller* não necessite ter estruturas condicionais definidas diretamente na linguagem. Assim, como todo *Controller* necessita redirecionar o usuário para alguma *View*, ou outro *Controller*, é necessário pelo menos um *LogicalPath* definido para o *Controller*. Entretanto, esta definição poderia ser calculada automaticamente pela implementação do sistema dentro da arquitetura alvo da aplicação. Sua definição, nestes casos, pode ser meramente simbólica ou mapear sequência de caracteres, por exemplo, para nome de *Views* ou *Controllers*.

- *Layout*

Como mencionado anteriormente, a *EngenDSL* inclui uma maneira de modelar o ***layout*** das *views*. Este conceito não está presente originalmente no IFML. A ideia é que o desenvolvedor possa descrever como os componentes serão produzidos, que partes estão presentes na *View* (cabeçalhos, menus, corpo, rodapé e etc.), que estilos CSS devem ser aplicados a cada tipo de campo na *View* e como as seções são compostas. A definição de um *layout* juntamente com a referência para seu estilo (definido na próxima seção) na *EngenDSL* pode ser vista na Figura 23.

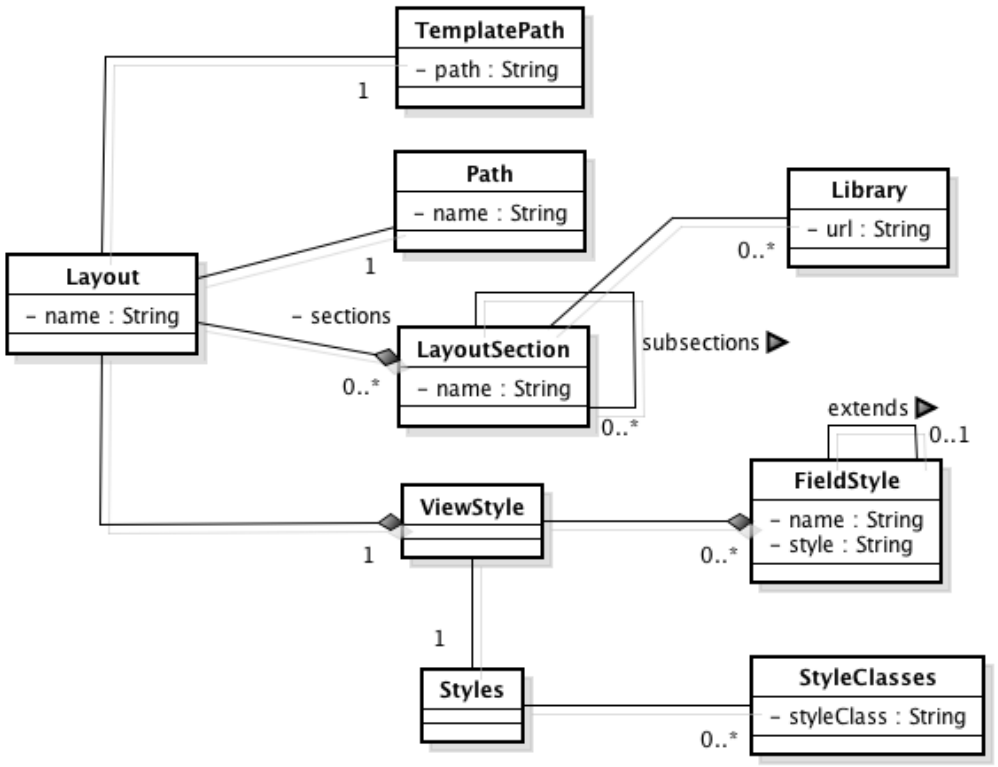
Figura 23 - *LayoutDef* e *ViewStyleDef* definições na *EngenDSL*

```
LayoutDef:
    "layout" name=ID "{"
        (templatePath=TemplateDef)?
        (path=PathDecl)?
        (sections+=LayoutSectionDefRef)*
        (style=ViewStyleDefRef)
    "}"
;
ViewStyleDefRef:
    "style" (COLON)? style=[ViewStyleDef]
;

ViewStyleDef:
    "view_style" name=ID "{"
        (blocksDef=BlockDef)?
        (styles=StylesDef)?
        (fieldStyles+=FieldStyleDef)*
    "}"
;
```

Fonte: Autoria própria.

Para um melhor entendimento da proposta de *layouts* da *EngenDSL* o modelo apresentado na Figura 24 é representativo do modelo conceitual criado.

Figura 24 - Modelo Conceitual de *Layouts* na *EngenDSL*



Fonte: Autoria própria.

Layouts possuem um nome que aparece logo após a palavra-chave *layout*, e possuem os atributos: *templatePath*, *path* e *style*. O *templatPath* permite definir qual *template* deve ser interpretado para o *layout*. O *path* define qual arquivo, no final da transformação, que será gerado e o *style* é uma referência para um *ViewStyle*, que representa os estilos a serem utilizados para a *View* e seus componentes. A próxima seção apresenta o *ViewStyle*.

Os *Layouts* podem ser compostos por bibliotecas (*Library*) – que representam componentes que o *layout* pode adicionar e referenciar como bibliotecas *JavaScript* ou CSS, e de seções de *layout* (*LayoutSection*), que possuem os mesmos atributos de um *layout*, e também pode conter outras seções ou bibliotecas. As seções de *layout* podem opcionalmente ser declaradas como *Strings* (com o nome da seção entre aspas) se a seção não precisar conter outras seções. A Figura 25 mostra um exemplo concreto da definição de um *layout* na *EngenDSL*.

Figura 25- Exemplo da definição de um *Layout* utilizando a *EngenDSL*

```
// The main layout used by this application
layout main_layout {
    path: "/WEB-INF/jsp/tiles/layouts/main_layout.jsp"
    section: head
    section: "header"
    section: two_column_body
    style: bootstrap_tabular
}
layout_section head {
    path: "/WEB-INF/jsp/head.jsp"
    library: jquery
    library: bootstrap
    library: customStyle
}
library jquery {
    urls {
        "/jquery/jquery-1.10.1.js",
        "/jquery/ui/jquery-ui.js",
        "/jquery/themes/base/jquery.ui.all.css"
    }
}
```

Fonte: Autoria própria.

- *ViewStyle*

ViewStyles começam com a palavra-chave *view_style* e são definições de formatação de estilos de uma *View*. A Figura 23 mostrou sua a definição na *EngenDSL*.

ViewStyles contém principalmente dois tipos de seções: *Styles* e *FieldStyle*. A seção *Styles* começa com a palavra-chave *styles* e define padrões de estilos genéricos para elementos da *View*. Por exemplo, na Figura 26 pode-se ver um exemplo da definição de um *ViewStyle* onde o estilo dos *fieldset* de uma *View* é definido com o valor *"form-group"*, enquanto que o estilo dos elementos *tab* (tabulações) da página será *"tab-section-style"*.

Figura 26 - Exemplo da definição de um ViewStyle usando a EngenDSL

```
view_style bootstrap_tabular {
    styles {
        tab: "tab-section-style"
        form: "form"
        form_div: "container-fluid"
        display: "row-fluid"
        fieldset: "form-group"
        tabheader: "nav nav-tabs"
        tabheaderitem: "tab-header-item-style"
        content: "div-content-style"
    }
    field_style GenericFieldStyle {
        type: any
        style: "form-control"
        label: "control-label"
    }
    field_style TextAreaStyle : GenericFieldStyle {
        type: textarea
    }
}
```

Fonte: Autoria própria.

Na Figura 27 é mostrado um exemplo de como os *templates*, utilizando a linguagem *XTend* (XTEND, 2016), utilizam esta informação, que foi previamente armazenada no modelo EIM produzido como resultado da transformação inicial M2M do modelo da *EngenDSL*.

Figura 27 - Exemplo de um template XTend utilizando a informação do ViewStyle

```
«IF field.visible»
    <!-- INIT SelectionInputFieldsTemplate.xtend#parse. Processing field: [«field.name»] for section: [«section.name»] -->
    <div id="field_control_group_«field.name»" class="«getStyleFor('fieldgroup')»">
        «IF field.useControlLabel»
            <label id="lbl_«field.name»" for="«field.name»" accesskey="«accessKey»" class="«field.labelStyle»">
                «IF required»
                    <strong><b>*</b></strong>
                «ENDIF»
                «field.label»
            </label>
        «ENDIF»
```

Fonte: Autoria própria.

Neste exemplo, a variável *field* representa o *ViewField* da *View* que está sendo processado no momento e faz parte do modelo EIM. O método do componente *EngenGenerator – getStyleFor* é acionado para retornar o estilo correto para este campo. O método é parecido com o código da Figura 28.

Figura 28 - Visão parcial do método *getStyleFor* do componente *Engen Generator* para os *templates*

```
/**
 * Returns a style for an element name. It will query all possible styles
 * (field, sections, etc)
 *
 * @param elementTypeName
 *            the element name to look for (e.g.: button, text, etc)
 * @return the style for the element
 */
public String getStyleFor(String elementTypeName) {
    // Check for sections
    ViewStyle layoutStyle = project.getControlLayer()
            .getCurrentLayoutViewStyle();
    UndefinedModelPropertyAction layoutStyleMissingAction = getUndefinedPropertyAction();
    if (layoutStyle == null) {
        UndefinedModelPropertyAction.execute(layoutStyleMissingAction,
                new IllegalArgumentException(
                        "The layout style is not defined in the project!"),
                log);
        return "undefined-style";
    }
    // Get the correct section type
    ViewSectionType viewSectionType = ViewSectionType.get(elementTypeName);
    String styleForSectionName = layoutStyle.getStyleFor(elementTypeName);
    if (StringUtils.isNotEmpty(styleForSectionName)) {
        return styleForSectionName;
    }
```

Fonte: Autoria própria.

Pode-se ver que este método é um método geral para os componentes da *View* e não somente para o campo. Ele pode ser acionado por qualquer *template* interessado em saber o estilo definido para uma porção da *View*.

- *FieldStyle*

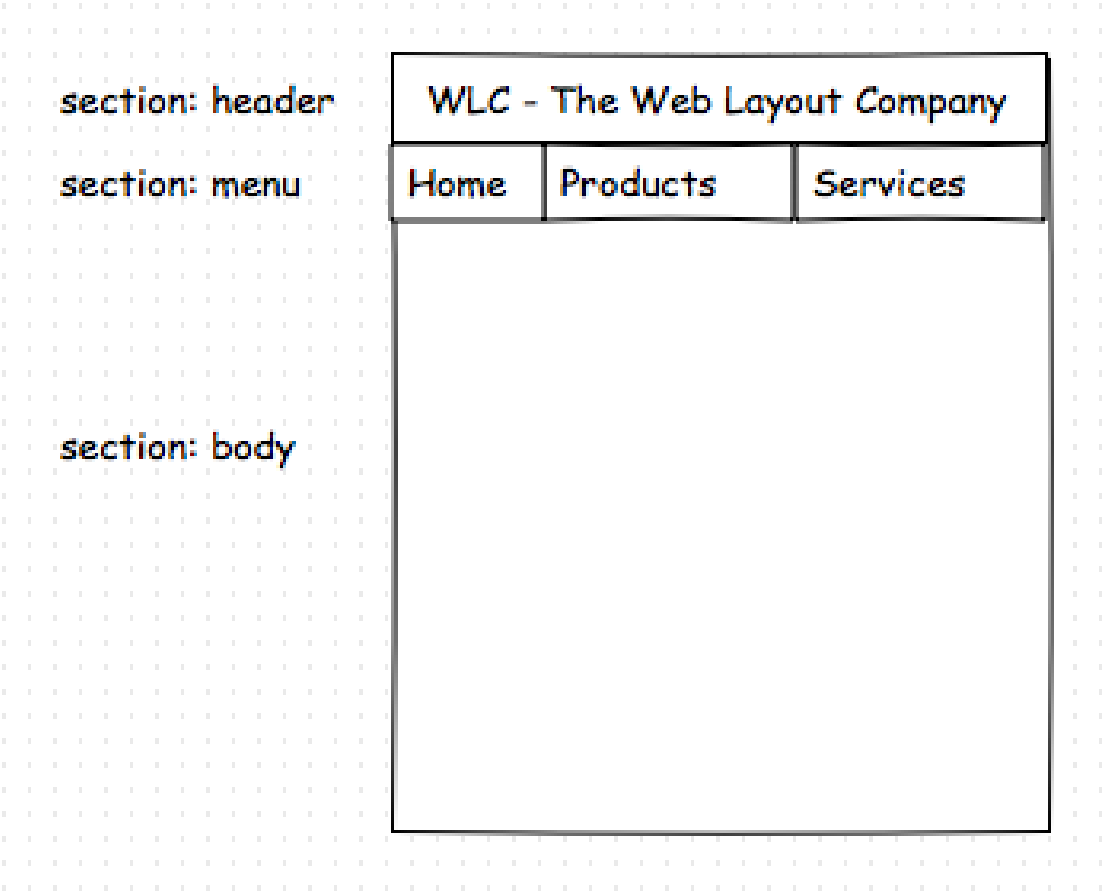
Um *FieldStyle* permite especificar o estilo para cada tipo de campo utilizado no projeto. A sua definição é feita utilizando-se a palavra-chave *field_style*, seguida por um nome, e opcionalmente seguido por dois pontos, e um nome de outro *FieldStyle* do qual se deseja herdar as características. Isto permite que estilos de campos sejam compostos para evitar retrabalho. O exemplo da Figura 27 mostra como alguns estilos estão definidos.

- *LayoutSection*

Os *layouts* podem ser compostos de uma ou mais seções. Uma seção de um *layout* representa uma parte da *View* que será composta para ser apresentada para o usuário. Uma

definição de uma seção representa uma parte desta *View*. Por exemplo, a Figura 29 mostra um protótipo de um *layout* em uma aplicação hipotética.

Figura 29 - Exemplo de um propósito de um *Layout* de uma *View*.



Fonte: Autoria própria.

As seções que compõem este *layout* devem ser mapeadas utilizando-se *LayoutSections* para o *layout* da aplicação. O código da *EngenDSL* necessário para que o mesmo seja produzido é mostrado na Figura 30.

Figura 30 - Exemplo da configuração do *Layout* para representar o protótipo na *EngenDSL*.

```
layout wlc_layout {
    path "/web/layout.html"
    section top
    section "body"
}

layout_section top {
    section "header"
    section "menu"
}
```

Fonte: Autoria própria.

É importante lembrar que os *Layouts* não se preocupam com a produção do conteúdo. Eles simplesmente fornecem informações de como este conteúdo será apresentado. É função dos *Controllers*, *Views*, *Sections* e *ViewFields* produzir tal conteúdo.

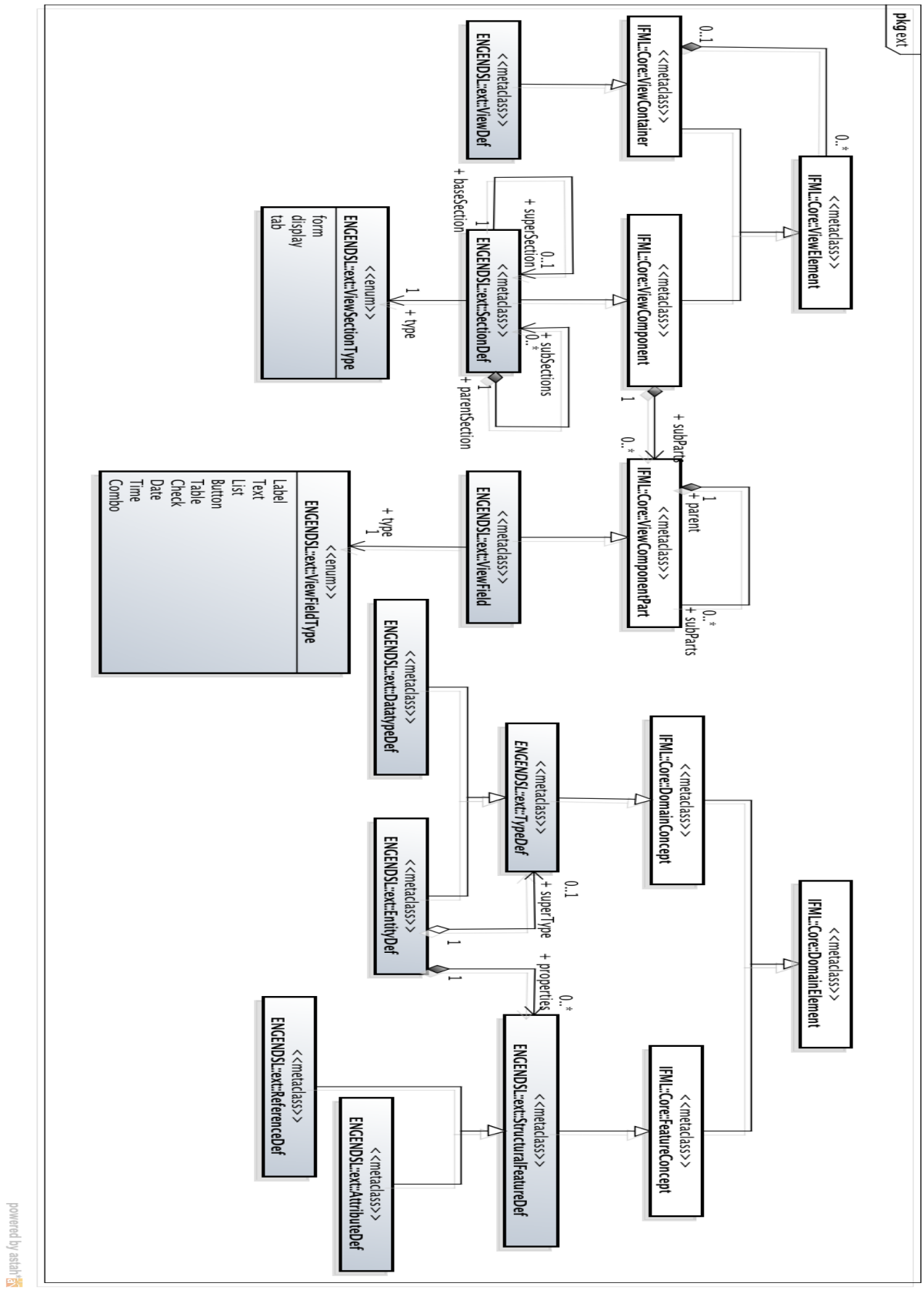
- *Library*

Os *LayoutSection* podem ser compostos também por uma ou mais *Library*. Uma *Library* é uma definição de um componente que inclui referências a bibliotecas *JavaScript* ou arquivos CSS dentro de uma seção. Isto permite que o analista especifique diferentes bibliotecas para cada seção. Para isto, um bloco de URLs é definida na *Library*. Exemplos de sua declaração foram mostradas na Figura 25 na seção de definição do *Layout*.

4.4.4 Aderência ao padrão IFML

Conforme mencionado anteriormente, a *EngenDSL* foi projetada para estar em conformidade com o padrão IFML. Os principais elementos do modelo da *EngenDSL* que estão mapeados para os elementos da IFML são mostrados na Figura 31.

Figura 31 - Metamodelo da *EngenDSL* e suas extensões ao padrão IFML



Fonte: Autoria própria.

Na Figura 32 é mostrado um mapeamento dos conceitos da *EngenDSL* para os conceitos presentes no padrão IFML para o melhor entendimento das relações entre os nomes dos conceitos presentes na *EngenDSL* e IFML. Alguns conceitos da *EngenDSL* não estão presentes na IFML. Também, conforme se pode notar, alguns elementos da IFML não possuem contrapartida direta na *EngenDSL*.

Figura 32- Tabela de mapeamento entre os conceitos da *EngenDSL* e IFML

IFML Concept	EngenDSL Core Concepts
ViewContainer	ViewDef
ViewComponent	SectionDef
ViewComponentPart	ViewField
Event	ViewEventDef
ActivationExpression	FieldDecoratorDef
Module	ModuleDef
InputPort, OutputPort	-
DomainConcept	TypeDef
	EntityDef
	DatatypeDef
FeatureConcept	StructuralFeatureDef
	AttributeDef
	ReferenceDef
Action	ControllerDef
Navigation Flow	LogicalPath
Data Flow	TargetDef
Parameter, Parameter Binding, Parameter Binding Group	ViewFieldPropertyDef, ViewFieldName
-	LayoutDef
-	LayoutSectionDef
-	LayoutStyleDef

Fonte: Autoria própria.

4.5 ENGENSBVR

Regras de negócio são requisitos que necessitam ser expressos de alguma forma e, preferivelmente, mantidos pelos praticantes do negócio. Estas regras geralmente são especificadas utilizando documentos texto em que o usuário, ou interessado no software que está para ser construído ou modificado, expressa suas expectativas em relação ao negócio gerido pelo sistema de informação, utilizando para isto linguagem natural. Como estes documentos são desenvolvidos por analistas de negócios e usuários, que geralmente não têm preocupação com um rigor formal, é necessário que sejam depois traduzidas por técnicos de tecnologia da informação para alguma notação formal, tornando-as assim passíveis de processamento computacional. Devido à ambiguidade inerente a linguagem natural, esta tradução geralmente introduz inconsistências entre o desejo original e o que foi traduzido, além de apresentar falta de um maior rigor semântico (NJONKO; EL ABED, 2012). Portanto, é desejável que estas regras sejam expressas em uma notação formal, sem, entretanto, perder sua característica de ser produzida, entendida e mantida por usuários e analistas de negócio.

