ENGENDSL – A DOMAIN SPECIFIC LANGUAGE FOR WEB APPLICATIONS

Carlos Eugênio Palma da Purificação (Universidade Salvador, Bahia, Brasil) – carloseugenio@gmail.com
Paulo Caetano Da Silva (Universidade Salvador, Bahia, Brasil) - paulo.caetano@pro.unifacs.br

Although the use of domain-specific languages is becoming a common practice in software development, there are still challenges ahead for software engineering in its implementation. This article presents some of these challenges and proposes a domain-specific language for Web applications called EngenDSL, which is a declarative specific language created for describing Web application specifications.

ENGENDSL - UMA LINGUAGEM ESPECÍFICA DE DOMÍNIO PARA APLICAÇÕES WEB

Embora o uso de linguagens específicas de domínio estar se tornando uma prática comum no desenvolvimento de software, ainda existem desafios que se apresentam para a engenharia de software na sua aplicação. Este artigo apresenta alguns desses desafios e propõe uma linguagem específica de domínio para aplicações Web chamada EngenDSL, a qual é uma linguagem criada para descrever especificações de aplicações Web.

Palavras-chave: Desenvolvimento baseado em modelos, linguagens específicas de domínio, modelagem de aplicações Web, Engenharia de Sistemas Web.

1. Introdução

Pesquisas na área de reúso de software mostram que para atingir resultados significativos é necessária uma mudança de paradigma no sentido da definição de famílias de software ao invés de sistemas individuais (Czarnecki, 2004). Novas metodologias de desenvolvimento foram definidas, sendo a MDSD – *Model Driven Software Development* uma técnica que está se tornando amplamente aceita para o desenvolvimento de software. A MDSD preconiza a utilização de modelos como elementos principais do processo de desenvolvimento (Moreno et al, 2007), utilizando transformações de modelo-a-modelo e modelo-a-código visando definir a solução final para o software em desenvolvimento (Lin et al, 2005; Czarnecki & Helsen, 2003). Essa técnica considera que os modelos, artefatos de análise e design, possuem o mesmo nível de importância que o código (Stahl et al, 2006). O domínio da Engenharia para Web (*Web Engineering* - *WebE*) é uma área em que os conceitos de MDSD podem ser aplicados com sucesso (Moreno et al, 2007).

Uma das abordagens empregadas para a construção de sistemas utilizando este paradigma é a utilização de Linguagens Específicas de Domínio (*Domain Specific Languages – DSL*) que, em termos gerais, permitem a especificação de conceitos de um domínio de maneira mais expressiva e, exatamente por representarem um domínio específico, operam em um nível maior de abstração que uma linguagem de propósito geral (Deursen, et al, 2000). Embora a utilização de DSLs no desenvolvimento de sistemas não ser uma abordagem nova, é uma prática que tem se disseminado rapidamente nas últimas décadas (Deursen, et al, 2000) e especificamente para sistemas na plataforma Web (Ceri et al, 2000; Visser, 2007).

A evolução e maturação dos conceitos da Engenharia de Software para a Web (*Web Engineering*) motivou a proposta de diversos métodos e *frameworks* para o desenvolvimento de aplicações Web (Souza et al, 2007). Este tipo específico de aplicação apresenta propriedades peculiares, tais como: navegação, menus, índices, regras de negócio e transações, e etc.. Desenvolver sistemas com linguagens sem construções específicas para atender tais propriedades, não é o ideal, posto que estaríamos utilizando primitivas, instruções básicas, com pouca expressividade, para implementar tais requisitos (Rossi et al, 1999). Na área de metodologia de construção de aplicações, especificamente na área de aplicações Web, vários métodos e abordagens de modelagem conceitual e especificação deste tipo de sistema já foram propostos (Souza et al, 2007; Pastor et al, 2000; Ceri et al, 2000; Visser, 2007), além de outros estudos sobre integração e validação dos diversos modelos utilizados para construção de aplicações para esta plataforma (Knapp & Zhang, 2006).