Segundo Bajec e Krisper (2005a), pesquisadores e profissionais de tecnologia da informação já reconheceram a importância do tratamento de regras de negócio explicitamente durante o desenvolvimento de sistemas de informação para garantir maior agilidade quando mudanças são necessárias. A falta de tratamento explícito de regras de negócio durante o processo de desenvolvimento, visando propiciar a rápida incorporação de mudanças em regras de negócio nos sistemas de informação, pode trazer diversos problemas, tais como: incapacidade de manter-se coerente com a realidade do negócio, falta de documentação das regras, dificuldade de localização e manutenção das regras que estão diretamente definidas e espalhadas pelo código-fonte da aplicação e sem um repositório adequado de regras de negócio (BAJEC ; KRISPER, 2005b). Segundo estes mesmos pesquisadores, uma solução para resolver o problema da representação de regras de negócio diretamente nos sistemas de informação, seria exatamente o uso de uma abordagem de desenvolvimento dirigido a modelos, porém a falta de ferramental adequado para geração de código baseado nas regras de negócio dificulta sua adoção.

O padrão SBVR mostra uma abordagem para a documentação e formalização de regras de negócio baseado em um modelo formal, de forma que os modelos expressos baseados em seu metamodelo possam ser processados automaticamente por ferramentas de software. Tal metamodelo permite a modelagem os conceitos de negócio, criando esquemas conceituais (KLEINER; ALBERT; BÉZIVIN, 2009). Para isto, implementa como forma de

representação do modelo uma Linguagem Natural Controlada, tornando as regras de negócio passíveis de processamento direto quando são expressas em conformidade com seu metamodelo. Entretanto, não é possível utilizar somente o padrão SBVR para implementar regras de negócio diretamente nos sistemas de informação, pois o mesmo fornece simplesmente meios para descrever regras de negócio sem abordar sua implementação (NJONKO ; EL ABED, 2012).

Um fator que foi considerado na escolha do padrão SBVR, para ser utilizado neste trabalho como meio de modelar e documentar regras de negócio, foi a possibilidade da verbalização das mesmas através de uma linguagem natural controlada, contribuindo para a melhoria de comunicação entre as áreas de negócio e TI, e evitando a tradicional abordagem da utilização de modelos UML ou documentos definição de regras de negócio sem um maior rigor formal. Outrossim, o fato de ser um padrão definido pela OMG e ser compatível com os conceitos de desenvolvimento dirigido a modelos, especificamente o MDA, fez deste padrão uma escolha coerente para o direcionamento deste trabalho.

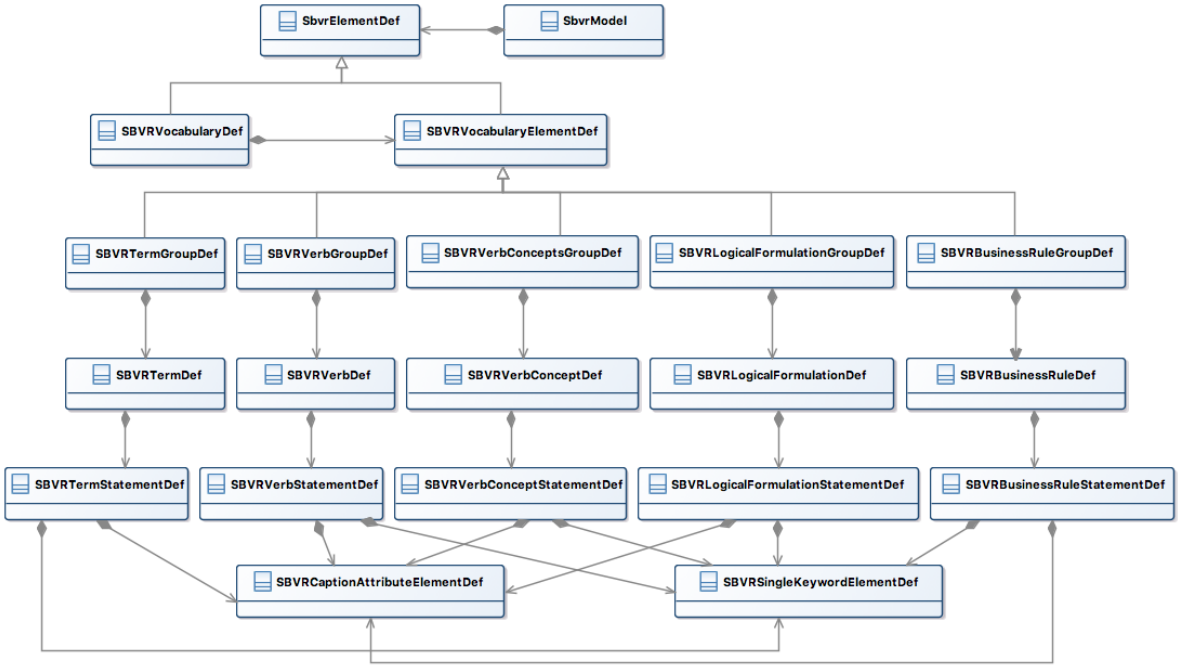
Outros estudos nesta área (JESUS 2013; SUL et al. 2011) já mostraram as vantagens e desvantagens de representações de conceitos de negócio em SBVR e em outras representações. Como uma das premissas deste trabalho é a utilização de uma DSL textual, para a representação dos modelos de esquemas conceituais para o domínio da aplicação, o modelo do SBVR é adequado às necessidades deste trabalho.

Neste trabalho, são abordadas regras estruturais e operacionais, permitindo que sejam diretamente definidas por agentes de negócio de forma textual para influenciar o comportamento de sistemas de informação.

4.5.1 Metamodelo da EngenSBVR

O modelo do padrão SBVR define alguns conceitos básicos sobre termos, conceitos verbais e regras de negócio já discutidos no Capítulo 2. O modelo proposto, além destes conceitos, contribui com novos conceitos necessários para permitir a definição de linguagens controladas baseadas no SBVR, como por exemplo, o conceito – *Logical Formulation Sets*. Os conceitos desse metamodelo são discutidos nesta seção e nas seções 4.5.2 e 4.5.3. A Figura 33 mostra o metamodelo *EngenSBVR* proposto, que estende os conceitos do padrão SBVR.

Figura 33 - Metamodelo da *EngenSBVR*



Fonte: Autoria própria.

Na Figura 33, o modelo do vocabulário SBVR na linguagem proposta é composto de elementos representados pelo elemento do modelo – *SbvrVocabularyElementDef*. Na *EngenSBVR*, estes elementos representam um grupo de conceitos dentro do vocabulário. Assim, no modelo proposto, cada entrada do vocabulário irá pertencer a um grupo como o *SbvrTermGroupDef*, *SbvrVerbGroupDef*, *SbvrVerbConceptGroupDef* e assim por diante. Isto permite estruturar conceitualmente os termos e verbos do vocabulário de negócio, o que contrasta com a abordagem sugerida na SBVR-SE em que os elementos são apresentados desconectados e aleatoriamente em um documento SBVR-SE criado pelo usuário, dificultando sua categorização.

Estes grupos são equivalentes ao conceito de *RuleEntry Set* no SBVR-SE só que representam um grupo de conceitos do vocabulário. Eles agrupam termos, conceitos verbais, definição de formulações lógicas e outras palavras-chave do vocabulário que está sendo definido. Todas as definições presentes em documentos SBVR no projeto contribuem para o vocabulário e regras.

Cada um dos grupos (de termos, verbos, etc.) *SbvrXXXGroupDef* possui uma entrada correspondente chamada *SbvrXXXDef*. Os últimos representam as entradas do vocabulário, palavras-chave da linguagem ou regras. Por sua vez, cada uma destas entradas são formadas por uma ou mais palavras e são representadas pelo elemento *SbvrXXXStatementDef*. Na

EngenSBVR, essas sentenças sintaticamente devem começar com um traço "-" seguido da representação primária do conceito que está sendo definido, dentro de seu respectivo grupo (*TermSet*, *VerbConceptSet*, *RuleSet* e *LogicalFormulationSet*). Cada sentença de definição de um conceito pode ser acompanhada de uma série de atributos chamados *SbvrCaptionAttributeDef* que representam os atributos de conceitos da SBVR. A Figura 34 mostra um exemplo de conceitos de um sistema de compras online para servir de exemplo às definições desta seção.

Figura 34 - Exemplo de conceitos de um sistema de compras *Online* definidos na *EngenSBVR*

```
TermSet {
    - order
        Definition: A instruction for supply goods
        Synonym: orders
    - product
    - payment method
}

VerbConceptsSet {
    - order is for product
    - order has product
    - order has payment method
}

RuleSet {
    - [R1] It is obligatory that each order
        has at least one product
    - each order has exactly one payment method
    - It is obligatory that each order has at least
        one payment method
}
```

Fonte: Autoria própria.

- *Termos*

No modelo proposto os termos do padrão SBVR são representados em um conjunto de sentenças de definição de termos que irão depois ser utilizados por conceitos verbais e regras.

No padrão SBVR estes termos são chamados de "*Noun Concepts"*, e são utilizados para expressar um único significado sobre algum conceito do domínio do negócio. Conforme já mencionado, as sentenças de definição dos conceitos podem estar acompanhadas de atributos que agregam alguma informação ao conceito ou definem seu tipo. No exemplo da Figura 34, o termo *order* possui dois atributos que adicionam informação aos conceitos. Além do atributo da definição do conceito (*Definition*) propriamente dita, que é uma frase sem rigor formal e não influencia na formulação das regras, está sendo indicado um sinônimo para o conceito *order*, o que faz com que o *parser* trate ambas as representações (*order* e *orders*) como o mesmo conceito. Esta forma de representação é similar à adotada no padrão SBVR.

- *Conceitos verbais*

De acordo com o padrão SBVR, conceitos verbais geralmente definem alguma relação entre os elementos do vocabulário. Uma vez que esta relação é definida no grupo de conceitos verbais ela pode ser utilizada adequadamente nas regras de negócio. Neste trabalho são considerados dois tipos de conceitos verbais – os conceitos verbais binários e os conceitos verbais unitários, ambos já discutidos no Capítulo 2.

Por exemplo, o conceito verbal *"order has product"* define que os conceitos nominais "*order"* e "*product"* possuem uma relação de dependência do tipo *conceito-possui-conceito*.

- *Regras de Negócio*

O padrão SBVR permite que analistas de negócio possam expressar as regras do negócio por meio de uma linguagem natural controlada. No caso deste trabalho, esta tarefa deverá ser realizada por meio da *EngenSBVR*. Na *EngenSBVR*, as regras devem estar sempre agrupadas em um *RuleSet*. Podem haver diversos *RuleSets* no projeto. As formulações lógicas do SBVR permitem a modelagem de conceitos como obrigação, necessidade e quantificação, por exemplo, já explicados no Capítulo 2. Uma vez que os termos, conceitos verbais e formulações lógicas estejam definidos, as regras podem ser criadas combinando-os. No exemplo da Figura 35, a formulação lógica de obrigatoriedade do SBVR-SE *"It is obrigatory that"*, está combinada com o quantificador universal *"each"* e o quantificador existencial

unitário *"exactly one"*, além do conceito verbal *"order has product"* (que por sua vez combina os termos *order* e *product* com o verbo *has*) para representar um conceito de uma regra de negócio.

Figura 35- Exemplo de uma regra de obrigatoriedade na EngenSBVR

Fonte: Autoria própria

4.5.2 Produção de Linguagens SBVR

O *SBVR Structured English* – SBVR-SE, é uma linguagem, definida no padrão SBVR, como um exemplo de uma linguagem natural controlada para expressar o vocabulário e regras de negócio em formato textual. Embora seja possível diversas representações dos conceitos do SBVR, o SBVR-SE tornou-se a notação mais utilizada para modelar conceitos de negócio baseado no padrão. Vale ressaltar que o padrão SBVR é baseado em lógica formal de primeira ordem, e, portanto, não está atrelado a nenhum tipo de representação específica.

A notação do SBVR-SE, conforme mostrado no padrão, possui um conjunto de palavras-chave para expressar os conceitos de modalidades lógicas como a modalidade de obrigação – *"It is obligatory that"*, ou então um quantificador universal – *"Each"*. Como a maioria das implementações de representação dos conceitos do SBVR utiliza o SBVR-SE, estes modelos e ferramentas estão atrelados a estas palavras-chave tornando difícil a representação destes conceitos em outros idiomas que não o inglês. Por exemplo, uma regra que torna obrigatório que "*todo chamado de suporte deve possuir exatamente uma data de abertura*" em português, teria que ser escrita desta forma: "*It is obligatory that each* chamado de suporte *must* especificar *exactly one* data de abertura" no SBVR-SE. Possivelmente, por este motivo, não existem muitas implementações deste padrão em sistemas brasileiros. Esta limitação já foi identificada em trabalhos anteriores (JESUS, 2013). Além disto, o SBVR-SE não é o único padrão aceito para escrever regras de negócio com uma linguagem natural controlada na forma textual. O *RuleSpeak Business Rule Notation* (ROSS, 2009) é um padrão conhecido, mas que usa um conjunto de palavras-chave diferente, por exemplo, a mesma regra em *RuleSpeak* seria escrita assim: "*Each* chamado de suporte *must* especificar *exactly one* data de abertura". Ainda assim, a representação do modelo e a ferramenta foram escritas para o idioma inglês e é necessário um grande esforço de tradução da ferramenta para que possa ser utilizada em outros idiomas de forma satisfatória. Portanto, é necessária a criação de

um modelo, baseado no SBVR, que permita a especificação de novas linguagens controladas, para representar o metamodelo do padrão SBVR, em qualquer idioma. Espera-se que com o modelo sugerido neste trabalho, seja possível a especificação de novas linguagens naturais controladas em qualquer idioma ou representação baseada no SBVR.

O principal elemento do modelo envolvido na produção de novas linguagens utilizando a *EngenSBVR* é o *SbvrLogicalFormulationStatementDef*. Este elemento permite ao designer da linguagem substituir as palavras chaves da SBVR-SE por outras que expressem o mesmo sentido, mas com uma diferente representação. No exemplo da Figura 35 os elementos que formaram as palavras chaves permitidas nas Regras de Negócio não foram mostrados. A Figura 36 mostra como estes conceitos e seus significados são definidos na *EngenSBVR*.

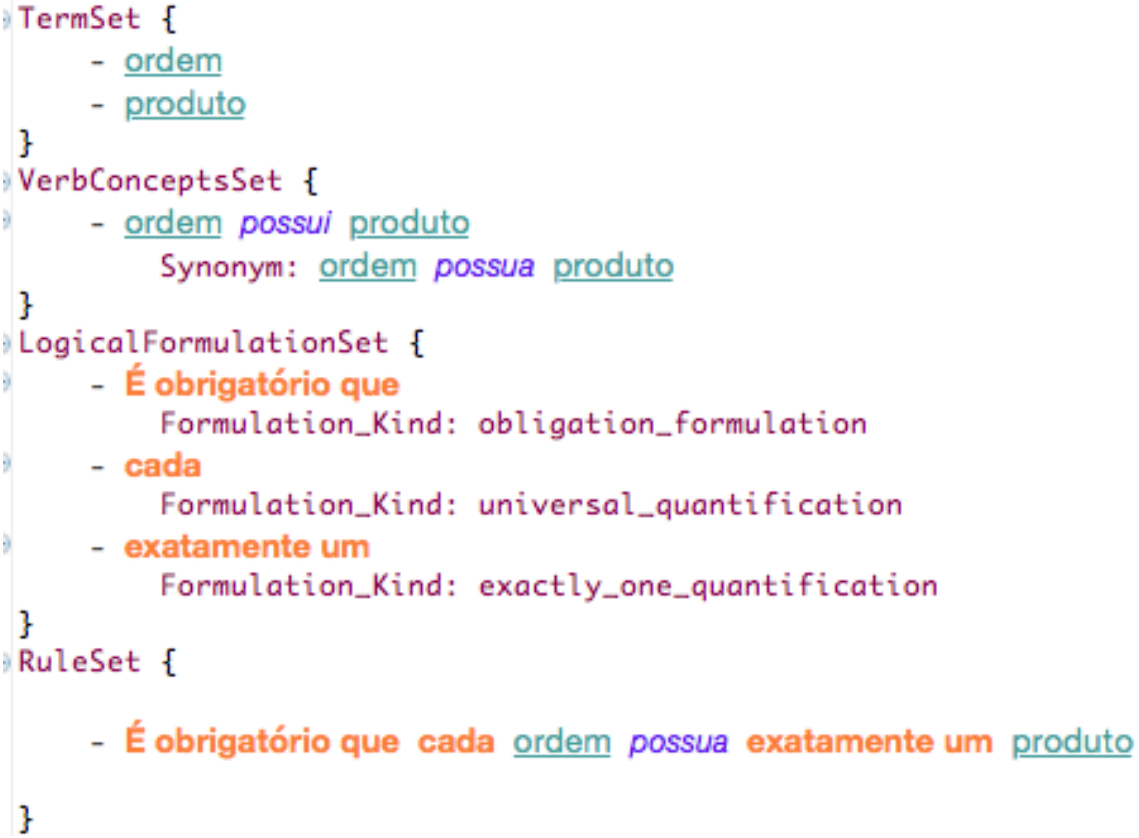
Figura 36 - Exemplo de formulações lógicas da EngenSBVR

```
LogicalFormulationSet {
    - It is obligatory that
        Formulation_Kind: obligation_formulation
    - each
        Formulation_Kind: universal_quantification
    - at least one
        Formulation_Kind: at_least_one_quantification
    - exactly one
        Formulation_Kind: exactly_one_quantification
}
```

Fonte: Autoria própria.

O atributo *Formulation_Kind* do *EngenSBVR* é utilizado para definir que tipo de formulação lógica a sentença define. Assim, "*It is obligatory that*" em qualquer regra de negócio neste projeto, significa uma formulação lógica modal de obrigatoriedade. Diferentes formulações podem ser definidas para a mesma forma modal da SBVR. Por exemplo, a representação destes conceitos e das formulações de modalidade podem ser descritas conforme a Figura 37 no idioma português.

Figura 37 - Exemplo de formulações lógicas e regras de negócio na *EngenSBVR* em português



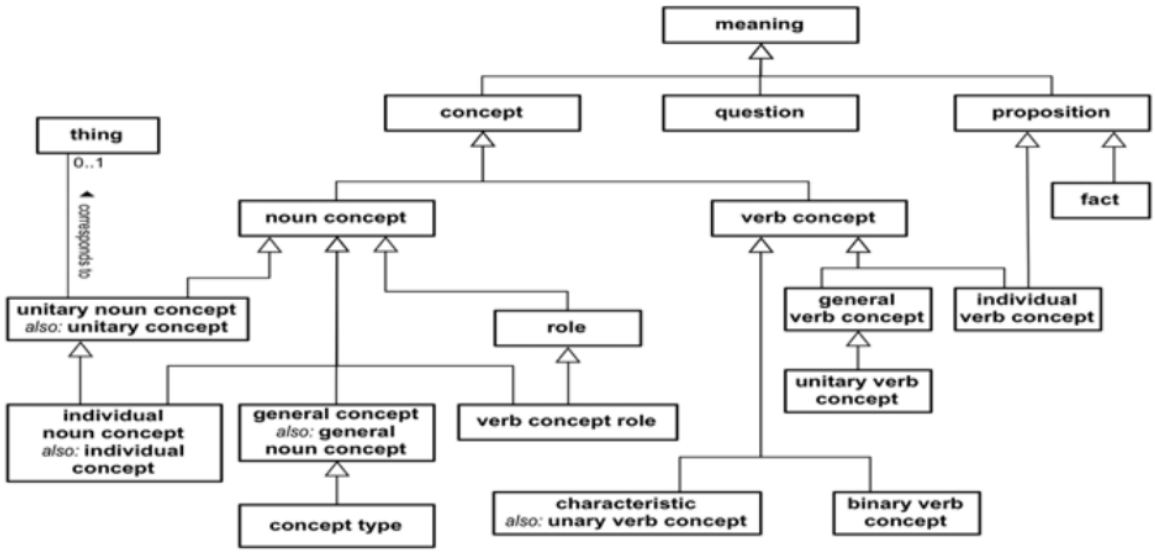
Fonte: Autoria própria.

Uma das contribuições da *EngenSBVR* é permitir que o editor *EngenSBVR* seja utilizado não só para representar conceitos no padrão SBVR por meio do SBVR-SE, mas para criar representações em outros idiomas diferentes do inglês, conforme mostrado na Figura 37. Esta característica do editor, e seus componentes, o diferencia de outras abordagens que trazem codificado no editor as palavras-chaves das formulações lógicas do SBVR, impedindo assim que novas representações de formulações sejam criadas naturalmente no idioma desejado. Por exemplo, para expressar uma formulação lógica de necessidade em português nos editores e trabalhos pesquisados (RAJ; PRABHAKAR; HENDRYX, 2008; NEMURAITE et al. 2010; KAMADA; GOVERNATORI; SADIQ, 2010; MARINOS; GAZZARD; KRAUSE, 2011; AGRAWAL ; SINGH, 2009), a necessidade *"É necessário que cada compra possua uma forma de pagamento"* ficaria da seguinte forma *"It is necessary that each compra possua one forma de pagamento"*, ou ainda *"It_is_necessary_that each compra possua one forma de pagamento"*. A partir da revisão da literatura, não se identificou até o momento um editor SBVR multilíngue e que permita dinamicamente, ou seja, sem esforço de codificação, a especificação de novas linguagens baseadas no SBVR.

4.5.3 Elementos do metamodelo

O metamodelo definido pela *EngenSBVR* foi mostrado na Figura 33. A Figura 38 mostra o metamodelo da SBVR (seção 8.1 SBVR, 2013) que define os principais elementos tratados nesta proposta.