Neste trabalho, partimos de conceitos já estabelecidos nestes estudos anteriores, para servir de apoio à definição de um metamodelo e construtores de uma linguagem específica de domínio, denominada *EngenDSL,* dirigida para especificação de aplicativos para a Web de forma textual que, aliada a um conjunto de técnicas preconizadas em abordagens MDSD, nas quais, não obstante outros trabalhos com direções distintas (Moreno et al, 2007), achamos que há espaço para a aplicação da proposta apresentada neste artigo. A DSL proposta define construtores específicos para a especificação de aplicações Web, de maneira a aumentar expressividade do modelo produzido e permitir a melhoria do entendimento da solução da final, em termos do domínio específico de aplicações Web. A *EngenDSL* é descritiva, e utiliza uma abordagem declarativa na modelagem e implementação para aplicações Web. A linguagem abstrai os principais conceitos para a plataforma Web, sem o uso de construções típicas de linguagens de programação de propósito geral, como *loops* e construções

condicionais (Ceri et al, 2000) e implementa conceitos como modelo de navegação, modelo estrutural e de visualização, além de utilizar padrões estabelecidos como o MVC – *Model View Controller* (Gamma et al, 1995), que serve de ponto de partida para a definição dos construtores da linguagem e criação da solução final de um sistema baseado na plataforma Web.

Este trabalho está assim estruturado: a Seção 2 apresenta os fundamentos teóricos dos principais conceitos sobre desenvolvimento baseado em modelos, linguagens específicas de domínio e engenharia para Web; a Seção 3 define a *EngenDSL* e mostra seu metamodelo, criado para a abstração dos elementos de implementação de uma aplicação Web; a Seção 4 aborda os problemas referentes à transformação modelo-a-modelo e modelo-a-código; a Seção 5 mostra um exemplo da utilização da linguagem em uma aplicação real e o ferramental utilizado; a Seção 6 contempla um resumo de trabalhos correlatos e, na última seção, são apresentadas as conclusões e trabalhos futuros.

2. Fundamentação Teórica

Sob a égide do desenvolvimento de aplicações baseado em modelos (*Model Driven Development* - MDD), existem vários enfoques que começaram a ser desenvolvidos e aplicados (Duarte & Cardona, 2011). Assim, passamos a descrever os mais importantes atualmente, suas áreas de interesse e inter-relacionamentos.

2.1. Model Driven Engineering

O *Model Driven Engineering* - MDE diz respeito à 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 novos conhecimentos na direção 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 próprios conceitos envolvidos no MDE (Stahl et al, 2006; Favre, 2004; Fondement & Silaghi, 2004; Rivera et al, 2008; Liddle, 1999). 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 et al, 2008).

Dentre os vários conceitos relacionados como o MDE, o principal certamente é o conceito de modelo. Este é definido em diversos trabalhos de maneira semelhante, porém com diferenças sutis. Neste trabalho, utilizamos o conceito de que é uma abstração ou simplificação de um sistema, e seu ambiente, visando melhorar ou facilitar seu entendimento (Stahl et al, 2006). Entretanto, para poder ser corretamente utilizado no contexto do MDE, um modelo precisa ter uma sintaxe e semântica bem definidos, 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 idéia 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.

2.2. Model Driven Software Development

Seguindo 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), se encontra o *Model Driven Software Development* (MDSD) (Moreno et al, 2007).

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, guardando os dois 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 et al, 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, 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 destes modelos. O objetivo final é que o processo de desenvolvimento seja melhorado nos quesitos produtividade e qualidade (Stahl et al, 2006).

Uma vantagem significativa no uso de abordagens de geração para o desenvolvimento de software é que ela permite uma maior adequação e consonância com padrões e estilos arquiteturais definidos antes do início da construção, e que estão materializados em artefatos que serão utilizados durante todo o processo de desenvolvimento (Bettin, 2003).

2.3. Model Driven Developement

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 de nível de abstração menor (Duarte & Cardona, 2011).

Alguns autores consideram esta denominação sinônima, e menos precisa, do conceito de *Model Driven Software Development* - MDSD (Stahl et al, 2006), enquanto outros utilizam os termos MDD e MDE com o mesmo significado (Liddle, 1999). Desta forma, 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.

2.4. Model Driven Architecture

O *Model Driven Architecture* - MDA é a iniciativa da OMG - *Object Management Group* na área de MDSD (Moreno et al, 2007), ou MDD. 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 et al, 2006), e outros padrões da OMG como MOF - *MetaObject Facility*, XMI - *XML Metada Interchange*, e o CWM - *Common Warehouse Metamodel* (Liddle, 1999).

Seus principais objetivos são a promoção de interoperabilidade e portabilidade utilizando níveis diferentes de abstrações para modelar todos os aspectos envolvidos na solução, separando modelos de negócio e modelos específicos da plataforma alvo do sistema, permitindo que ambos os modelos evoluam independentemente, e possibilitando, através de transformações modelo-a-modelo e modelo-a-código, a geração da aplicação final (Stahl et al, 2006).

2.5. Model Centric Software Development

Em Schimidt (2006), o MCSD - *Model Centric Software Development*, é citado como uma forma de MDE que preconiza princípios como: utilização de DSLs para representar aspectos específicos da aplicação, sem tentar representar todo o domínio da mesma; a geração automática, e parcial, dos artefatos da plataforma da aplicação baseados nos modelos, dado que os modelos não são suficientes para expressar a implementação final; integração de artefatos de sistemas legados aos artefatos gerados e, finalmente, verificação e validação do modelo baseado na DSL.

Para efeitos práticos da implementação proposta neste trabalho, esta abordagem é perfeitamente compatível como a abordagem MDSD que seguimos.

2.6. Web Engineering

A Engenharia para Web (*Web Engineering – WebE*) pode ser definida como a utilização de princípios e abordagens disciplinadas para o desenvolvimento, implantação e manutenção de aplicações Web (Souza et al, 2007). Estas técnicas e princípios, por tratarem de um domínio específico, abrem espaço para que os conceitos de MDSD possam ser aplicados com sucesso (Moreno et al, 2007). Existem requisitos específicos para a construção de aplicações neste domínio como navegação, apresentação, processos de negócio, acesso a dados e transações.

A construção de aplicações para esta plataforma é uma tarefa que envolve considerável esforço e tempo, e a necessidade de consideração de diversos aspectos como o modelo de domínio, arquitetura de navegação, profiles e personalização, são apenas algumas vertentes que devem ser consideradas. Abordagens como o uso de conceitos de orientação a objetos e frameworks para a modelagem deste tipo de aplicação já foram propostas em diversos trabalhos OO-H (Gómez et al, 2001), OOHDM, OOHDM-Web, OOWS (Pastor et. al, 2000), OO-H, UWA, UWE, WebML (Ceri et al, 2000), WebSA, dentre outras (Schwabe et al, 2001; Distante et al, 2007; Liddle, 1999).

A complexidade envolvida neste tipo de aplicação está em diferentes níveis de abstração, necessárias para a criação de aplicações que atendam aos diversos requisitos do usuário, como sofisticadas aplicações financeiras, médicas, geográficas, facilidade de navegação, portabilidade entre dispositivos, visualização de grande volume de dados multimídia e etc. (Schwabe et al, 2001).

Apesar desta variedade, os conceitos chaves existentes nestas aplicações possuem

estreita semelhança, e continuam a permear a maioria das aplicações para a Web. Tais conceitos estão por vezes disseminados através de padrões específicos para aplicações Hipermídia (Garzotto et al, 1999), enquanto outros, são mais genéricos e aceitos. Como exemplo, temos o padrão arquitetural *Model View Controller* - MVC (Gamma et al, 1995), no qual os três conceitos básicos – controladores (*Controllers*), responsáveis pelo processamento de uma requisição e preparação da resposta, visualizadores (*Views*) responsáveis por renderizar o resultado do processamento para o cliente e o modelo (*Model*) que oferece diversos serviços definidos pelos requisitos da aplicação, ou através de plataformas específicas como o JPA – *Java Persistence API* (*Application Programming Interface*) do JEE (Java EE 6, 2012) para persistência de dados em bases relacionais. Outros padrões típicos destas aplicações são a navegação e visualização baseada em documentos, *workflows*, transações e outros (Liddle, 1999).