Figura 38 - SBVR Meaning and Concepts Metamodel



Fonte: SBVR (2013).

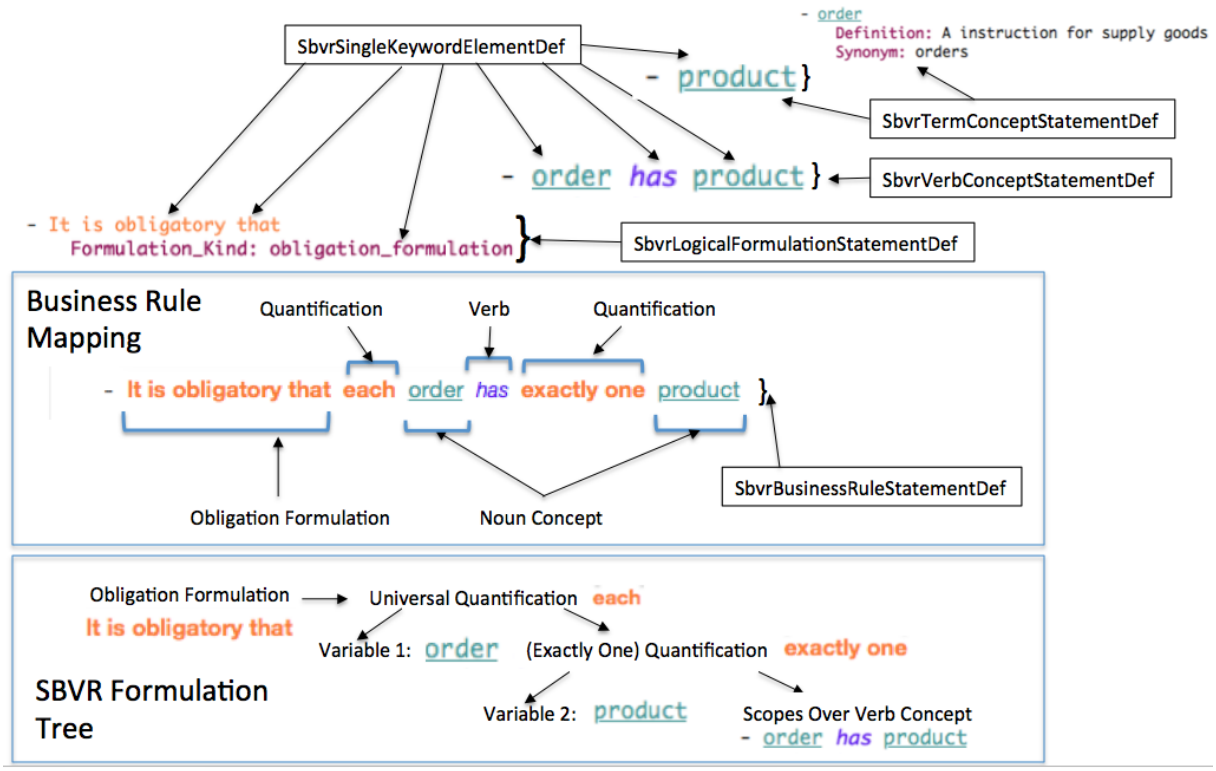
O metamodelo definido na *EngenSBVR* é fortemente baseado nos conceitos do metamodelo do padrão SBVR para permitir que os elementos definidos no mesmo, sejam facilmente descritos pelos modelos construídos utilizando a linguagem. Entretanto, devido às características específicas de um *parser* para uma DSL, que precisa processar e prever todos os tokens textuais, estudos anteriores (RAIMUND et al., 2008) indicam que é uma abordagem equivocada tentar definir diretamente na linguagem todos os conceitos do metamodelo da SBVR em qualquer domínio de negócio, principalmente para conceitos que em sua representação trazem espaços (conceitos verbais ou conceitos compostos). Por exemplo, se fosse criada uma palavra-chave na linguagem para o formulador lógico de necessidade – *"É necessário que"* ficaríamos impedidos de utilizar o sinal *"É"* em outro contexto, pois o parser irá esperar obrigatoriamente um "n", seguido do restante da palavra-chave *"ecessário que"*. Isto pode gerar conflitos entre palavras-chaves que devem ser interpretadas pelo *parser* com algum significado especial, e palavras que devem ser consumidas sem agregar significado. Tal dificuldade possivelmente norteou outras implementações – (NEMURAITE et al. 2010; AGRAWAL ; SINGH, 2009) a utilizar alguma forma de conector para os diversos termos

separados por espaço, como por exemplo, o símbolo de sublinhado como no exemplo *"It_is_necessary_that"*.

Desta forma, optou-se por mapear os elementos básicos do padrão SBVR, mas deixar a construção e identificação propriamente dita dos elementos ocorrerem em fases de análise, posteriores à léxica e do *parser*, na fase de validação dos elementos do modelo. Uma das vantagens trazidas por essa abordagem é a possibilidade de representar palavras-chaves do modelo com qualquer representação textual, o que permite um aspecto multilíngue para a *EngenSBVR*. Uma possível desvantagem deste modelo é a complexidade da implementação e a necessidade de incluir uma biblioteca que contenha as palavras-chave do padrão já definidas antes da construção de qualquer modelo baseado no SBVR.

A Figura 39 mostra o modelo parcial da DSL quando o modelo é interpretado pelo *EngenSBVR Editor*, que é representativo para o entendimento da linguagem.

Figura 39 - Modelo da *EngenSBVR* mapeado para o SBVR na definição do vocabulário e regras de negócio



Fonte: Autoria própria.

Conforme pode ser visto na Figura 39, o modelo produzido utilizando a linguagem SBVR possui diversos elementos. Cada *Tokén* isoladamente representa uma entrada para uma palavra-chave, no modelo representado pelo elemento *SbvrSingleKeywordElementDef*, que

não possui um significado no SBVR. Entretanto, como estes elementos estão localizados dentro de grupos no modelo da *EngenSBVR*, é possível classifica-los corretamente. Um ou mais elementos deste tipo são utilizados para a definição de sentenças (*SbvrTermConceptStatementDef* e *SbvrVerbConceptStatementDef*, por exemplo), no vocabulário ou regras de negócio SBVR. Na *EngenSBVR*, estes elementos não representam entradas do vocabulário – como ocorre na da linguagem de exemplo SBVR-SE (*SBVR Structured English*) do padrão (OMG, 2013). Em vez disso, eles representam um grupo de entradas do vocabulário, no sentido em que cada entrada real do vocabulário pertence a uma categoria, ao invés de um elemento isolado no documento. Na especificação SBVR estes grupos são o equivalente ao conceito de *Rule Entry Set* (SBVR, 2013, p.251), mas representam um conceito genérico definido na *EngenSBVR* para agrupar conceitos específicos, como regras de negócio, termos, conceitos verbais e palavras-chave.

Para transformar o modelo EMF (NATALI, 2012) da *EngenSBVR* no modelo conceitual do SBVR, o *EngenSBVR LF Parser*, que é detalhado na Seção 4.6.3, é utilizado. Este componente identifica os elementos lógicos do SBVR (parte central da Figura 39) em uma sentença de regras de negócio (*SbvrBusinessRuleStatementDef*). Após a identificação, o componente faz o mapeamento dos elementos identificados no modelo da *EngenSBVR* para o modelo do padrão SBVR, criando a árvore de conceitos de formulações lógicas do SBVR (parte inferior da Figura 39), para cada regra de negócio.

Apesar de que se pode editar diretamente os arquivos da *EngenSBVR*, ou *EngenDSL*, utilizando qualquer editor de texto, o *XText* cria editores básicos para cada linguagem, embora os mesmos devam ser personalizados para que ofereçam serviços mais especializados para cada linguagem. A Seção 4.6 detalha os editores construídos na solução proposta.

4.6 EDITORES DA PLATAFORMA

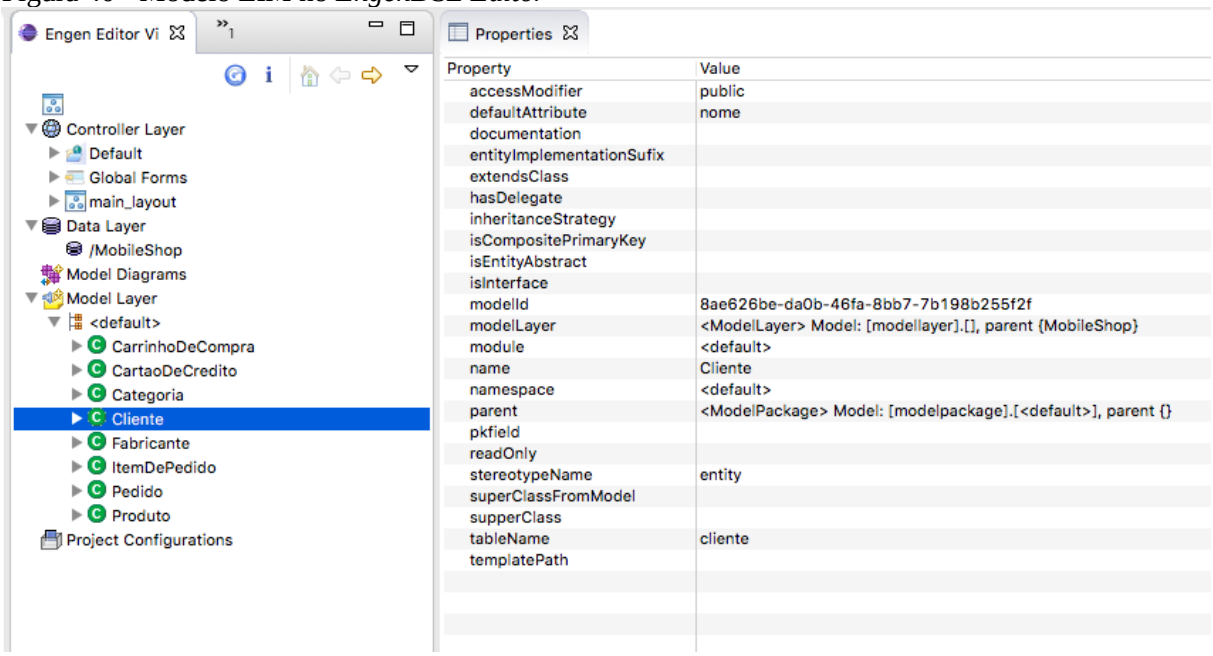
Pode-se criar modelos da *EngenDSL*, ou *EngenSBVR*, utilizando qualquer editor de texto. Entretanto, para facilitar a manipulação dos modelos utilizando as DSL propostas, tornando-as atrativas para serem utilizadas por desenvolvedores, analistas de sistemas e de negócio conforme os requisitos descritos na introdução deste capítulo, foi necessária a personalização dos editores básicos criados para ambas as linguagens pelo *XText Framework* (XTEXT, 2015). A partir do modelo EMF (NATALI, 2012) gerado pelo *XText*, é possível utilizá-lo como fonte para prover serviços de auto-complemento de termos, validação semântica e personalização da interface do usuário.

O *XText Framework* gera um editor básico com algumas funcionalidades de validação sintática e busca de referências. Entretanto, isto não é suficiente para a maioria das linguagens mais complexas, como as que foram criadas para este trabalho. Por exemplo, estes editores básicos não resolvem problemas de como validar se uma regra de negócio, escrita em SBVR, está semanticamente correta em relação a este modelo, de como atribuir diferentes estilos gráficos a diferentes partes do discurso de uma regra de negócio SBVR, ou de como definir se uma propriedade do modelo de negócio é válida como campo de uma *View*. Desta forma, foi necessário a fatoração da análise semântica do modelo EMF gerado pelo *XText*, para ambas as linguagens, e personalização dos editores de acordo com o metamodelo da *EngenSBVR* e *EngenDSL*. Esta seção explica os componentes que precisaram ser criados para permitir uma melhor experiência do usuário ao utilizar estas linguagens.

4.6.1 Editor da EngenDSL

O editor da *EngenDSL* construído neste trabalho, permite que o usuário defina de forma textual (gerado pelo *XText*) elementos do modelo EMF por meio da DSL, e elementos os elementos do modelo EIM visualmente, após a primeira fase de transformação dos modelos. A manipulação dos modelos permite que os valores de elementos do modelo sejam personalizados antes da fase final de geração. Na Seção 4.4, a *EngenDSL* e os principais elementos de seu metamodelo foram apresentados. As Figuras 40 e 41 mostram exemplos do editor visual de propriedades do modelo EIM, gerado a partir de um modelo hipotético da *EngenDSL*, ao lado da árvore de conceitos do EIM à esquerda.

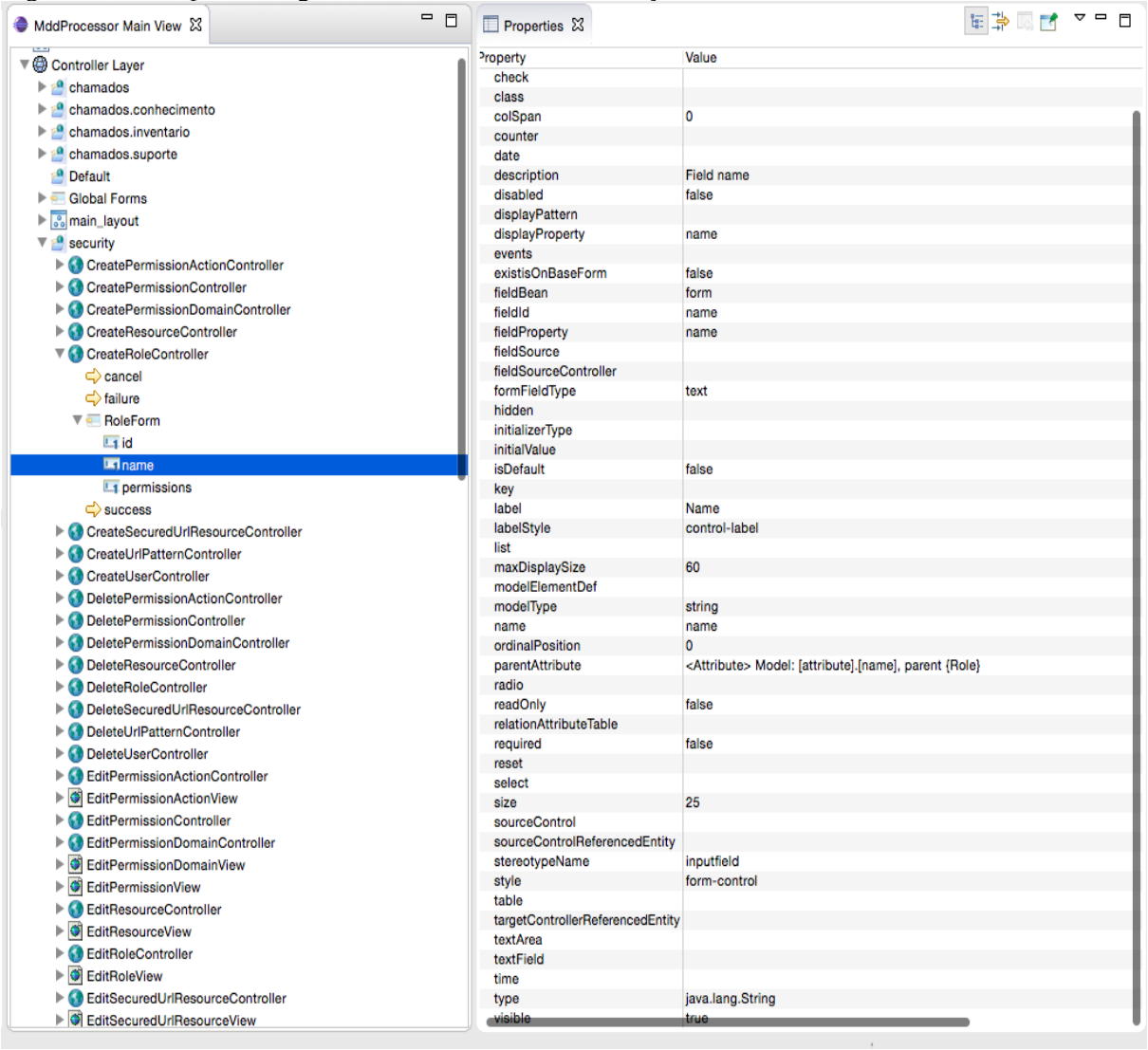
Figura 40 - Modelo EIM no *EngenDSL Editor*



Fonte: Autoria própria.

O editor permite a personalização do modelo EIM carregado a partir de um arquivo XML conforme descrito na Seção 4.2.

Figura 41- Exemplo do *EngenDSL Editor* com a visualização de um EIM



Fonte: Autoria própria.

Neste editor, à esquerda é mostrado um exemplo de uma árvore de conhecimento do modelo EIM gerado após a primeira transformação M2M da *EngenDSL*. Esta árvore mostra os conceitos do EIM separados por camadas da aplicação onde cada tipo de componente é alocado. Conforme se pode ver nestes exemplos, os *controllers* estão agrupados na camada do modelo chamada *"ControlLayer"*. Esta definição é consistente com o modelo IFML que define módulos arquiteturais para a solução. À direita estão as propriedades do nó do modelo EIM selecionado. Tais propriedades são personalizáveis por meio do editor e incorporadas ao processo conforme explicado na Seção 4.3.

Uma vez selecionado um elemento do EIM (propriedade "*name*" do "*RoleForm*" do "*CreateRoleController*" no caso da *Figura 41*), é possível personalizar os valores dos atributos do EIM, e inclusive adicionar novas definições para um conceito. Estes valores de

personalização são armazenados separadamente do modelo EIM, conforme descrito na Seção 4.3, para que na próxima geração M2M eles não sejam sobrescritos.

4.6.2 Editor da EngenSBVR

O *EngenSBVR Editor*, proposto nesta dissertação, é um dos componentes da *Plataforma Engen* (mostrada na Figura 10), que assim como a *EngenDSL Editor*, é baseado no *Framework XText* (XTEXT, 2016), e que permite que seja criado um metamodelo de uma DSL. O editor proposto para a linguagem *EngenSBVR*, possui funcionalidades como validação sintática, semântica, assistência de conteúdo, visualização e representação da sintaxe de acordo com os requisitos definidos na introdução deste capítulo.

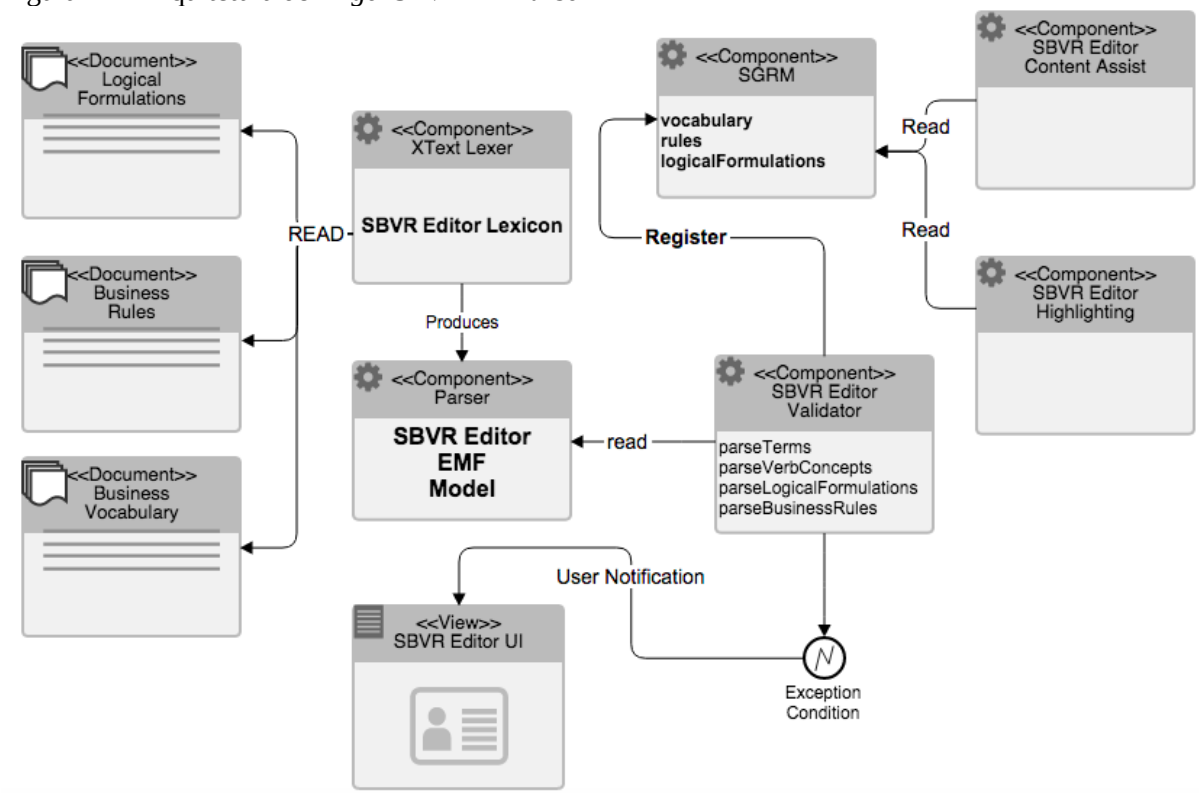
O metamodelo da *EngenSBVR* foi mostrado na Figura 33 e discutido nas Seções 4.5.2 e 4.5.3, e representa as principais características do modelo SBVR que o editor suporta. Conforme os requisitos definidos na introdução deste capítulo, o *EngenSBVR Editor* suporta a facilidade de visualização dos conceitos do padrão SBVR definidos conforme o metamodelo da *EngenSBVR*. O componente de coloração da sintaxe da *EngenSBVR – EngenSBVR Editor Syntax Highlight*, é um componente criado para implementar, de forma independente do idioma, um esquema de cores semelhante à definida para o SBVR-SE. As Figuras 37 e 43 mostram exemplos do reconhecimento dos termos, conceitos verbais e formulações lógicas da SBVR, por meio da formatação e coloração das fontes no editor realizada por este componente. Esse componente é uma personalização do componente padrão gerado pelo *XText*, e é responsável pela detecção dos elementos do modelo válidos em cada contexto de um documento da *EngenSBVR,* e mudará a fonte apresentada no editor conforme a seguinte definição: palavras-chave de formulações lógicas aparecerão em tonalidade laranja e em negrito; os conceitos verbais serão marcados em azul e itálico; conceitos nominais são marcados em verde e sublinhados e palavras-chaves da linguagem aparecem em tom magenta. Todas essas formatações podem ser alteradas no editor por meio da seção de configurações da plataforma *Eclipse*.

4.6.3 Análise Sintática e Semântica de Formulações Lógicas da SBVR

O processamento de formulações lógicas da SBVR definidas nas regras de negócio é necessário para a interpretação do sentido da regra. Uma abordagem similar à descrita por Selway, Mayer e Stumptner (2014) foi utilizada para modularizar a interpretação do sentido

das formulações lógicas nas regras de negócio. A Figura 42, mostra esquematicamente os subcomponentes do *EngenSBVR LF Parser* que é o responsável pelo processamento das regras de negócio na arquitetura proposta nesta dissertação, sendo os principais o *XText Lexer*, o *SBVR Editor Validator*, o SGRM (*SBVR Global Registry Module*), além dos componentes auxiliares do editor: *SBVR Editor Content Assist* e *SBVR Editor Highlight*.

Figura 42 - Arquitetura do *EngenSBVR* LF *Parser*



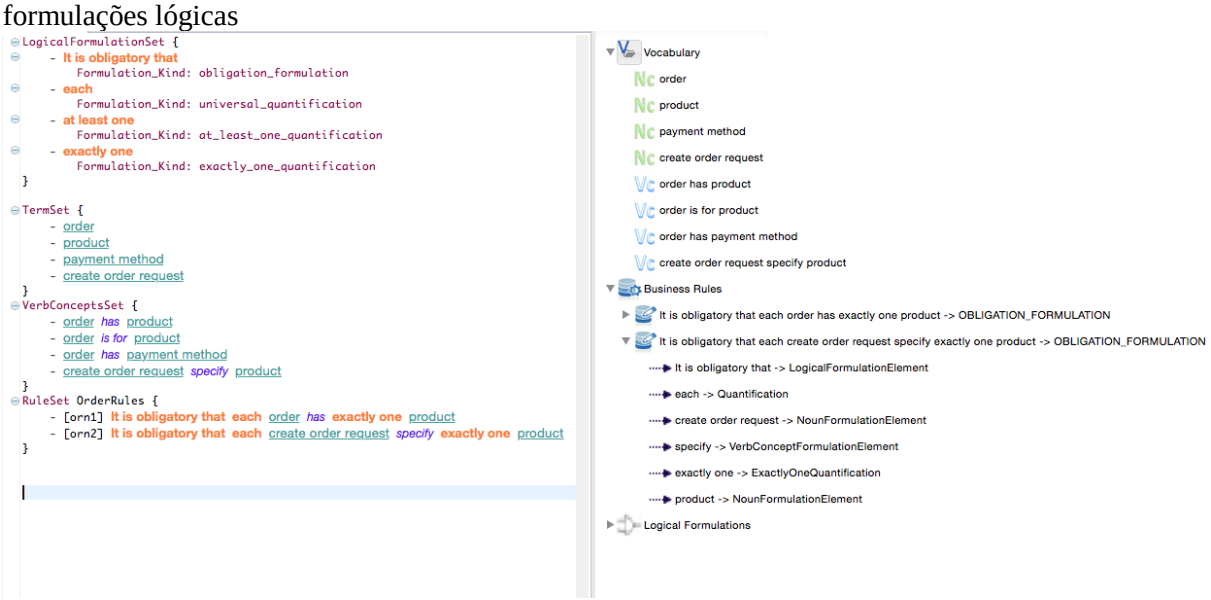
Fonte: Autoria própria.

O *Engen SBVR LF Parser* é um analisador sintático e semântico (*parser)* das formulações lógicas da SBVR que cria uma estrutura de dados do tipo árvore para as regras de negócio definidas textualmente baseadas na *EngenSBVR*. Como mostrado na Figura 42, a arquitetura escolhida para o parser é dividida em módulos. O primeiro componente é o *XText Lexer* (componente produzido automaticamente pelo *XText Framework*) que realiza o trabalho de criar *tokens* com as palavras definidas na representação textual das regras de negócio definidas utilizando a *EngenSBVR*, e produzem um modelo chamado *SBVR Editor Lexicon* (dicionário de conceitos do domínio). Estes elementos irão formar o modelo *SBVR Editor EMF Model*, que representa os conceitos da *EngenSBVR* em um modelo EMF, e que é lido pelo componente *SBVR Editor Validator*. Este componente é responsável pela validação

dos termos do vocabulário perante os conceitos verbais e regras de negócio. Caso alguma inconsistência seja encontrada, o *SBVR Editor Validator* notificará o usuário imediatamente, se o mesmo estiver utilizando o *EngenSBVR Editor* na plataforma *Eclipse*, ou o script de transformação será interrompido com o erro, caso o *EngenGenerator* tenha sido acionado fora da Plataforma *Eclipse*. Caso o *SBVR Editor Validator* não encontre problemas de inconsistências entre os termos, conceitos verbais e regras de negócio, o SGRM (*SBVR Global Registry Module*) será atualizado com os conceitos do modelo da *EngenSBVR*, que ficará disponível em memória para a *Plataforma Eclipse*, ou para o *EngenGenerator*, em um processo de transformação. Este módulo (SGRM), fica disponível para outros componentes da arquitetura do editor como o assistente de conteúdo e de formatação das fontes utilizadas no editor para representar os conceitos de negócio na *Plataforma Eclipse*. Este processo é mostrado esquematicamente na Figura 42.

As entradas no SGRM são elementos léxicos para o modelo e são de três tipos: *Vocabulary* – entradas que correspondem a termos e conceitos verbais no modelo; *Rules* – entradas regras correspondentes a regras de negócio depois de serem processadas e *Logical Formulation Phrases* que correspondem à saída do módulo *Logical Formulation Mapping*. Estes elementos podem ser visualizados no editor gráfico da SBVR ilustrados na Figura 43, que mostra o resultado da análise gramatical realizada em tempo real, que ocorre enquanto o usuário insere as regras de negócio.

Figura 43 - *EngenSBVR Editor* - exemplo de compras *Online* e a visualização gráfica do registro de formulações lógicas



Fonte: Autoria própria.

Apesar da relativa complexidade envolvida no processamento dos elementos da *EngenSBVR*, o *parser* proposto neste trabalho possui uma complexidade menor do que aqueles para linguagem natural visto em outras abordagens (GOEDERTIER; HAESEN; VANTHIENEN, 2007; MARINOS ; KRAUSE, 2009; MARINOS; MOSCHOYIANNIS; KRAUSE, 2010; BAJWA; LEE; BORDBAR, 2011; AFREEN ; BAJWA, 2011). Isto foi possível em razão da abordagem adotada na construção do editor para *EngenSBVR*, que também é baseado em um metamodelo EMF formal (NATALI, 2012). A abordagem utilizada permite que as regras sejam validadas sempre em face ao vocabulário de negócio. Como as construções permitidas no editor têm um direto relacionamento com um metamodelo definido em *EMF*, e aderente ao metamodelo do padrão SBVR, o modelo resultante também é formal. Os exemplos mostrados nas Figuras 34, 35, 36 e 37, foram extraídos do editor SBVR construído e são de elementos de definição de um vocabulário, ou regras, utilizando o estilo de representação similar ao da SBVR-SE já comentado neste trabalho.

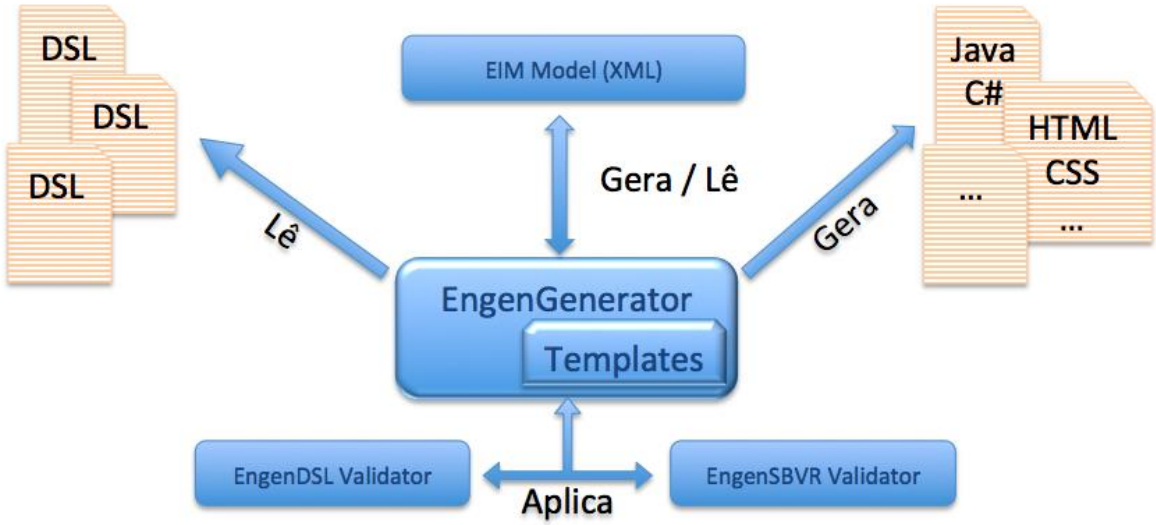
O foco desta dissertação é uma proposta de incorporação de regras de negócio descritas em SBVR em sistemas de informação baseado em um processo MDSD. Entretanto, para que a solução fosse implementada com sucesso, durante as pesquisas realizadas sobre implementação da SBVR com desenvolvimento de software dirigido a modelos em sistemas de informação, ficou claro que somente a linguagem SBVR não seria suficiente para a especificação e execução do processo completo de transformação dos modelos de regras de negócio em um sistema empresarial. O suporte de ferramental, a integração com o modelo de interação com o usuário, a definição do momento de verificação, e o suporte a múltiplos idiomas na definição das regras de negócio, são questões em aberto. O padrão SBVR fornece simplesmente meios para descrever regras de negócio sem abordar aspectos de sua implementação de como e quando serão verificadas (NJONKO ; EL ABED, 2012). Esta dissertação aborda estes problemas. Na Seção 4.5.2 foi mostrado como o modelo SBVR proposto trata a questão do suporte a múltiplos idiomas.

4.7 TRANSFORMAÇÃO ENTRE MODELOS – COMPONENTE ENGENGENERATOR

O componente da arquitetura responsável pela execução das transformações modelo-para-modelo e modelo-para-texto é o *EngenGenerator*. Conforme mostrado na Figura 10, é subdividido em um conjunto de componentes encarregados das transformações modelo-para-modelo e modelo-para-texto (*M2M Engine* e *M2T Engine* respectivamente). Para cada tipo de transformação existem regras associadas (M2M *Rules* e M2T *Rules*) e *templates* para

transformação do modelo intermediário (EIM) em código. A arquitetura de transformação é mostrada na Figura 44.

Figura 44 - Arquitetura de Transformação do *EngenGenerator*



Fonte: Autoria própria.

Na Figura 44 é mostrado, esquematicamente, como o componente *EngenGenerator* se relaciona com outros artefatos do processo MDSD sugerido e componentes da arquitetura. As entradas para o componente são os arquivos de definição dos modelos do sistema descritos utilizando a *EngenDSL* e *EngenSBVR*, as configurações do projeto e opcionalmente o EIM serializado em um arquivo XML. No caso da transformação modelo-para-modelo, o resultado será o arquivo EIM atualizado de acordo com os modelos descritos nos arquivos por meio da *EngenDSL* e *EngenSBVR*, depois que os modelos tenham sido validados pelos seus respectivos validadores. Para as transformações modelo-para-código, o modelo EIM (opcionalmente com as personalizações conforme descrito na Seção 4.3) é utilizado como entrada. Os *templates* contendo regras de transformação, para cada tipo de elemento do EIM, serão utilizados para transformar o modelo em arquivos texto. Cada template define o tipo de elemento do EIM que irá tratar, e o arquivo de saída. O *EngenGenerator* utilizará as configurações do projeto para gravar o arquivo de saída indicado pelo template no local adequado.

A Figura 10 mostra um componente chamado *GeneratorTests*. Este componente é responsável por executar uma bateria de testes, para verificação da correção das regras e saídas, sempre que os *templates* são alterados.

Conforme mencionado, os *templates* são componentes que, baseados em um modelo (*Domain Model*) e levando em consideração o domínio arquitetural e tecnológico alvo da aplicação (*Java, Web, Java Server Faces, C#, ASP .NET, PHP*, etc.) gerará o código fonte correspondente. A Figura 45 mostra um trecho de código de um *template* para um campo do tipo rótulo (*label* em HTML), escrito utilizando a linguagem de *XTend* (XTEND, 2016) direcionado para a plataforma *Java/JSP*.

Figura 45 - Exemplo de template para um campo HTML

```
/**
 * Renders a label for the given {@link InputField}
 */
class FieldRender_label extends AbstractFieldRender {

    @Inject extension IterableUtils
    @Inject extension SifModelExtensions

    override parse(Map<String, ? extends StereotypedItem> elements) {
        super.parse(elements)
        '''
            <label id="«super.field.fieldId»-span" class="«super.field.style»">
                <c:out value=«super.field.outValue» default="No value found" />
                «sourceControlRenderScript»
            </label>
        '''
    }
```

Fonte: Autoria própria.

Neste exemplo, o elemento "*field*" representa um elemento do modelo EIM que é um *Field* de uma *View* descrito utilizando a *EngenDSL* conforme explicado na Seção 4.4.3.

Os *templates* provêm a base para a transformação de todos os elementos do modelo EIM para o código fonte alvo da plataforma definida. Esta metodologia provê a abordagem com uma separação entre o modelo de alto nível da aplicação e a plataforma escolhida para o projeto, permitindo que conjunto de *templates* arquiteturais, previamente criados por arquitetos de sistemas, possam ser facilmente reutilizados promovendo o reúso e independência de plataforma.

4.8 CONSIDERAÇÕES FINAIS

Um dos maiores problemas da abordagem da utilização do padrão SBVR para a especificação de regras de negócio em sistema de informação, é que o SBVR só define como expressar as regras de negócio, mas não define como, quando ou onde aplicá-las. Existem

vários tipos de regras de negócio em um sistema de informação. Neste trabalho são tratadas somente as regras de negócio do tipo que restringem entrada de dados na interface do usuário conforme descrito na introdução do Capítulo 1. Como este tipo de regras de negócio é muito comum em sistemas de informação, o escopo do trabalho foi definido para abordar somente este tipo de regras de negócio. Independente disto, a abordagem sugerida neste trabalho provê o subsídio necessário para aplicar as técnicas mostradas em outros tipos de regras de negócio. Internamente, durante a transformação final, um componente chamado *EngenDSL-SBVR-Mapping*, para cada restrição de obrigação, mapeia os conceitos de negócio, expressos em SBVR, aos conceitos do modelo de interação com o usuário. Um exemplo da aplicação deste conceito será mostrado no Capítulo 5.

Conforme apresentado nas seções deste capítulo, diversos componentes foram produzidos como parte da solução. Tais componentes são enumerados a seguir:

a) *EngenDSL* – uma DSL Textual para a criação e manutenção de regras de interação com o usuário baseadas na IFML chamada *EngenDSL*;
b) *EngenSBVR* – uma DSL Textual para a criação e manutenção de regras de negócio escritas em SBVR, que é integrada ao modelo da *EngenDSL* por meio de validadores;
c) *EngenDSLEditor* / *EngenSBVREditor* - editores para as DSLs, capazes de prover facilidades para a construção, armazenamento, visualização, recuperação e referência à regras de negócio e que permitem a definição de conceitos de interação com o usuário incluindo *layouts* e componentes de tela, bem como facilidades de usabilidade tais como validação sintática e semântica ao digitar, propostas automáticas para o complemento de regras e formatação visual, dentre outros. É importante notar que o suporte dado ao SBVR é parcial e específico para determinados tipos de regras e conceitos;
d) *EngenSBVRParser* - um *parser* para tipos específicos de formulações lógicas da SBVR;
e) associado ao editor *EngenSBVREditor*, foi criado um visualizador do modelo resultado da execução do *EngenSBVRParser* para as formulações lógicas das regras de negócio;
f) *Engen Intermediate Model* - EIM - um modelo intermediário capaz de integrar o modelo conceitual definidos através das DSLs, para posterior geração de código;
g) *EngenGenerator* - um motor de transformação capaz de realizar dois tipos de transformações: transformar o modelo EMF gerado pelas DSLs no EIM – *M2M EngenEngine*; e transformar o EIM, através de *Templates Arquiteturais*, definidos para qualquer arquitetura de software para sistemas empresariais, no código final da aplicação.

É importante ressaltar que, excetuando os editores, os componentes e bibliotecas funcionam independentemente da plataforma de desenvolvimento escolhida para o projeto, permitindo que o processo de desenvolvimento não esteja associado a uma IDE ou editores específicos pertencentes à *Plataforma Engen*.

5 EXEMPLO DE APLICAÇÃO DA ENGEN

Este capítulo apresenta um exemplo de aplicação da *Plataforma Engen* com o objetivo de ilustrar o seu funcionamento por meio da aplicação das linguagens e metodologia sugeridas neste trabalho. Este exemplo é descrito, modelado e implementado nas seções seguintes utilizando as linguagens, ferramentas e editores propostos.

5.1 INTRODUÇÃO

Esta seção apresenta a modelagem de um sistema hipotético de compras online chamado de *MobileShop*, com o intuito de avaliar a proposta deste trabalho utilizando a *Plataforma Engen*, com a abordagem MDSD sugerida, juntamente com os editores das DSLs criadas e ferramentas propostas. Uma arquitetura da aplicação sugerida é mostrada, processos de transformação baseado no modelo criado pelas linguagens e baseados nos *templates* da arquitetura sugerida serão exemplificados, é mostrado um exemplo de como os artefatos criados manualmente são integrados à solução, bem como exemplos das telas em execução em um servidor de aplicação.

Um pressuposto para a iniciação do processo de desenvolvimento utilizando a abordagem sugerida, é que os *templates* arquiteturais para a plataforma escolhida já tenham sido criados e testados. No caso deste estudo de caso, a plataforma escolhida é Java EE para *Web*. Os *templates* construídos são capazes de transformar o modelo EIM produzido pelo modelo das linguagens em artefatos de código fonte Java, arquivos de representação das *views*, testes, arquivos de configuração e *scripts* de compilação e implantação do sistema.

Os próximos passos são o levantamento de requisitos, mapeamento dos conceitos para as linguagens *EngenDSL* e *EngenSBVR*, opcionalmente o desenvolvimento de componentes manuais que serão integrados à solução e aplicação das transformações, que serão discutidos nas seções subsequentes.

5.2 REQUISITOS DO SISTEMA MOBILESHOP

O sistema deve permitir a interação tanto dos funcionários da loja interessados em manipular informações de apoio – *backend*, quanto a interação com o usuário final, interessado em realizar compras – *frontend*. Os produtos possuem um fabricante que podem existir mais de um produto oferecido na loja. O cliente pode realizar seu próprio cadastro,

selecionar produtos e adicioná-los ao carrinho de compras, assim como realizar ordem de compras. Ordens de compra só podem ser realizadas por meio de cartões de crédito. Os validadores de cartão devem comunicar-se com as operadoras por meio de componentes de software específicos já fornecidos pelas operadoras e acessados via *WebServices*. Os produtos possuem um modelo, descrição, valor, e uma figura associada que deve ser apresentada quando o usuário estiver realizando pesquisas ou compras. Os fabricantes possuem uma identificação. Os clientes, em seu cadastro, devem informar o nome, CPF e um cartão de crédito que deve possuir número e validade. Cada ordem de compra deve ser feita por um cliente para um, ou mais, produtos e quantidade que constituem os itens de um pedido no sistema.