2.7. Model Driven Web Engineering

À junção de técnicas de desenvolvimento baseado em modelos e engenharia para a Web dá-se o nome de *Model Driven Web Engineering* - MDWE. Como visto, aplicações Web compartilham alguns padrões específicos e são perfeitas candidatas ao uso da abordagem baseada em modelos. De fato, diversos trabalhos têm sido propostos para modelar este tipo de aplicações e abstrair seus conceitos em um modelo específico (Liddle, 1999).

A Engenharia para Web trata de um domínio específico, em que o MDSD pode ser aplicado com sucesso. Diversas características deste domínio já estão identificadas como navegação e apresentação. Os tipos básicos de aplicações para a plataforma Web já são conhecidos, e um conjunto de padrões arquiteturais e propriedades estruturais também já está mapeado (Moreno et al, 2007).

A abordagem da linguagem definida neste artigo pretende contribuir para esta área, apresentando uma abstração que pode ser utilizada de forma direta em aplicações para a Web, independentemente do ferramental utilizado para suportar os conceitos do MDSD. Entretanto, vale ressaltar, que sem as ferramentas adequadas, a utilização desta abordagem torna-se inapropriada por não poder entregar uma das suas vantagens: o aumento significativo da produtividade na construção de software.

2.8. Domain Specific Languages

Como em todas as áreas da ciência e engenharia, sempre existem abordagens genéricas e específicas para resolver determinado problema (Deursen et al, 2000). A descrição de um problema de um domínio em uma linguagem desenvolvida especificamente para o mesmo, tende a ser uma solução otimizada e direta, além de possivelmente carregar 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 et al, 2005). Deursen et al (2000), define DSLs como linguagens de programação ou linguagens de especificação executáveis que oferecem um grande poder de expressar, com notações e abstrações focadas, e geralmente restritas a um domínio, um problema específico. O estudo e utilização de linguagens específicas de domínio não são recentes, 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.

Outras terminologias e processos são encontrados nos estudos atuais que se referem de

alguma forma às abstrações e conceitos definidos neste trabalho. Em (Stahl et al, 2006), por exemplo, é mostrado o MDSD (*Model Driven Software Development*) que é baseado no conceito do processo de desenvolvimento de software baseado na modelagem específica de domínio, com o uso de DSLs para especificação do modelo e aplicado em várias fases do processo de construção do software. O OMG – *Object Management Group*, propõe e endossa o MDA (MDA, 2012) – *Model Driven Architecture*, que apresenta uma forma similar de desenvolvimento em que o processo é baseado na modelagem do domínio por meio da UML – *Unified Modeling Language* e utilização de *UML Profiles*.

É importante notar que para o desenvolvimento de soluções complexas de software, o designer possivelmente estará em contato com mais de uma DSL (Walter, 2009) e poderá combiná-las para a construção da solução final, o que torna importante a especificação de seu escopo e terminologias. No entanto, tal estudo não é o foco deste trabalho.

3. A *EngenDSL*

Como já foi comentado, ainda não existe um consenso em torno de um modelo, ou metamodelo para aplicações Web. Várias aplicações já foram criadas, testadas e refeitas para a plataforma Web. Uma análise sobre as várias implementações existentes de aplicações para a plataforma Web, mostra as possibilidades de abstração da mesma e, consequente, a possibilidade de criação de uma linguagem de alto nível para definição de seus conceitos.

A *EngenDSL* foi concebida para tirar proveito destas possíveis abstrações, com o intuito de agilizar o desenvolvimento de aplicações para a Web. Algumas das premissas e requisitos utilizados para a especificação desta linguagem foram: (i) utilizar construções declarativas; (ii) encapsulamento dos conceitos do padrão arquitetural MVC - *Model View Controller* (Gamma et al, 1995); (iii) não possuir construções típicas de linguagens genéricas de alto nível como condicionais, laços (*loops*), definição ou chamada métodos com parâmetros; (iv) as abstrações devem ser definidas de forma a contemplar o maior número de arquiteturas possíveis, sem se comprometer com nenhuma especificamente; (v) o número de instruções deve ser o menor possível, e que possa expressar as semânticas necessárias para aplicações Web; (vi) a linguagem não precisa ser expressiva suficiente para definir todos os conceitos da aplicação, ou seja, outras transformações podem ser necessárias para a definição completa das funcionalidades da aplicação; (vii) as construções definidas na linguagem alvo do sistema devem poder ser integradas ao sistema em desenvolvimento, sem prejuízo da semântica e sintaxe da linguagem.