A partir desta descrição, podemos enumerar as seguintes regras de negócio para o sistema *MobileShop*:

[Rn01] - Cada cadastro de cliente deve especificar exatamente um nome;

[Rn02] - Cada cadastro de cliente deve especificar exatamente um cpf;

[Rn03] - Cada cadastro de cliente deve especificar exatamente um endereço de entrega;

[Rn04] - Cada cadastro de cliente deve especificar pelo menos um cartão de crédito;

[Rn05] - Cada ordem de compra deve especificar exatamente um cliente;

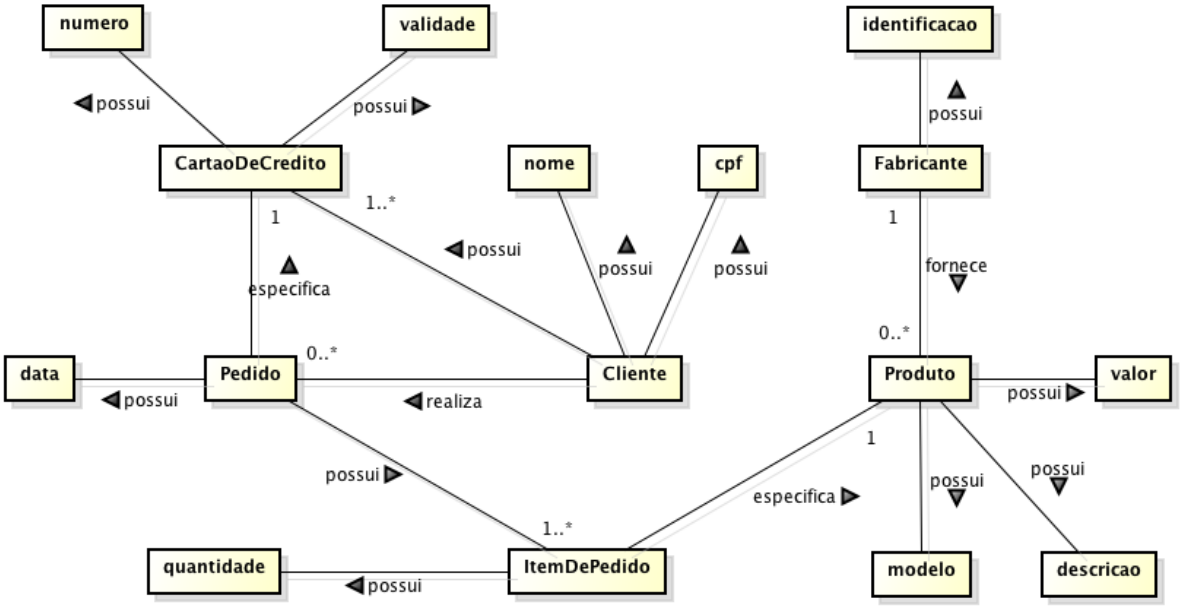
[Rn06] - Cada ordem de compra deve especificar exatamente um cartão de crédito;

[Rn07] - Cada ordem de compra deve especificar pelo menos um produto.

Em algumas propostas de tratamento de regras de negócio em sistemas de informação (JESUS, 2013) é necessário a tradução destes conceitos e regras para a notação do SBVR-SE para que possam ser utilizadas. Na abordagem sugerida neste trabalho estas regras podem ser inseridas diretamente no editor, armazenadas, versionadas e adaptadas sempre que for necessário. Por exemplo, para a regra [Rn01], todos os editores e propostas que utilizam o SBVR-SE como modelo de representação de regras de negócio, teriam que expressar a mesma regra, no idioma português, da seguinte maneira: "***It is obligatory that each*** **cadastro de cliente** *especifica* ***exactly one*** **nome**". Conforme já explicado na Seção 4.5 deste trabalho, esta representação das regras em outros idiomas não é apropriada, e a faceta multilíngue da abordagem proposta neste trabalho não apresenta esta limitação. As regras podem ser utilizadas no *EngenSBVR Editor*, exatamente conforme foram descritas, e serão reconhecidas como regras SBVR válidas.

Para melhor entendimento do problema um modelo conceitual de domínio do sistema é mostrado na Figura 46.

Figura 46 - Modelo conceitual do sistema MobileShop



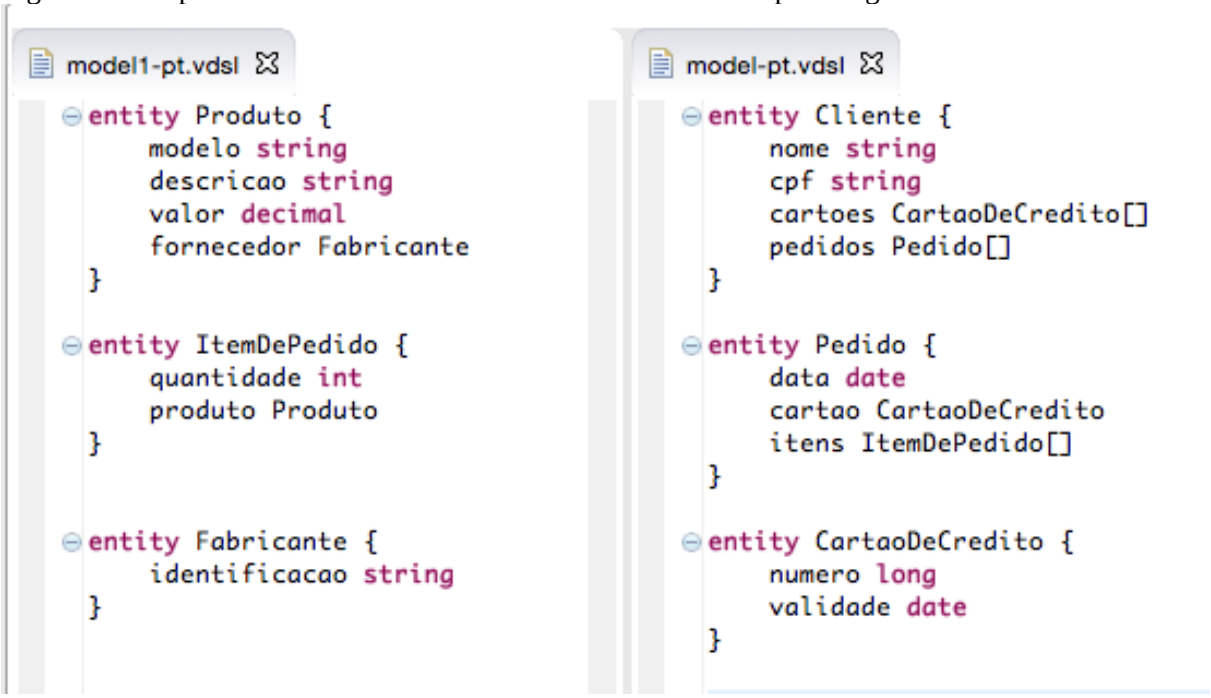
Fonte: Autoria própria.

De acordo com o modelo de domínio mostrado anteriormente, podemos mapear estes vários conceitos para modelos da *EngenDSL* e *EngenSBVR*.

5.3 MAPEAMENTO DOS CONCEITOS DO DOMÍNIO

Na Figura 47 é mostrado o mapeamento dos conceitos de domínio do *MobileShop* na *EngenDSL*. Posteriormente, estes conceitos poderão ser utilizados nas *Views* e *Controllers* do modelo. As extensões dos nomes dos arquivos são importantes, pois diferenciam qual linguagem deve carregar e transformar os tokens definidos em elementos do metamodelo da linguagem. Para a *EngenDSL* os arquivos devem ter a extensão "**.VDSL"** e estarem localizados na pasta de código fonte do projeto (tipicamente a pasta *"src"* em projetos *Java*).

Figura 47 - Mapeamento dos conceitos do domínio do MobileShop na EngenDSL



Fonte: Autoria própria.

A Figura 47 mostra arquivos contendo um mapeamento de conceitos de domínio e seus atributos, que recebe o nome de *"entity"* na *EngenDSL*. É importante notar que, enquanto que alguns atributos representam um relacionamento de um conceito do modelo do domínio de dados simples, por exemplo, o atributo *nome* da entidade *Cliente* é do tipo *string* (sequência de caracteres), outros representam um relacionamento entre entidades do domínio, como no caso do atributo *cartoes*, desta mesma entidade, que representa o relacionamento entre os conceitos *Cliente* e *CartaoDeCredito*, definidos no modelo conceitual de domínio da Figura 46.

Da mesma forma, esses conceitos de domínio podem ser mapeados utilizando a notação SBVR. O conjunto de termos do modelo do domínio do negócio formam, juntamente com suas associações, nos termos da SBVR, o *Vocabulário dos Termos do Negócio*, em oposição ao *Vocabulário de Regras de Negócio*. Consequentemente, esse vocabulário constitui o esquema conceitual do domínio da aplicação. A Figura 48 mostra como esse modelo é definido na *EngenSBVR*.

Figura 48 - Mapeamento dos conceitos de domínio do MobileShop na EngenSBVR

```
TermSet {
    - produto
    - modelo
    - descrição
        Synonym: descricao
    - valor
    - fornecedor
    - item de pedido
    - quantidade
    - fabricante
    - identificacão
        Synonym: identificacao
    - cliente
    - nome
    - cpf
    - cartão de crédito
        Synonym: cartao de credito
    - pedido
    - data
}
```

Fonte: Autoria própria.

A Figura 48 mostra como os conceitos do domínio são definidos utilizando a *EngenSBVR* em seu editor. Primeiramente os termos, ou conceitos nominais, são definidos dentro do escopo (especificado na linguagem por sinais de abre e fecha chaves - "{" e "}") de um marcador que representa um conjunto de termos chamado *TermSet*. Os termos podem possuir atributos associados, definidos em uma linha imediatamente posterior ao termo, como é o caso do atributo *Synonym* do conceito *"endereço"*. Os termos podem ter vários tipos de atributos e suas funções estão definidas no documento normativo da SBVR (OMG, 2013).

5.4 MAPEAMENTO DE CONCEITOS ADICIONAIS

O modelo completo do sistema não envolve somente entidades do domínio do problema. É necessário o mapeamento de toda a interação do usuário com o sistema. Por exemplo, a *"ordem de compra"* é uma ação que o usuário – *Cliente*, do modelo de domínio, realiza, enviando ao sistema informações sobre o *Pedido*. Conforme a descrição do sistema, tais informações são a identificação do próprio cliente e os itens do pedido. Na *EngenDSL* tal

tipo de interação é representada por *Views* e *Controllers*. As *Views* são responsáveis por enviar os dados necessários para os *Controllers* realizarem as ações necessárias. Neste exemplo, apenas as informações necessárias para que os itens de pedido já adicionados sejam enviados para o *Controller* serão mapeadas na *EngenDSL*. A Figura 49 mostra o mapeamento do *Controller* e da *View* que participam desta ação.

Figura 49 - Mapeamento dos conceitos da ordem de compra na EngenDSL

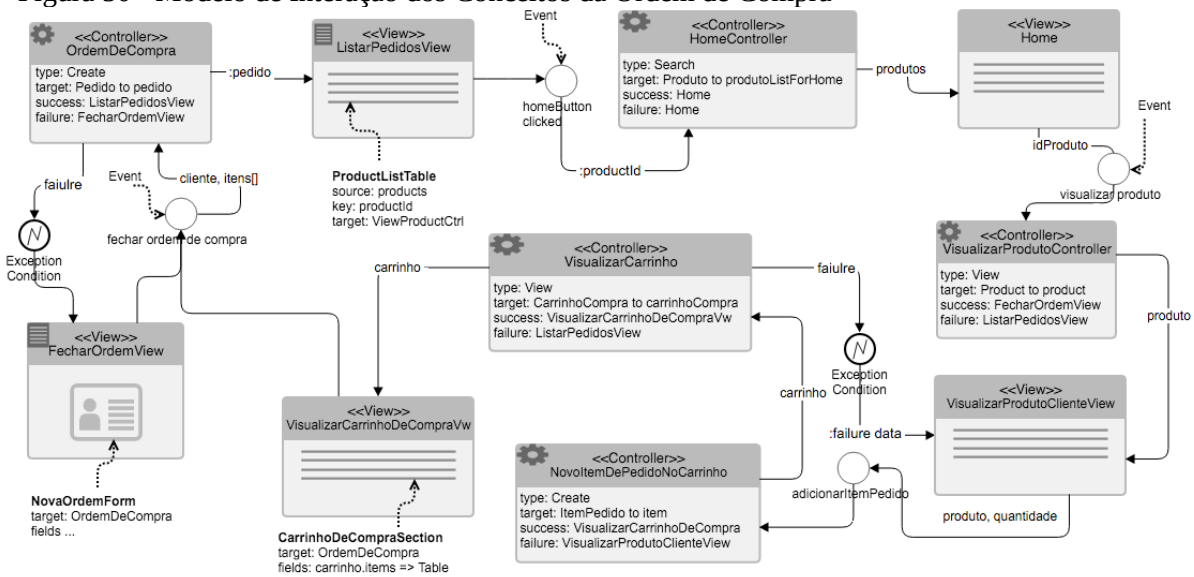
```
controller OrdemDeCompra : Create {
    target Pedido to pedido
    success ListarPedidosView
    failure FecharOrdemView
}

view FecharOrdemView {
    label "Fechar Ordem"
    section NovaOrderForm : form {
        target OrdemDeCompra
        field nomeCliente : label {
            label "Nome do Cliente"
            property nome
            source "cliente"
        }
        field cliente : HIDDEN {
            property cliente
            source "cliente"
        }
        field itens : table {
            label "Itens selecionados"
            target VisualizarItemDePedidoRequest
            property produto.modelo
            property produto.valor
            property quantidade
            property calculaTotalItem
        }
        field totalCompra : label {
            label "Total desta Ordem"
            property calculaTotalOrdem
        }
        section buttons {
            field createOrderBtn : button {
                label "Confirmar Ordem"
            }
            field cancelBtn : button {
                label "Cancelar Ordem"
                target ListagemDeProdutos
            }
        }
    }
}
```

Fonte: Autoria própria.

No exemplo, um *controller – OrdemDeCompra*, e uma *view – FecharOrdemView* estão mapeadas utilizando a sintaxe da *EngenDSL*. O *controller OrdemDeCompra* é do tipo *Create*, o que indica que ela está mapeada para um tipo de componente na arquitetura do sistema que permite a criação de entidades do modelo de domínio, conforme explicado na Seção 4.4.3. A entidade que esta sendo manipulada é definida pela propriedade target do *controller – Pedido*, neste caso. Os *LogicalPaths* (ver Seção 4.4.3) *success* e *failure* respectivamente indicam para qual elemento do modelo de interação com o usuário o sistema deverá redirecionar o processamento no caso de finalização normal, ou com erro. A *view FecharOrdemView* possui um formulário que o usuário utilizará para visualizar os detalhes da ordem (*Pedido*) e fazer alterações se necessário antes da confirmação. O atributo *target* da seção aponta para o controller *OrdemDeCompra*, fazendo que esta seja a ação padrão do formulário. Por esse motivo, o botão de confirmação de compra *createOrderBtn* não possui um *target*. Por outro lado, o botão que permite cancelar a ordem *cancelBtn* redireciona o fluxo da aplicação para permitir a listagem e busca de novos produtos – *ListagemDeProdutos* (código não mostrado), para permitir a adição de novos produtos na lista de produtos de interesse do *Cliente*. O campo "itens" do formulário é uma listagem de itens de pedido. Quando não mencionado em um *field*, o valor da propriedade "*source*" será o nome do campo por padrão. É responsabilidade do *controller* que acionou esta *view* disponibilizar com este nome os *ItemDePedido* que serão mostrados. Assim que o botão *createOrderBtn* for acionado, o objeto de domínio *Pedido* é criado pelo controller *OrdemDeCompra* e os itens de pedido listados na view são associados ao *Pedido*.

O modelo de interação descrito acima é melhor visualizado na Figura 50, que mostra o modelo de navegação e interação da aplicação.

Figura 50 - Modelo de Interação dos Conceitos da Ordem de Compra



Fonte: Autoria própria.

Analogamente, na *EngenSBVR*, os mesmos conceitos devem ser mapeados. Da mesma forma que no exemplo anterior, será mostrado somente o mapeamento dos conceitos necessários para a demonstração da aplicação das regras que envolvem a *ordem de compra*. No modelo definido neste trabalho, o mapeamento desses conceitos é feito utilizando a linguagem natural do usuário do negócio, com algumas peculiaridades. Primeiro é necessário mapear os termos em um *TermSet* e depois mapear as relações com *conceitos verbais* em um *VerbConceptsSet* junto com um *VerbSet* para permitir que verbos sinônimos sejam utilizados nas regras de negócio. Os mapeamentos na *EngenSBVR* são mostrados na Figura 51.

Figura 51 - Mapeamento dos conceitos verbais da ordem de compra na *EngenSBVR*

```
TermSet {
    - ordem de compra
}

VerbConceptsSet {
    - ordem de compra especifica produto
    - ordem de compra especifica cliente
    - ordem de compra especifica cartão de crédito
}

VerbSet {
    - especifica
        Synonym: especificar
}
```

Fonte: Autoria própria.

O mapeamento das próprias regras de negócio, baseadas na notação que se pretende utilizar para expressar conceitos da SBVR no projeto atual, deve ser informado no *EngenSBVR Editor*. Esta notação deve estar baseada no próprio padrão, e isto é verificado pelo editor. Para criar esse mapeamento, é necessário informar ao *Parser de Formulações Lógicas da EngenSBVR*, quais formulações lógicas estão disponíveis para representar as diversas formulações modais do SBVR na linguagem. Na Figura 52 são mostradas as formulações criadas para este projeto.

Figura 52 - Formulações lógicas iniciais para o MobileShop

```
LogicalFormulationSet {
    - É obrigatório que
        Formulation_Kind: obligation_formulation
    - deve
        Formulation_Kind: obligation_formulation
    - cada
        Formulation_Kind: universal_quantification
    - Cada
        Formulation_Kind: universal_quantification
    - exatamente um
        Formulation_Kind: exactly_one_quantification
    - pelo menos um
        Formulation_Kind: at_least_one_quantification
}
```

Fonte: Autoria própria.

Depois de realizado o mapeamento das possíveis formulações lógicas para o projeto atual, as regras de negócio podem ser inseridas no editor, que as reconhecerá como regras de obrigação e restrição de dados, devido à presença de operadores lógicos de obrigação – *"deve"*. A Figura 53 mostra que o *Parser de Formulações Lógicas da EngenSBVR* reconheceu as regras e sabe como classificá-las, pois, a formatação e coloração das fontes foi alterada pelo *EngenSBVR Editor*, conforme descrito na Seção 4.6.2.

Figura 53 - Regras de negócio de ordem de compras para o *MobileShop* no *EngenSBVR Editor*

```
RuleSet {
    - [Rn05] Cada ordem de compra deve especificar exatamente um cliente
    - [Rn06] Cada ordem de compra deve especificar exatamente um cartão de crédito
    - [Rn07] Cada ordem de compra deve especificar pelo menos uma data
    - [Rn08] Cada novo produto deve especificar exatamente um modelo
}
```

Fonte: Autoria própria.

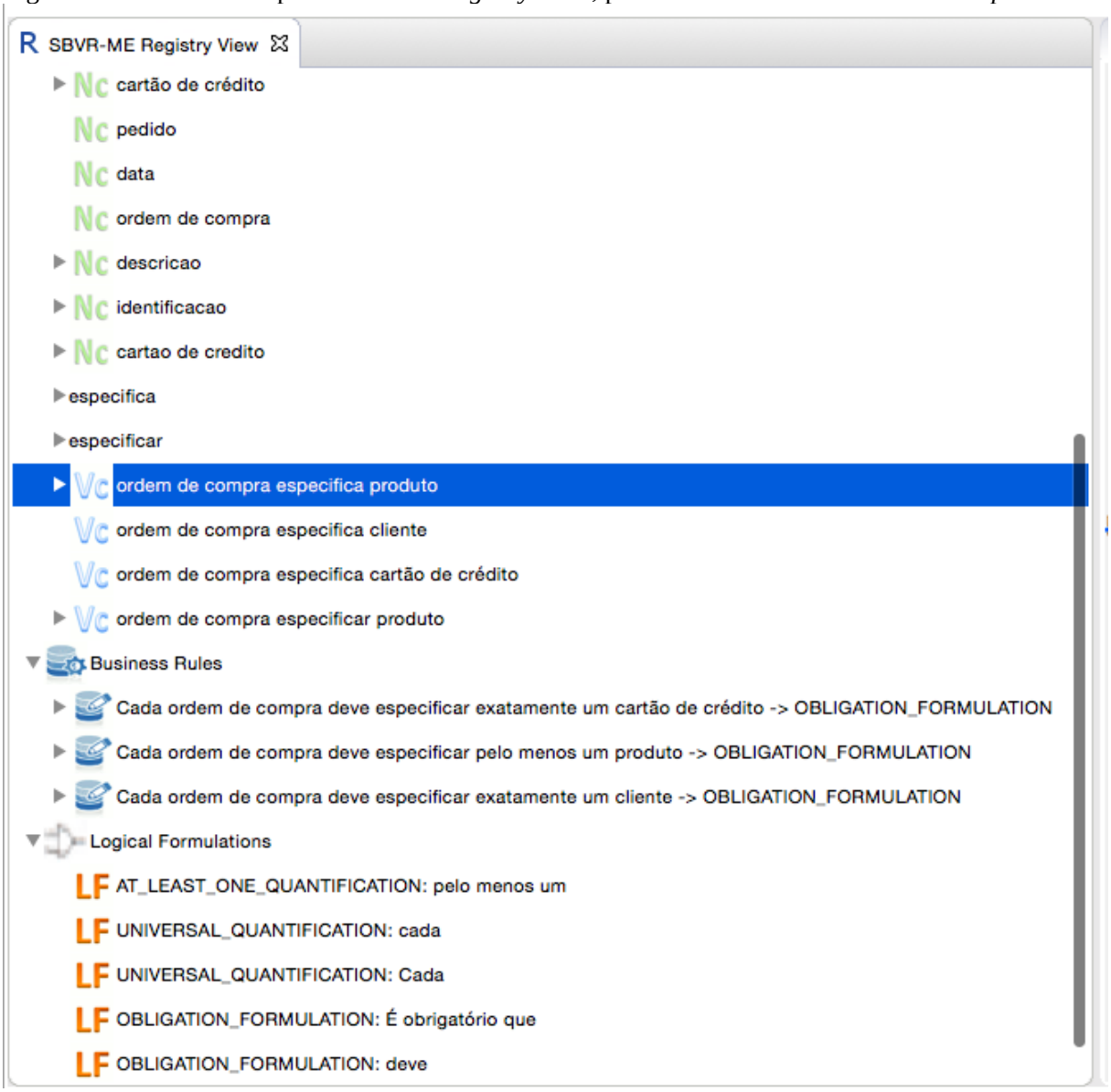
Como pode ser visto, devido à abordagem sugerida neste trabalho permitir a definição de novos elementos representativos das Formulações Lógicas da SBVR implementada pelo Editor *EngenSBVR*, pode-se criar linguagens semelhantes à linguagem natural utilizada em regras de negócio escritas pelos próprios usuários, sem interferência de interpretação por parte de analistas de software, evitando assim a introdução de interpretações errôneas no sistema.

5.5 TRANSFORMAÇÃO DOS MODELOS

Nesta seção será mostrado como os modelos definidos na *EngenDSL* e na *EngenSBVR* para o sistema *MobileShop*, são sucessivamente transformados e adaptados até se chegar a solução final.

Após a especificação dos Vocabulários de Negócio e Vocabulários de Regras de Negócio utilizando o Editor da *EngenSBVR*, o *SBVR Registry View* contendo o resultado do *Parser de Formulações Lógicas da EngenSBVR* pode ser visto conforme mostrado na Figura 54.

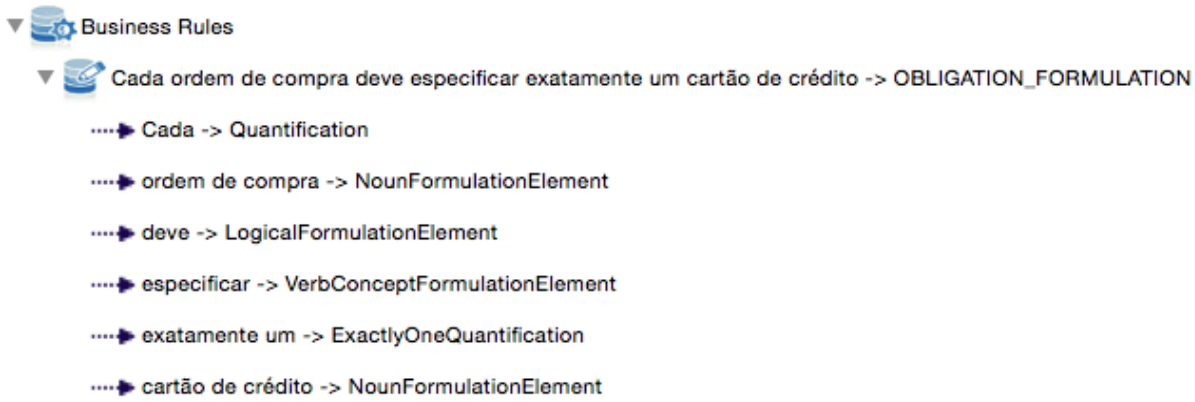
Figura 54 - Resultado do parser - SBVR *Registry View*, para o modelo SBVR do *MobileShop*.



Fonte: Autoria própria.

Mais especificamente, pode-se visualizar nesta mesma *View* o resultado emitido pelo *EngenSBVR Parser* para uma regra de negócio específica como mostrado na Figura 55.

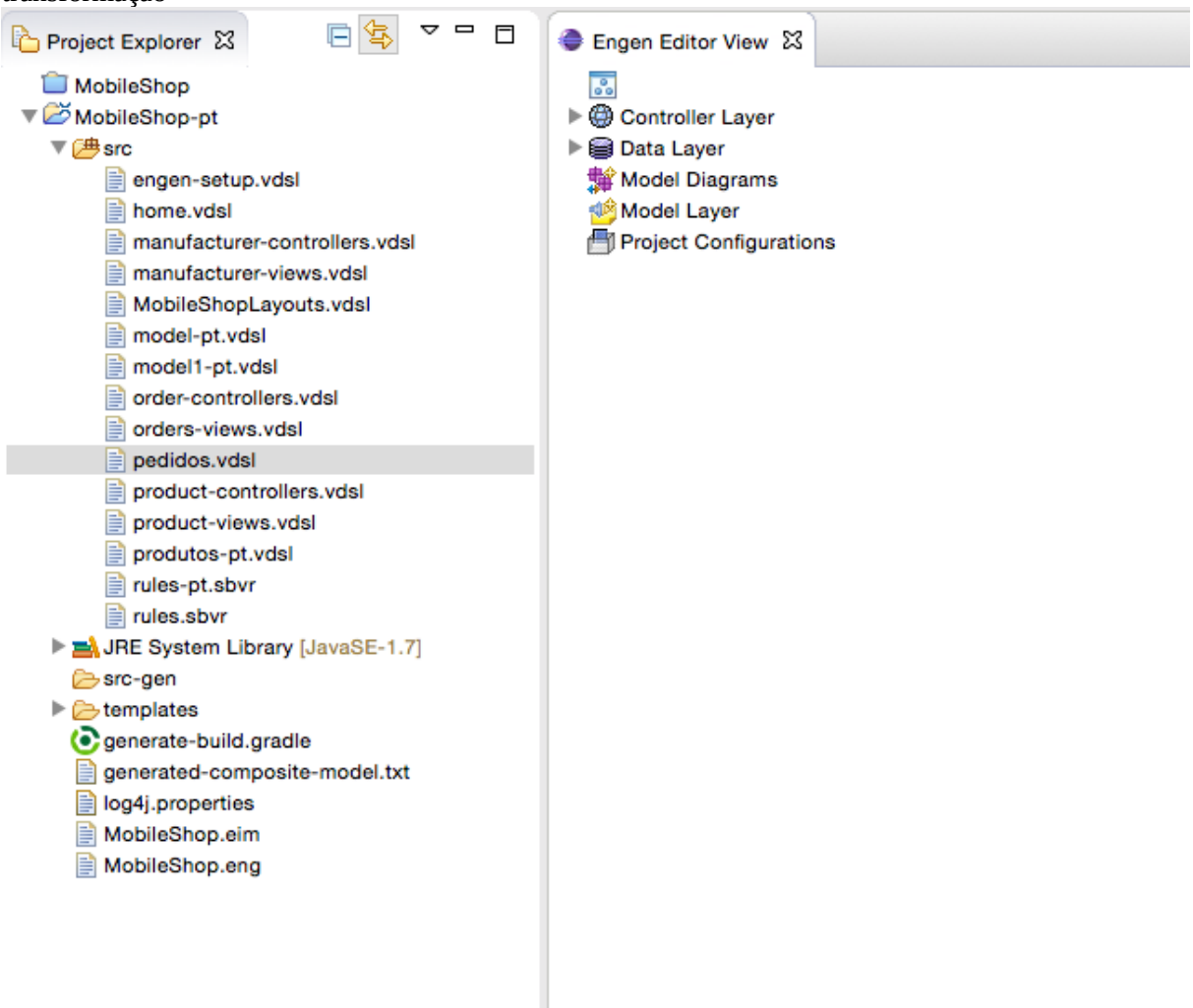
Figura 55 - Resultado do *parse* para uma regra de negócio no *MobileShop*



Fonte: Autoria própria.

Inicialmente, antes da primeira execução de transformação, a visão de projetos do *Eclipse* e do Editor do EIM (*Engen Intermediate Model*) – *Engen Editor View* são mostrados na Figura 56.

Figura 56 - Exemplo do ambiente de trabalho do *Eclipse* para o projeto *MobileShop* antes da primeira transformação



Fonte: Autoria própria.

Como se pode ver, os únicos itens de código presentes no lado esquerdo da Figura 56, dentro da pasta de fontes do projeto (*SRC*), são os próprios arquivos de descrição do modelo utilizando as DSLs (arquivos com extensões ".VDSL" para a *EngenDSL* e .SBVR para a *EngenSBVR*). Além desses, alguns arquivos aparecem no projeto, sendo o *generate-build.gradle* o que será utilizado para realizar a primeira transformação. Este arquivo é um arquivo que contém código *Gradle* (GRADLE, 2015), e são *scripts*, que executam tarefas diversas, utilizando os componentes criados neste trabalho para a implementação do modelo de desenvolvimento dirigido a modelos proposto. Este arquivo é gerado automaticamente para o projeto quando um novo projeto para as linguagens é criado dentro do *Eclipse*. A Figura 57 mostra uma parte importante deste arquivo que define as tarefas de transformação definidas nesta abordagem.

Figura 57 - Seção do arquivo *generate-build.gradle* que contém a definição das tarefas de transformação

```
task cleanGenerated (type: Delete) {
    delete 'src-gen', 'gen-web'
}

task generateAll << {
    ant.taskdef(name: 'generator',
        classname: 'org.engeny.ant.tasks.GeneratorTask',
        classpath: configurations.generatorconfig.asPath + configurations.eclipseConfig.asPath)
    ant.generator(projectMddFile: 'MobileShop.eim', projectfolder: '.')
    println "Finished at " + new Date()
}

task M2M << {
    ant.taskdef(name: 'm2m',
        classname: 'org.engeny.ant.tasks.ModelToModelTransformationTask',
        classpath: configurations.generatorconfig.asPath + configurations.eclipseConfig.asPath)
    ant.m2m(projectMddFile: 'MobileShop.eim', projectfolder: '.')
}

task M2T << {
    ant.taskdef(name: 'm2t',
        classname: 'org.engeny.ant.tasks.ModelToTextTransformationTask',
        classpath: configurations.generatorconfig.asPath + configurations.eclipseConfig.asPath)
    ant.m2t(projectMddFile: 'MobileShop.eim')
}
```

Fonte: Autoria própria.

As tarefas M2M e M2T (configurações de execução de tarefas que podem ser invocadas em um arquivo *Gradle*) utilizam os componentes do *EngenGenerator* para a execução das transformações. Isto permite que a arquitetura da solução proposta possa ser utilizada independente da *Plataforma Eclipse*, via linha de comando, por exemplo.

5.6 MAPEAMENTO DAS VIEWS E CONTROLLERS NA ENGENDSL

Na definição do *controller – ListProdcutCtrl* estão especificadas as suas propriedades e restrições (conforme explicado na Seção 4.4.3). No caso do *ListProdcutCtrl,* o tipo *Search* é definido, enquanto que no *CreateProductCtrl* o tipo *Create* é definido. Os tipos de configuração de *Search* e *Create* de *controllers*, fazem com que este elemento do modelo faça referência a um componente externo definido pelos desenvolvedores. Neste caso, estes componentes desempenham papéis de persistência no sistema, como pesquisa e criação e armazenamento de entidades do modelo respectivamente. A propriedade *target* do *controller* especifica qual entidade ela está manipulando. Os critérios que o *controller* irá utilizar também são especificados, pelo *controller* ou *view* que deu origem ao processamento deste *controller*. Neste exemplo específico, se o *controller* foi acessado da *home-page* do sistema, todos os elementos na base de dados que representam a entidade gerenciada por este *controller* serão retornados, já que nenhum critério está definido, pois esse é o comportamento padrão do componente que está sendo acionado na arquitetura específica deste estudo de caso. Outros comportamentos podem ser utilizados, dependendo do componente acionado na plataforma arquitetural de cada projeto.

No elemento *ListProductVw,* ilustrado na Figura 58, é mostrado como os elementos *Section* podem ser aninhados e como a pesquisa por produtos pode ser personalizada para cada *View*.

Figura 58 - Exemplo de uma *View* na *EngenDSL* com critérios de pesquisa

```
view ListProductVw {
    label "Products found"
    section productListSection {
        field productTable : table {
            target ProductDetailCtl
            // The request attribute the controller
            // will produce with all elements to iterate
            source "returnCollection"
            // The property used to identify the
            // current item in the table
            key "id"
            // Properties to show in this table
            property photo
            property name
        }
    }
    section RefineSearchSection : form {
        target ListProductCtl
        label "Search Products"
        section SearchFields {
            field name : text {
                property name "Name:"
            }
            field manufacturer : select {
                id manufacturer
                label "Manufacturer:"
                property manufacturer "Manufacturer"
            }
            field NewSearch : button {
                id newSearchButton
                label "Search Products"
                target ListProductCtl
            }
        } // End of SearchFields
    } // End of RefineSearchSection
}
```

Fonte: Autoria própria.

Quando o *ListProductsCtrl controller* for acessado sem nenhum critério contextual de pesquisa ele realizará uma pesquisa por todos os produtos. Entretanto, o alvo definido no atributo *target* do *NewSearchButton*, no formulário *RefineSearchSection*, irá utilizar as informações contextuais dos campos da página na qual está inserido (*name* e *manufacturer*) como critérios de pesquisa para que o *controller* possa refinar os resultados da busca.

Estas funcionalidades são executadas pelo *Web Framework* que está sendo utilizado neste trabalho. É possível identificar na especificação da configuração do botão *NewSearch* que a propriedade *target* aponta para o *controller ListProductCtl*. Esta informação corresponde ao modelo de eventos projetado para este componente (*ViewField*). No modelo proposto, esta informação é suficiente para que os *templates* interpretem o modelo de eventos: o *controller* alvo da ação, assim como os campos contendo os critérios para a pesquisa informadas pelo usuário.

Similarmente, outras *views* podem ser definidas utilizando as várias construções permitidas pela *EngenDSL*. A Figura 59 mostra a definição da *view AlterarProdutoVw*.

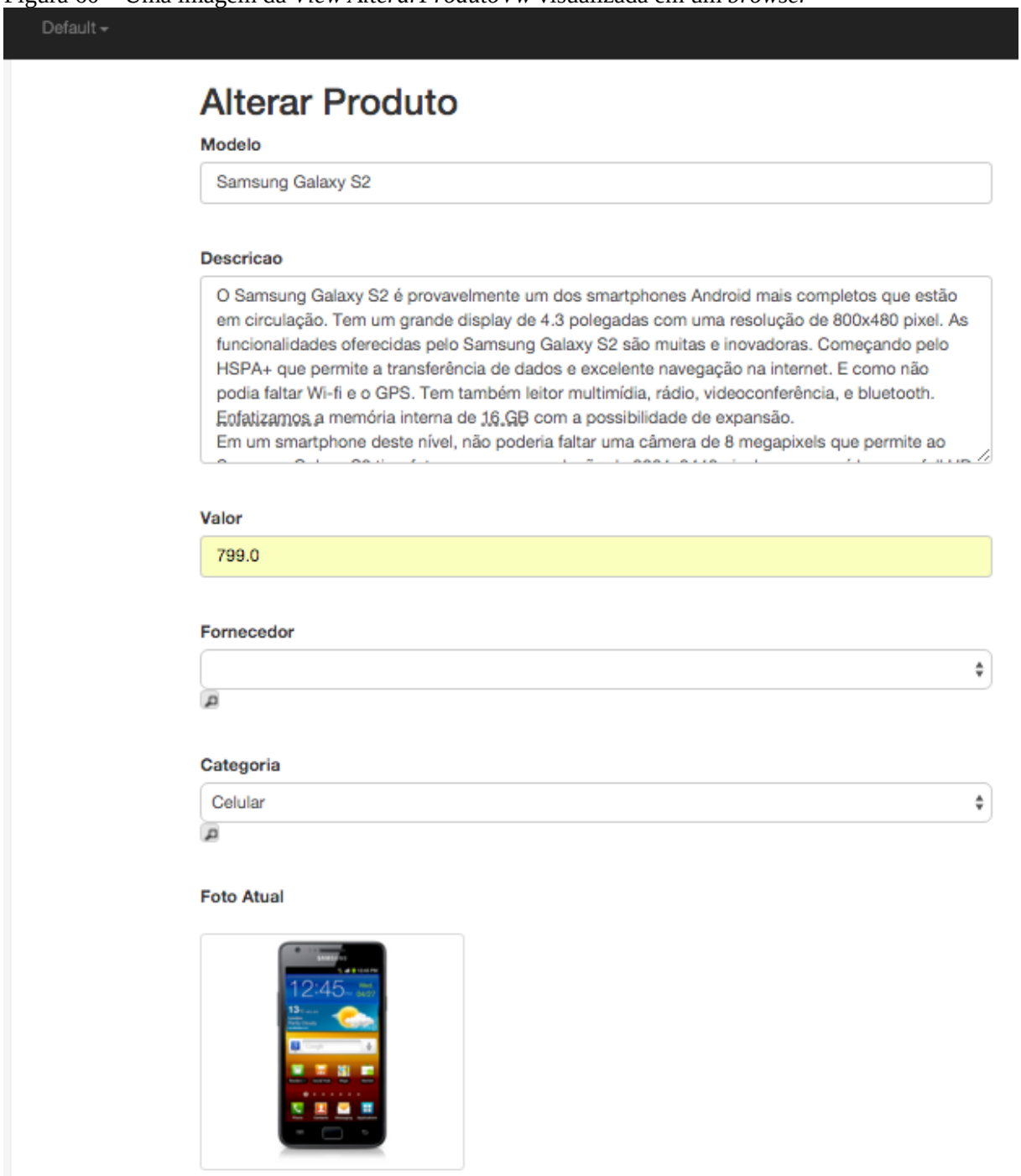
Figura 59 - Definição da *View AlterarProdutoVw* no sistema de exemplo *MobileShop*

```
view AlterarProdutoVw {
    label "Alterar Produto"
    section AlterarProdutoForm : form {
        target AtualizarFotoDoProduto
        putFields
        // Cria um novo campo do tipo file para
        // receber a imagem do prodoto
        field ArquivoDeFoto : file {
            // Um campo no formulário criado para
            // conter os dados da figura
            property: foto_file
            source: "map"
        }
        section FotoAtualSection {
            style "form-group"
            field FotoAtualLabel : label {
                style "control-label"
                value "Foto Atual"
            }
            field FotoAtual : image {
                style "img-thumbnail img-responsive col-xs-4"
                property: foto_encoded
            }
        }
        section FotoAtualDivSection {
            style "row"
        }
        section AlterarProdutoFormButtons {
            style "row-fluid button-group"
            field AlterarProdutoBtn : button {
                label "Alterar Produto"
            }
            field ExcluirProdutoBtn : button {
                label "Excluir Produto"
                target ExcluirProdutoController
            }
            field CancelarBtn : button {
                label "Cancelar"
                target ListagemDeProdutos
            }
        }
    }
}
```

Fonte: Autoria própria.

A Figura 60 mostra uma imagem da página HTML da aplicação, gerada neste exemplo, em execução quando acessada por um *browser* em um servidor de aplicações, de acordo com a definição da tela *AlterarProdutoVw*.

Figura 60 – Uma imagem da *View AlterarProdutoVw* visualizada em um *browser*



Fonte: Autoria própria.

5.7 EXEMPLO DE APLICAÇÃO DE REGRAS DE NEGÓCIO

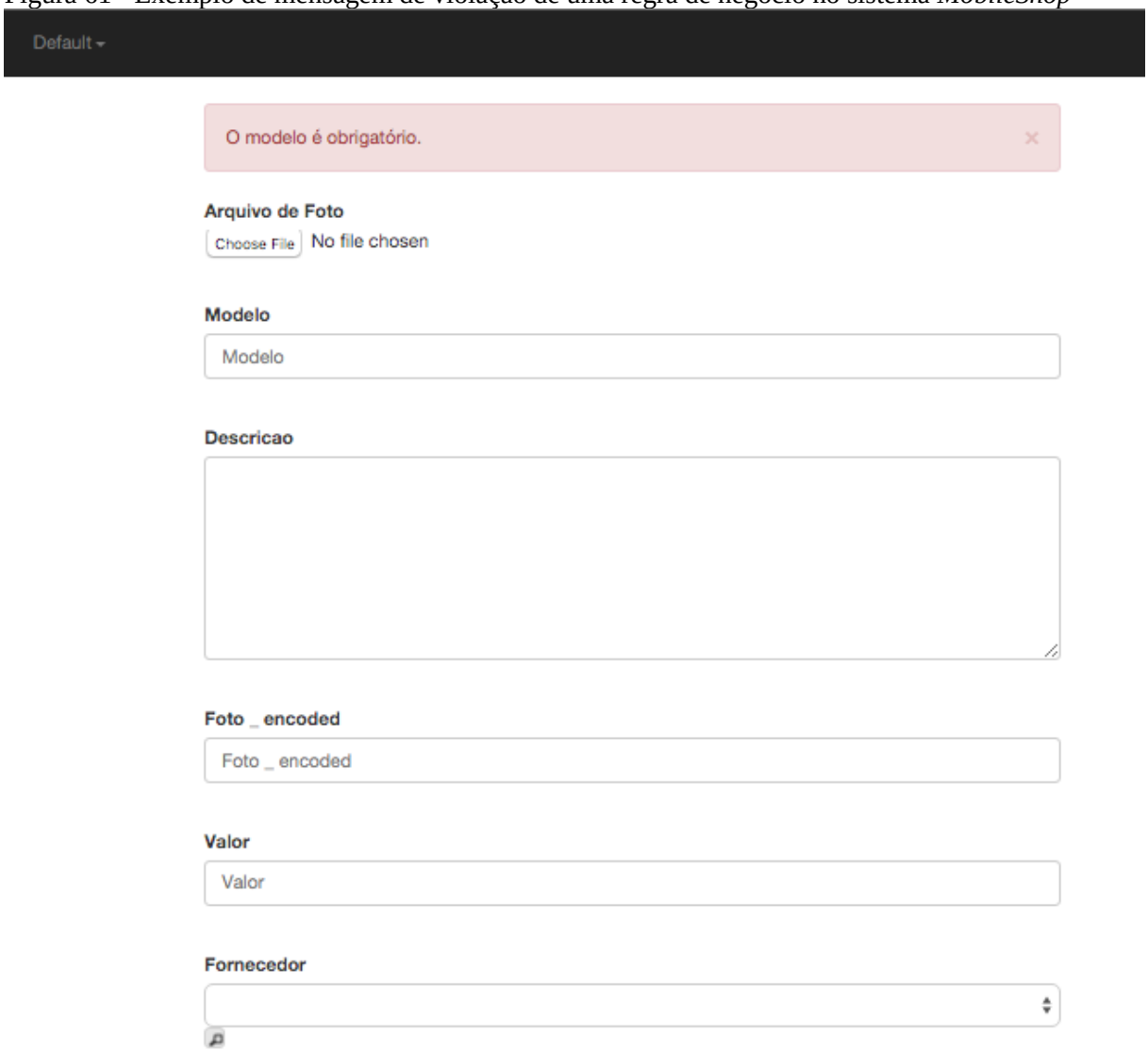
Como exemplo de aplicação de regras de negócio em sistemas de informação, é mostrada uma forma de aplicação de *Regras de Negócio Restritivas de Entrada de Dados* e de como tal requisito é tratado pela *Plataforma Engen*. Um dos tipos de restrições que podem ser implementadas por essas regras são as de especificação de campos obrigatórios. Esse tipo de

regra de negócio, explicita quais campos de um domínio são obrigatórios na interface com o usuário, e geralmente é implementada em sistemas de informação por meio de validadores de dados de entrada.

A *EngenDSL* permite que sejam definidas *Views*, que podem possuir seções com formulários de entrada de dados. Desta forma, é possível definir um formulário de cadastro de uma "ordem" que irá enviar a um *controller* as informações que serão utilizadas para a criação da mesma. Entretanto, a definição de quais informações são obrigatórias pertencem ao conjunto de regras definidas para cada negócio. Apesar de ser possível inserir estas restrições diretamente na *EngenDSL* como mostrado na Seção 4.4.3, tais regras deveriam estar presentes no conjunto de regras do negócio e manipuladas diretamente pelos analistas de negócio e usuários do negócio.

A integração de regras de negócio ocorre no nível do modelo EIM gerado para cada aplicação. No exemplo do *MobileShop*, a regra de negócio [Rn08] definida na Figura 53, é aplicada sempre que há uma entrada de um novo produto no sistema. Como foi definido que o modelo do produto é obrigatório para cada novo produto, o modelo de integração proposto irá inserir esta restrição ao controller *NovoProduto*, pois os nomes "novo produto" no modelo da *EngenSBVR*, e "*NovoProduto*" no modelo da *EngenDSL* são compatíveis, e serão interpretados como o mesmo conceito. Assim, após uma tentativa de criação de um produto, sem especificar o modelo, utilizando o controller *NovoProduto*, o sistema emitirá uma mensagem informando o problema, e não irá permitir o cadastramento de novos produtos sempre que violem esta regra. Esse comportamento pode ser visto na Figura 61.

Figura 61 - Exemplo de mensagem de violação de uma regra de negócio no sistema *MobileShop*



Fonte: Autoria própria.

5.8 INTEGRAÇÃO DE COMPONENTES EXTERNOS

O exemplo de *controller* da Figura 62, mostra como componentes externos podem ser adicionados ao modelo da *EngenDSL* para compor o modelo de interação da aplicação.

Figura 62 - Exemplo de associação de componentes externos à conceitos da EngenDSL.

```
// Um controller específico para codificar array de bytes em String-base64
typedef: Base64EncoderAction as org.sif.struts.Base64EncoderAction
// Um controller personalizado para fazer o upload de arquivos
typedef: UploadFileAction as org.sif.struts.UploadFileAction

// Para criar um produto, primeiro é necessário fazer o
// upload do arquivo de foto, codificar estes dados em base64
// e guardar o resultado no atributo correto do bean
controller UploadNovaFotoDoProduto : UploadFileAction {
    target Produto to produto
    showOnMenu false
    success CodificarFotoDoProduto1
    failure NovoProdutoVw
}

controller CodificarFotoDoProduto1 : Base64EncoderAction {
    target Produto to produto
    showOnMenu false
    // O controller UploadFileAction irá encontrar o campo ArquivoDeFoto
    // da view NovoProdutoVw e disponibilizar o conteúdo em um atributo
    // da requisição com o mesmo nome e o sufixo "_bytes"
    pre-process {
        request.attributes_to_encode = "ArquivoDeFoto_bytes"
    }
    success NovoProduto
    failure ListagemDeProdutos
}
```

Fonte: Autoria própria.

Neste exemplo, novos tipos de *controllers* foram definidos: *Base64EncoderAction* e *UploadFileAction*. Estes *controllers* estão associados a componentes externos: *org.sif.struts.Base64EncoderAction* e *org.sif.struts.UploadFileAction*, respectivamente.

Após a transformação M2T, a Figura 63 mostra um exemplo de como o *controller UploadNovaFotoDoProduto* foi transformado em código Java, integrando o componente externo ao *controller* gerado: o *controller UploadNovaFotoDoProduto* "*extends*", ou seja, deriva do componente externo *org.sif.struts.UploadFileAction.*

Figura 63 - Exemplo do código Java gerado como resultado da associação de um componente externo

```
41 public class UploadNovaFotoDoProduto extends org.sif.struts.UploadFileAction<Produto, Long> {
42
43     private final static Logger log = LoggerFactory.getLogger(UploadNovaFotoDoProduto.class);
44
45
46     @Override
47     public void preProcessRequest(BaseServiceController<Produto, Long> service,
48             ActionMapping mapping, ActionForm actionForm, HttpServletRequest request,
49             HttpServletResponse response) throws Exception {
50         DynaActionForm form = (DynaActionForm) actionForm;
51         HttpSession session = (HttpSession) request.getSession();
52         super.preProcessRequest(service, mapping, actionForm, request, response);
53     }
54
```

Fonte: Autoria própria.

5.9 CONSIDERAÇÕES FINAIS

Neste capítulo foi mostrado um exemplo de uso da *Plataforma Engen*, utilizando a abordagem sugerida neste trabalho. Foram definidos um modelo conceitual e requisitos da aplicação, juntamente com a modelagem conceitual e implementação do modelo utilizando a *EngenDSL* e *EngenSBVR*. Além disto, foi mostrado como é possível modelar a interação com o usuário através da implementação dos aspectos referentes à esta interação e a validação da entrada de dados na fronteira aplicação utilizando a abordagem do modelo SBVR sugerida.

Além da modelagem da aplicação, foi mostrado como os processos de transformação contribuem para a arquitetura da abordagem proposta, permitindo que os elementos do modelo sejam transformados, por meio de *templates* direcionados para a arquitetura alvo da aplicação de exemplo, em artefatos da aplicação final. Tais artefatos incluem código-fonte, arquivos de configuração e telas do sistema baseados na plataforma *Web*.

A abordagem MDSD utilizada na proposta desta dissertação, permite a reutilização de soluções arquiteturais, baseadas em um modelo de interação com o usuário e integração de regras de negócio de entrada de dados. Nesta perspectiva, o modelo utilizado permite que sistemas de informações empresariais sejam criados de forma célere, com uma qualidade somente limitada à qualidade dos *Templates Arquiteturais* utilizados, e pode ser aplicado independentemente da IDE utilizada pelos analistas que estão trabalhando com o sistema. O modelo contribui com a legibilidade de regras de negócio, com o suporte a múltiplos idiomas e sua rastreabilidade em relação aos elementos da interface com o usuário. Sob o ponto de vista de manutenção do modelo conceitual, a abordagem textual das DSLs permite uma fácil integração com ferramentas de versionamento de código, diminuindo o custo associado à evolução do sistema.

6 CONCLUSÃO

A utilização de processos de desenvolvimento dirigido a modelos é uma tendência crescente que se verifica em projetos de software. Esses processos tendem a melhorar o resultado do esforço de desenvolvimento, promovendo um aumento na produtividade, reúso e qualidade de soluções informatizadas. Um dos fatores preponderantes no sucesso de um projeto de sistemas de informação é sua capacidade de adaptação às mudanças no contexto empresarial ao qual estão inseridas, permitindo que regras de negócio deste contexto possam ser rapidamente criadas, manipuladas e integradas.

Este trabalho apresentou uma proposta de especificação de regras de negócio baseadas no padrão SBVR – *Semantic of Business Vocabulary and Rules*, juntamente com a especificação de interações com o usuário baseadas no padrão IFML – *Interaction Flow Modeling Language*, utilizando DSLs – *Domain Specific Languages*, que aliadas a técnicas MDSD – *Model Driven Software Development*, permitem modelar e implementar sistemas de informações empresariais. O objetivo da abordagem adotada, é permitir que regras de negócio sejam definidas em uma linguagem de fácil verificação e manutenção pelo usuário de negócio, e que esta mesma definição faça parte do modelo conceitual do sistema, validadas no modelo de interação com o usuário, possibilitando que sejam diretamente transformadas em artefatos de software e restrinjam o comportamento do sistema. Neste sentido, foi apresentado como as regras são integradas ao modelo de interação com o usuário, permitindo que imprimam restrições ao mesmo.

Conforme exposto na introdução deste trabalho, a incorporação de regras de negócio explicitamente no desenvolvimento de sistemas, para garantir uma maior agilidade nas mudanças e sua incorporação nas aplicações, é uma necessidade apontada em diversos trabalhos como uma lacuna na implementação de regras de negócio no processo de desenvolvimento de sistemas. Entretanto, o padrão SBVR se restringe a formular regras de negócio, não apresentando como, ou quando, tais regras devem ser validadas, ou verificadas, nos sistemas de informação. Com o objetivo de propor uma forma de integrar regras de negócio ao processo de desenvolvimento e tratá-las explicitamente nos sistemas de informação, foi mostrada uma solução que permite a integração de um modelo de interação com o usuário com regras de negócio restritivas de entrada de dados.

Neste contexto, os padrões SBVR e IFML, ambos definidos pela OMG, foram utilizados para a descrição de regras de negócio e interação com o usuário respectivamente,

sendo que tal descrição é fornecida por meio da utilização de duas linguagens específicas de domínio criadas para este fim – a *EngenSBVR* e *EngenDSL* respectivamente.

Especificamente no contexto de desenvolvimento de sistemas empresariais com interação com o usuário, a *EngenDSL* contribui como uma implementação dos conceitos de interação com o usuário lastreado na IFML, além de adicionar características importantes como a possiblidade de definição e reúso de *layouts* e estilos para *views*, especificação de bibliotecas externas, especificação de comportamento e encadeamento de ações do usuário. Além das DSLs, uma série de componentes que implementam os conceitos do MDSD para a criação de sistemas foram implementados e apresentados nesta dissertação.

Foi mostrado como as linguagens se integram a fim de permitir que regras de negócio de restrição de entrada de dados possam impor restrições na interação com o usuário do sistema, ou seja, identificar entradas de dados inválidas. Apesar de existirem diferentes tipos de regras de negócio, limitamos o escopo deste trabalho em mostrar a viabilidade da integração sugerida, que abrange um espectro representativo de regras relativas à interação com o usuário, sem, no entanto, considerar todas as diversas possibilidades de representação destas regras.

Para uma melhor experiência com o uso das linguagens criadas, e da utilização da abordagem MDSD proposta, foram criados editores que facilitam a manipulação das linguagens específicas de domínio, e componentes que implementam o processo de transformações dos modelos descritos através das DSLs, em outros modelos e código na arquitetura alvo da aplicação especificada pelos *templates* arquiteturais. Especificamente, mostrou-se como é possível executar duas fases distintas de transformação: de modelo-para-modelo (M2M) que gera um modelo intermediário chamado EIM; e uma fase de transformação modelo-para-texto (M2T) que gera o código final da aplicação. Também foi mostrado como os *templates* arquiteturais, que definem a configuração final do sistema na arquitetura alvo, podem ser personalizados, permitindo uma boa flexibilidade em sua utilização. Também, foram mostradas técnicas de integração de código gerado e código feito manualmente que permitem a adaptação de sistemas legados e componentes externos, ao processo.

Além disto, o metamodelo da SBVR foi utilizado e estendido, para permitir a definição de novas linguagens naturais controladas. Foi mostrado como é possível utilizar uma das ferramentas propostas – o editor da *EngenSBVR*, para a definição de linguagens naturais textuais, baseadas no padrão SBVR. Um exemplo deste tipo de linguagem é a SBVR-SE, que é uma linguagem natural controlada em inglês, para expressar vocabulário e regras de

negócio baseados no SBVR. Entretanto, nas abordagens pesquisadas não foram encontrados exemplos de ferramental, ou modelos, para a produção de novas linguagens, ou até mesmo linguagens multilíngues, assim a necessidade das extensões criadas e ferramental produzido nesta proposta.

Conforme analisado no Capítulo **3**, analogamente à proposta deste trabalho, vários estudos foram feitos para a transformação de linguagem natural para SBVR em modelos executáveis, ou semi-executáveis. A maioria se concentra em gerar algum tipo de representação intermediária como UML ou OCL. Outros, propõem uma abordagem MDD baseado em regras de negócio por meio de modelagem UML ou por meio de uma DSL específica. Entretanto nenhuma das abordagens analisadas propõe um modelo de interpretação do modelo SBVR, descrito por meio de uma linguagem controlada multilíngue e que permita a transformação em código final da aplicação. Além disto, outros aspectos de um sistema de informação tradicional como navegação, *layout* e visualização também não são abordados em conjunto com regras SBVR nos estudos anteriores. Neste trabalho estes aspectos são mapeados, juntamente com regras SBVR para restrição de entrada de dados e aplicados a técnicas MDD para a criação da aplicação final.

6.1 BENEFÍCIOS ALCANÇADOS

Conforme exposto no Capítulo 1, os objetivos esperados deste trabalho foram o de apresentar uma solução para a especificação de regras de negócio utilizando o padrão *Semantics of Business Vocabulary and Rules – SBVR*, aliada à especificação de interações com o usuário baseados na *Interaction Flow Modeling Language – IFML*, que permita modelar sistemas de informação utilizando-se técnicas de desenvolvimento de software dirigido a modelos. Conforme mostrado nas definições do Capítulo 4 e implementado no estudo de caso do Capítulo 5, a solução da Plataforma *Engen*, permite a criação de aplicações para a plataforma *Web* seguindo estes conceitos. Os principais elementos da solução foram enumerados na Plataforma *Engen* e estão resumidos a seguir:

- ***Processo Dirigido a Modelos*** – processo adaptado de desenvolvimento para apoiar o desenvolvimento baseado na solução proposta.
- ***EngenDSL*** – uma linguagem específica de domínio que implementa os conceitos do modelo de navegação e interação com o usuário definidos no padrão IFML;

- ***EngenSBVR*** – uma linguagem específica de domínio que permite modelar o vocabulário e regras de negócio utilizando o padrão SBVR como modelo de implementação e com suporte a múltiplos idiomas;
- ***EngenDSL Editor e EngenSBVR Editor*** – editores das linguagens criadas que permitem a integração com a Plataforma *Eclipse* possibilitando edição dos modelos em forma textual e visualização dos mesmos de forma gráfica, além de validação semântica e sintática em tempo de desenvolvimento dos modelos;
- ***Modelo de interação com o usuário*** – modelo baseado no IFML, mas com extensões definidas para auxiliar na criação de aplicações *Web*;
- ***Modelo SBVR e extensão para idiomas*** – modelo criado para permitir a extensão do modelo de sintaxe do SBVR para qualquer idioma;
- ***Ferramentas de Apoio à Transformação*** – conjunto de ferramentas, agrupadas pelo *EngenGenerator*, de apoio à transformação entre modelos que permite atuar no processo de desenvolvimento proposto;

A potencial facilidade da descrição do modelo de regras de negócio SBVR e de interação com o usuário, aliado ao ferramental criado para validação, versionamento de código e manutenção dos modelos, propicia um rápido tratamento dos problemas do esforço envolvido no aumento da qualidade e facilidade de manutenção de Sistemas de Informações Empresariais para a *Web*.

6.2 LIMITAÇÕES E RESTRIÇÕES

Neste trabalho, não foi abordada a problemática de transformação de regras de negócio expressas em linguagem natural utilizando SBVR. Consideramos que as regras devem ser escritas, ou transformadas para o modelo SBVR, para efeito de especificação de regras de negócio em um sistema de informação.

Além disto, as ferramentas apresentadas não implementam toda a extensão do padrão SBVR, ou está pronta para criar um sistema completo e colocá-lo em produção. Elas ainda necessitam de implementação de aspectos importantes para a descrição de uma aplicação completa, e.g. como aspectos de um *workflow*. Desta forma, a proposta deste trabalho, limitou-se a possibilitar a utilização de uma gama de funcionalidades que permitisse a descrição regras de negócio estruturais e comportamentais para validações de entrada de

dados e subsequente geração de uma aplicação que possibilitasse o entendimento de como as regras definidas, possibilitam alterar o comportamento do sistema, conforme descrito nos exemplos e no estudo de caso, visando validar a proposta. Além disto, a ferramenta necessita melhorar o suporte à manipulação de modelos visualmente, por se tratar de uma funcionalidade usual neste tipo de solução.

6.3 TRABALHOS FUTUROS

No contexto deste trabalho, diversas extensões e melhorias são desejáveis e podem ser alvo de trabalhos futuros. A incorporação de um maior número de regras de negócio para a integração com o modelo de interação com o usuário; a inclusão de regras de produção; integração com outros modelos de definição de vários outros aspectos de sistemas de informação como padrões de intercomunicação e *workflows*; geração de outros tipos de artefatos como modelos intermediários, a exemplo da UML; o *design* gráfico, ou visualização, dos modelos produzidos pelas DSLs, além da execução direta dos mesmos, sem um estágio de geração de código, são todos trabalhos que podem ser desenvolvidos a partir desta proposta.

6.4 PUBLICAÇÕES

Durante a confecção do trabalho apresentado, foram realizadas as seguintes publicações para validar a proposta:

- PURIFICAÇÃO, Carlos Eugênio Palma da; SILVA, Paulo Caetano da. **ENGENDSL: uma linguagem específica de domínio para aplicações Web**. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGY MANAGEMENT – CONTECSI, 10., 2013, São Paulo. **Anais...** São Paulo, *June*, 12 *to* 14, 2013.
- PURIFICAÇÃO, Carlos Eugênio Palma da; SILVA, Paulo Caetano da. A domain-specific language for modeling web user interactions with a Model Driven Approach. In: ICIW 2015: THE TENTH INTERNATIONAL CONFERENCE ON INTERNET AND WEB APPLICATIONS AND SERVICES, 10., 2015, Brussels, Belgium. **Anais...** June 21 - 26, 2015.

REFERÊNCIAS

ABRAN, Alain et al. **Guide to the Software Engineering Body of Knowledge - SWEBOK.** Nj, Usa: Ieee Press Piscataway, 2001.

AFREEN, Hina; BAJWA, Imran Sarwar. Generating UML Class Models from SBVR Software Requirements Specifications. In: BENELUX CONFERENCE ON ARTIFICIAL INTELLIGENCE: BNAIC 2011, 23., 2011. Gent, Belgium. **Proceeding…** 2011. p.23-32. Disponível em: <https://www.researchgate.net/publication/215778543_Generating_UML_Class_Models_from_SBVR_Software_Requirements_Specifications>. Acesso em: 10 ago. 2016.

AGRAWAL, Ashish; SINGH, Sharad. **Automatic code structure and workflow generation from models:** graphical editor for Knowledge Representation based on SBVR. London: The London School Of Economics And Political Science, 2009. 36 p. (OPAALS D10.14 Report). Open Philosophies for Associative Autopoietic Digital Ecosystems. Disponível em: <http://www.lse.ac.uk/media@lse/research/OPAALS/D10.14.pdf>. Acesso em: 10 ago. 2016.

AQUINO, Nathalie et al. Conceptual modelling of interaction. **Handbook Of Conceptual Modeling:** Theory, Practice, and Research Challenges**,** p.335-358, 2011. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15865-0>. Acesso em: 10 ago. 2016.

BACHERLER, Christian et al. Automated Test Code Generation Based on Formalized Natural Language Business Rules. In: ICSEA 2012: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ADVANCES, 7., 2012, Lisbon, Portugal. **Proceedings ….** 2012. p.165-171.

BAJEC, Marko; KRISPER, Marjan. Issues and Challenges in Business Rule Based Information Systems Development. In: ECIS 2005 IN INFORMATION SYSTEMS IN A RAPIDLY CHANGING ECONOMY, 2005. **Proceedings**… 2005. p. 1-12.

BAJEC, Marko; KRISPER, Marjan. A methodology and tool support for managing business rules in organisations. **Journal Information Systems,** Oxford, Uk, v. 30, n. 6, p.423-443, set. 2005. Elsevier BV. http://dx.doi.org/10.1016/j.is.2004.05.003

BAJEC, Marko; RUPNIK, Rok; KRISPER, Marjan. Using Business Rules Technologies To Bridge The Gap Between Business and Business Applications. In: INFORMATION TECHNOLOGY FOR BUSINESS MANAGEMENT, 16., 2000, Peking, China. **Proceedings...** Peking, China: Rechnu G., 2000. p. 77 - 85. Disponível em: <http://bajecm.fri.uni-lj.si/downloads/Bajec IFIP 2000.doc>. Acesso em: 10 ago. 2016.

BAJWA, Imran Sarwar. **A natural language processing approach to generate SBVR and OCL.** 2012. 209 f. Tese (Doutorado) - Curso de Computer Science, School Of Computer Science, University Of Birmingham, Edgbaston, Uk, 2014. Disponível em: <http://etheses.bham.ac.uk/4890/>. Acesso em: 11 ago. 2016.

BAJWA, Imran Sarwar; BORDBAR, Behzad; LEE, Mark. SBVR vs OCL: A comparative analysis of standards. In: IEEE INTERNATIONAL MULTITOPIC CONFERENCE, 14., 2011. **Proceedings**… 2011. p.261-266. http://dx.doi.org/10.1109/inmic.2011.6151485.

BAJWA, Imran Sarwar; BORDBAR, Behzad; LEE, Mark G.. OCL Constraints Generation from Natural Language Specification. In: IEEE INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE, 14., 2010. **Proceedings…** 2010. http://dx.doi.org/10.1109/edoc.2010.33. Disponível em: <http://www.cs.bham.ac.uk/~bxb/Papres/1005.pdf>. Acesso em: 9 ago. 2016.

BAJWA, Imran Sarwar; LEE, Mark G.; BORDBAR, Behzad. SBVR Business Rules Generation from Natural Language Specification. In: AAAI 2011 SPRING SYMPOSIUM SERIES: AI FOR BUSINESS AGILITY (AI4BA), 2011, Palo Alto, California. **Proceedings…** 2011. Disponível em: <http://aaai.org/ocs/index.php/SSS/SSS11/paper/view/2378/2918>. Acesso em: 8 ago. 2016.

BETTIN, Jorn. Model-Driven Software Development. **Mda Journal.** [S.l..], p. 2-11. abr. 2004. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.1393&rep=rep1&type=pdf>. Acesso em: 9 ago. 2016.

BROCKE, Jan Vom et al. Current and Future Issues in BPM Research: A European Perspective from the ERCIS Meeting 2010. **Communications Of The Association For Information Systems.** Brisbane, Australia, p. 394-412. maio 2011.

BROSCH, Petra et al. Conflicts as First-Class Entities: A UML Profile for Model Versioning. **Models In Software Engineering,** [S.l.], v. 6627, p.184-193, 2011. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-21210-9_18.

CADAVID, Juan et al. A Domain Specific Language to Generate Web Applications. In: MEMORIAS DE LA CONFERENCIA IBEROAMERICANA DE SOFTWARE ENGINEERING: CIBSE, 12., 2009, Medellín, Colombia. **Proceedings…** 2009.

CERI, Stefano; FRATERNALI, Piero; BONGIO, Aldo. Web Modeling Language (WebML): a modeling language for designing Web sites. Proceedings Of The 9th International World Wide Web Conference On Computer Networks. **The International Journal of Computer and Telecommunications Networking,** Amsterdam, The Netherlands, v. 33, n. 1-6, p.137-157, jun. 2000. Elsevier BV. http://dx.doi.org/10.1016/s1389-1286(00)00040-2.

CVS. Free Software Foundation. **Concurrent Version System.** 2015. Disponível em: <http://www.nongnu.org/cvs/>. Acesso em: 20 jul. 2016.

CZARNECKI, Krzysztof; HELSEN, Simon. Classification of Model Transformation Approaches. In: OOPSLA'03 WORKSHOP ON GENERATIVE TECHNIQUES IN THE CONTEXT OF MDA, 2., 2003, Anaheim, Ca, Usa. **Proceedings...** 003. Disponível em: <http://s23m.com/oopsla2003/czarnecki.pdf>. Acesso em: 17 ago. 2016.

DUARTE, Helga; CARDONA, Víctor. Interplay among Software Product Lines, Model Driven Architecture and Service Oriented Architectures for industrial production of software. In: CONTECSI - INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGY MANAGEMENT, 2011, São Paulo. **Proceedings...** 2011. Disponível em:<http://www.contecsi.fea.usp.br/envio/index.php/contecsi/8contecsi/paper/view/2987/1677. Acesso em: 2 jun. 2016.

SOUZA, Vitor Estevao Silva; FALBO, Ricardo de Almeida; GUIZZARDI, Giancarlo. A UML Profile for Modeling Framework-based Web Information Systems. In: INTERNATIONAL WORKSHOP ON EXPLORING MODELING METHODS IN SYSTEMS ANALYSIS AND DESIGN, 12., 2007, Trondheim, Norway. **Proceedings... .** Trondheim, Norway: Ceur-ws, 2007. v. 365, p. 143 - 152. Disponível em: <http://ceur-ws.org/Vol-365/paper15.pdf>. Acesso em: 12 ago. 2016.

FAVRE, Jean-marie; NGUYEN, Tam. Towards a Megamodel to Model Software Evolution Through Transformations.**Electronic Notes In Theoretical Computer Science,** [S.l.], v. 127, n. 3, p.59-74, abr. 2005. Elsevier BV. http://dx.doi.org/10.1016/j.entcs.2004.08.034.

FAYOUMI, Amjad; YANG, Lili. SBVR: Knowledge Definition, Vocabulary Management, and Rules Integrations. **International Journal Of E-business Development,** Melbourne

Austrália, p. 70-76. jan. 2012. Disponível em: <http://www.academicpub.org/ijed/paperInfo.aspx?paperid=583>. Acesso em: 12 ago. 2016.

FONDEMENT, Frédéric; SILAGHI, Raul. Defining Model Driven Engineering Processes. In: INTERNATIONAL WORKSHOP IN SOFTWARE MODEL ENGINEERING (WISME), 3., 2004, Lisbon, Portugal. **Conference Paper...** Lisbon: Epfl, 2004. p. 1 - 7.Foundation, Eclipse. 2015. “Eclipse XText.” https://eclipse.org/Xtext/index.html.

GAILLY, Frederik. Transforming Enterprise Ontologies into SBVR formalizations. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING (CAISE - 2013) CEUR WORKSHOP, 25., 2013, Valencia, Spain. **Proceedings...** Valencia, Spain: Rébecca Deneckère And Henderik A. Proper, 2013. v. 1, p. 1 - 8.

GAMMA, Erich et al. **Design patterns:** Elements of reusable object-oriented software. 2. ed. Mountain View, California: Addison-wesley Professional, 1994. 416 p.

GIT. Software Freedom Conservancy. **GIT.** 2016. Disponível em: <https://git-scm.com/>. Acesso em: 20 jul. 2016.

GOEDERTIER, Stijn; HAESEN, Raf; VANTHIENEN, Jan. EM-BrA2CE v0.1: A Vocabulary and Execution Model for Declarative Business Process Modeling. **Ssrn Electronic Journal,** [S.l.], v. 1, p.1-76, 2007. Social Science Electronic Publishing. Disponível em: <http://dx.doi.org/10.2139/ssrn.1086027>. Acesso em: 19 ago. 2016.

GROENEWEGEN, Danny M.; VISSER, Eelco. Integration of data validation and user interface concerns in a DSL for web applications. **Softw Syst Model,** [S.l.], v. 12, n. 1, p.35-52, 7 set. 2010. Springer Science + Business Media. http://dx.doi.org/10.1007/s10270-010-0173-9.

HAY, David; HEALY, Keri Anderson. **Defining business rules:** what are they really? 2000. Disponível em: <http://www.businessrulesgroup.org/first_paper/br01c0.htm>. Acesso em: 2 ago. 2016.

SBVR INSIDER (Org.). SBVR Speaks: How SBVR Supports the Business Rules Approach. **Business Rules Journal**, v. 7, n. 2, 2006. Disponível em: <http://www.brcommunity.com/p-b274.php>. Acesso em: 23 ago. 2016.

SBVR INSIDER (Org.). **SBVR Speaks: The Key Notions of the SBVR Approach.** 2005. Business Rules Journal, Vol. 6, No. 12 (Dec. 2005), revised (Oct. 2013). Disponível em: <http://www.brcommunity.com/b263.php>. Acesso em: 23 ago. 2016.

JESUS, Jandisson Soares de. **Um método para a implementação de regras de negócio à partir da semântica SBVR**. 2013. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-22012014-144303/>. Acesso em: 2016-08-23.

EIBL, Joachim. **KDIFF3.** 2014. Disponível em: <http://kdiff3.sourceforge.net/>. Acesso em: 20 jul. 2016.

KAMADA, Aqueo; GOVERNATORI, Guido; SADIQ, Shazia. Transformation of SBVR Compliant Business Rules to Executable FCL Rules. **Semantic Web Rules,** [S.l.], n. 6403, p.153-161, 2010. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-16289-3_14.

KLEINER, Mathias; ALBERT, Patrick; BÉZIVIN, Jean. Parsing SBVR-Based Controlled Languages. In: MODELS '09 PROCEEDINGS OF THE INTERNATIONAL CONFERENCE

ON: MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, 12., Berlim Hiedelberg, 2009. **Proceedings…**, 2009. http://dx.doi.org/10.1007/978-3-642-04425-0_10.

KÖVESDÁN, Gábor; ASZTALOS, Márk; LENGYEL, László. A Classification of Domain-Specific Language Intents. **International Journal Of Modeling And Optimization: IJMO.** Barcelona, p. 67-73. fev. 2014. Disponível em: <http://www.ijmo.org/index.php?m=content&c=index&a=show&catid=42&id=397>. Acesso em: 20 jul. 2016

KROISS, Christian; KOCH, Nora; KNAPP, Alexander. UWE4JSF: A Model-Driven Generation Approach for Web ApplicationS.l.**ecture Notes In Computer Science,** [S.l.], p.493-496, 2009. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-02818-2_46. Disponível em: <https://www.pst.ifi.lmu.de/~kochn/icwe2009_kroiss_uwe4jsf.pdf>. Acesso em: 3 ago. 2016.

LEFFINGWELL, Dean; WIDRIG, Don. **Managing software requirements:** a unified approach. [S.l]: Addison-wesley Professional, 1999. 528 p. ISBN 0201615932.

LEVY, Margi; POWELL, Philip. **Strategies for Growth in SMEs:** the role of information and information systems. [S.l..]: Elsevier Butterworth-heinemann Information Systems Series, 2005. ISBN: 978-0-7506-6351-9.

LIDDLE, Stephen W. Model-Driven Software Development. **Handbook Of Conceptual Modeling,** Berlin, p.17-54, 2011. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-15865-0_2.

LINEHAN, Mark H.. Ontologies and Rules in Business Models. In: INTERNATIONAL IEEE EDOC CONFERENCE WORKSHOP**,** 11., 2007. **Proceeedings...** Institute of Electrical & Electronics Engineers (IEEE). 2007. http://dx.doi.org/10.1109/edocw.2007.23.

LINEHAN, Mark H. Semantics in Model-Driven Business Design. In: 2ND INTERNATIONAL SEMANTIC WEB POLICY WORKSHOP (SWPW'06): IN CONJUNCTION WITH THE 5TH INTERNATIONAL SEMANTIC WEB CONFERENCE (ISWC), 2., 2006, Athens, Georgia. **Proceeedings...** 2006. Disponível em: <http://aisl.umbc.edu/resources/319.pdf\#page=92>. Acesso em: 23 ago. 2016.

LUCREDIO, Daniel. **Uma abordagem orientada a modelos para reutilização de software**. 2009. Tese (Doutorado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2009. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-02092009-140533/>. Acesso em: 2016-08-23.

MARINOS, Alexandros; KRAUSE, Paul. An SBVR Framework for RESTful Web ApplicationS.l.**ecture Notes In Computer Science:** Rule Interchange and Applications, Las Vegas, Nevada, v. 5858, p.144-158, 2009. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-04985-9_15.

MARINOS, Alexandros; GAZZARD, Pagan; KRAUSE, Paul. An SBVR Editor with Highlighting and Auto-completion. In: INTERNATIONAL RULEML2011@BRF CHALLENGE, CO-LOCATED WITH THE 5TH INTERNATIONAL RULE SYMPOSIUM, 5., 2011, Fort Lauderdale, Florida. **Proceeedings...** Fort Lauderdale, Florida: CEUR, 2011. v. 799, p. 1 - 8. Disponível em: <http://ceur-ws.org/Vol-799/paper13.pdf>. Acesso em: 20 ago. 2016.

MARINOS, Alexandros; MOSCHOYIANNIS, Sotiris; KRAUSE, Paul. An SBVR to SQL Compiler. In: RULEML-2010 CHALLENGE, AT THE INTERNATIONAL WEB RULE

SYMPOSIUM, 4., 2010, Washington, Dc. **Proceeedings...** Washington, Dc: CEUR, 2010. v. 649, p. 1 - 8. Disponível em: <http://ceur-ws.org/Vol-649/paper7.pdf>. Acesso em: 23 ago. 2016.

MARKIEWICZ, Marcus E.; LUCENA, Carlos J. P. de; COWAN, D. D. Departamento de Informática. **Abstract design views and design patterns.** Rio de Janeiro: PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2000. 28 p.

MERNIK, Marjan; HEERING, Jan; SLOANE, Anthony M.. When and how to develop domain-specific languages. **Csur,** [S.l.], v. 37, n. 4, p.316-344, 1 dez. 2005. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1118890.1118892.

MILNER, Robin. **Communicating and mobile systems:** the [symbol for pi]-calculus. Cambridge, Uk: Cambridge University Press, 1999. 159 p.

MOOLENAAR, Bram (Comp.). **Vim:** the ubiquitous text editor. 2016. Disponível em: <http://www.vim.org/>. Acesso em: 23 ago. 2016.

MORENO, Nathalie; ROMERO, José Raúl; VALLECILLO, Antonio. An Overview Of Model-Driven Web Engineering and the Mda. **Web Engineering:** Modelling and Implementing Web Applications, [S.l.], n. 2, p.353-382, 2008. Springer Science + Business Media. http://dx.doi.org/10.1007/978-1-84628-923-1_12.

MORGAN, Tony 2008. **Business rules and information systems:** aligning it with business goals. 4. ed. Boston, Ma: Addison-wesley Longman Publishing Co., Inc., 2008. 384 p. ISBN 0201743914.

NATALI, Antonio. 2012. **Introduction to EMF, Ecore and XText**. UNIVERSITA' DI BOLOGNA. 2012. Disponível em: http://edu222.deis.unibo.it/ANIS1213/CorsoIS1213BOLM/ target/site/pdf/Models/IntroEmfEcoreXtext.pdf. Acesso em 23 ago. 2016.

NEMURAITE, Lina et al. VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL; models. In: INTERNATIONAL CONFERENCE ON INFORMATION AND SOFTWARE TECHNOLOGIES (IT 2010), 16., 2010, Kaunas, Lithuania. **Proceeedings…** Kaunas, Lithuania, 2010. p. 377 - 384.

NJONKO, Paul Brillant Feuto; ABED, Walid El. From natural language business requirements to executable models via SBVR. In: INTERNATIONAL CONFERENCE ON SYSTEMS AND INFORMATICS (ICSAI2012), Yantai, 2012. **Proceedings**… 2012. p.2453-2457. http://dx.doi.org/10.1109/icsai.2012.6223550.

OBJECT MANAGEMENT GROUP. 2013. **SBVR 1.2**: Semantics of Business Vocabulary and Business Rules (SBVR). 1.2 ed. [s. I.]: OMG, 2013. 292 p. Disponível em: <http://www.omg.org/spec/SBVR/1.2/PDF>. Acesso em: 02 fev. 2015.

———. Object Management Group Model Driven Architecture (MDA) MDA Guide. ed. [s. I.]: OMG, 2014. rev. *2.0*. June. Disponível em: <http://www.omg.org/mda>. Acesso em: 02 fev. 2015.

———. *Object Constraint Language* - OCL. ed. [s. I.]: OMG, 2014a. Disponível em: <http://www.omg.org/spec/OCL/>. Acesso em: 6 jul. 2015.

———. Interaction Flow Modeling Language (IFML). ed. [s. I.]: OMG, 2015. Disponível em: <http://www.omg.org/spec/IFML/1.0/>. Acesso em: 6 jul. 2015.

PARR, Terence. 2015. AntLR. ed. [s. I.]: ANTLR, 2015. Disponível em: http://www.antlr.org/. Acesso em: 2 fev. 2015

RAJ, Amit; PRABHAKAR, T. V.; HENDRYX, Stan. Transformation of SBVR business design to UML models. **Proceedings Of The 1st Conference On India Software Engineering Conference - Isec '08,** Hyderabad, India, p.29-38, 2008. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1342211.1342221.

RIEGGER, Felix. **Test-based Feature Management for Agile Product Lines.** 2010. 93 f. Tese (Doutorado) - Curso de Agile Software Engineering Group, Department Of Computer Science, University Of Calgary, Hs Mannheim, 2010. Disponível em: <http://ase.cpsc.ucalgary.ca/uploads/Publications/DA_Riegger_2010.pdf>. Acesso em: 20 abr. 2016.

RIVERA, José Eduardo; ROMERO, José Raul; VALLECILLO, Antonio. Behavior, Time and Viewpoint Consistency: Three Challenges for MDE. **Models In Software Engineering,** [S.l.], v. 5421, p.60-65, 2009. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-01648-6_7.

ROOVER, Willem de; VANTHIENEN, Jan. A Transformation from SBVR Business Rules into Event Coordinated Rules by Means of SBVR Patterns. **Towards A Service-based Internet. Servicewave 2010 Workshops,** [S.l.], v. 6569, p.172-179, 2011. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-642-22760-8_19.

ROSS, Ronald G.. **Basic RuleSpeak Guidelines:** Do's and Don'ts in Expressing Natural-Language Business Rules in English. 2009. Disponível em: <http://www.rulespeak.com/en/Basic RuleSpeak Dos and Donts v2-2-5.pdf>. Acesso em: 31 ago. 2016.

ROUSSEV, Borislav. Empirical Evidence Justifying the Adoption of a Model- Based Approach in the Course Web Applications Development. **Journal Of Information Technology Education: Research.,** [S.l..], p. 73-90. 1 jan. 2003. Disponível em: <http://www.learntechlib.org/d/111464>. Acesso em: 31 ago. 2016.

RULEARTS. **RuleExpress.** Disponível em: <http://www.rulearts.com/RuleXpress>. Acesso em: 31 ago. 2016.

SCHWABE, D. et al. Engineering Web applications for reuse. **Ieee Multimedia,** Los Alamitos, Ca, v. 8, n. 1, p.20-31, jan. 2001. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/93.923950.

SELWAY, Matt et al. Formalising Natural Language Specifications Using a Cognitive Linguistics/Configuration Based Approach. IEEE INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE, 17., 2013, Vancouver, Bc, 2013. **Proceedings…** Vancouver, Bc: Institute of Electrical & Electronics Engineers (IEEE), 2013. . p.59-68, http://dx.doi.org/10.1109/edoc.2013.16.

SELWAY, Matt; MAYER, Wolfgang; STUMPTNER, Markus. Semantic Interpretation of Requirements through Cognitive Grammar and Configuration. **Lecture Notes In Computer Science,** [S.l.], v. 8862, p.496-510, 2014. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-319-13560-1_40.

SIEGEL, Jon. "Why use the model driven architecture to design and build distributed applications?" In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE '05**,** 27., 2005. **Proceedings…** [S.l.]: Association for Computing Machinery (ACM), 2005. http://dx.doi.org/10.1145/1062455.1062471.

STAHL, Thomas; VOELTER, Markus; CZARNECKI, Krzysztof. **Model-driven software development:** technology, engineering, management. [S.l..]: John Wiley & Sons, 2006. 436 p.

RAIMUND, Eder et al. **Automatic code structure and workflow generation from models:** automatic code structure and workflow generation from natural language modelS.l.ondon: The London School Of Economics And Political Science, 2008. (OPAALS D2.2 Report). Open Philosophies for Associative Autopoietic Digital Ecosystems. Disponível em: <http://www.lse.ac.uk/media@lse/research/OPAALS/D2.2.pdf>. Acesso em: 20 ago. 2016.

SUL, Ricardo Diniz et al. **Estudo de linguagens para representação de regras de negócio:** foco na SBVR. Rio de Janeiro: Universidade Federal do Estado do Rio de Janeiro, 2011. 63 p. Disponível em: <http://www.seer.unirio.br/index.php/monografiasppgi/article/view/1634/1441>. Acesso em: 31 ago. 2016.

VAN DEURSEN, Arie; KLINT, Paul; VISSER, Joost. Domain-Specific Languages: An Annotated Bibliography. **Acm Sigplan Notices,** [S.l.], v. 35, n. 6, p.26-36, 1 jun. 2000. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/352029.352035. Disponível em: <http://people.cs.ksu.edu/~schmidt/505f12/Lectures/DSLbib.pdf>. Acesso em: 16 ago. 2016.

VAN DIJK, David. **Changeability in model driven web development.** 2009. 71 f. Dissertação (Mestrado) - Software Engineering, Faculty Of Science, University Of Amsterdam, Amsterdam, 2009. Disponível em: <http://dare.uva.nl/cgi/arno/show.cgi?fid=143272>. Acesso em: 8 ago. 2016.

VISSER, Eelco. WebDSL: A Case Study in Domain-Specific Language Engineering. **Lecture Notes In Computer Science,** [S.l.], p.291-373, 2007. Springer Science + Business Media. http://dx.doi.org/10.1007/978-3-540-88643-3_7.

WAN-KADIR, W.m.n.; LOUCOPOULOS, Pericles. Relating evolving business rules to software design. **Journal Of Systems Architecture,** [S.l.], v. 50, n. 7, p.367-382, jul. 2004. Elsevier BV. http://dx.doi.org/10.1016/j.sysarc.2003.09.006.

XTEND. **XTend.** 2016. Disponível em: <http://www.eclipse.org/xtend/>. Acesso em: 6 jul. 2016.

XTEXT. **XText.** 2016. Disponível em: <http://www.eclipse.org/Xtext/>. Acesso em: 6 jul. 2016.