Assim, o principal objetivo perseguido na construção da *EngenDSL* foi a de construir uma abstração declarativa para a construção de aplicações Web, evitando ao máximo construções típicas de programação como estruturas condicionais e laços (loops) e o comprometimento com uma tecnologia específica. Além disto, requisitos de integração com componentes desenvolvidos na linguagem alvo (e.g.: *Java*, *C#*, etc) são definidos, de maneira que a linguagem possui meios de integração com componentes desenvolvidos manualmente pelos desenvolvedores.

Na *EngenDSL* estes conceitos estão atrelados e conectados de forma a fornecer um modelo único para modelagem de aplicações Web. A Figura 1 mostra um modelo conceitual simplificado destes elementos e suas relações, os quais serão descritos a seguir:

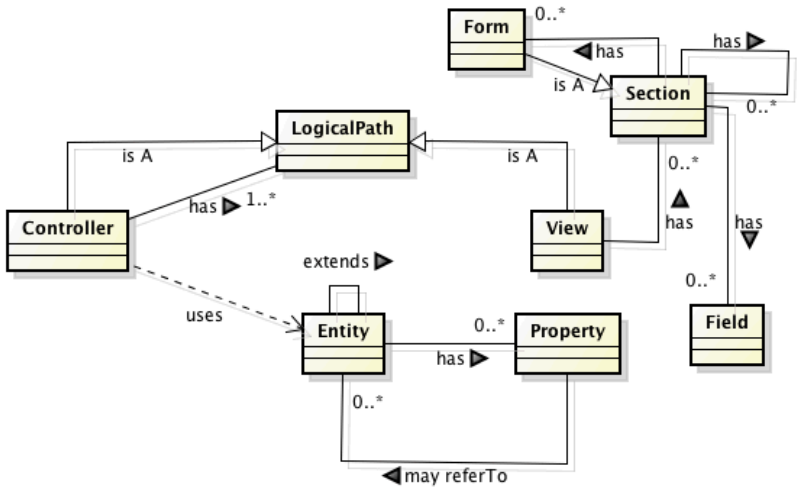

Illustration 1

Figura 1 - Modelo Conceitual da EngenyDSL.

3.1. EngenDSL – metamodelo

A seguir é definida a estrutura da linguagem, seus principais conceitos e notações. Por razões de brevidade omitimos algumas partes de modelo para dar uma visão geral de seu conteúdo e possibilidades. Desta forma apenas as partes principais do MVC serão descritas como a definição das *Views, Model* e *Controllers*.

O metamodelo da linguagem é definido utilizando o projeto *XText* (*Eclipse XText,* 2012) da fundação *Eclipse* (*Eclipse Project*, 2012). Este componente permite, como será mostrado a seguir, a definição de linguagens específicas de domínio e a criação de um editor de texto completo, para a linguagem que está sendo definida, com funcionalidades similares as de editores para linguagens de propósito geral e integrado à plataforma *Eclipse*.

Entidades

No conceito de aplicações *Web*, via de regra, temos presente a idéia de entidades do modelo que representam conceitos do mundo real. Conforme os requisitos funcionais da própria linguagem, tais conceitos representam entidades advindas dos requisitos do sistema e precisam ser armazenados de forma a se preservar uma memória sobre as mesmas. Tais entidades são definidas na linguagem através da palavra chave "*entity*".

Cada entidade na *EngenDSL* possui os seguintes papéis:

- Representar uma entidade, possivelmente com dados de negócio a serem persistidos;
- Definir os relacionamentos entre a entidade e outras entidades do modelo;
- 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 a seguir:

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

StructuralFeatureDef:
    AttributeDef | ReferenceDef
;
```

Figura 2 - Definição da sintaxe da declaração de uma entidade do modelo.

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 *EngenDSL* por tratar-se de uma simples definição da extensão de um conceito com semântica própria. Na linguagem, o *NamespaceDef* é uma forma de identificar o pacote ao qual algum elemento da linguagem está associado. O construtor *StructuralFeatureDef* permite definir 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.

Controllers

Para satisfazer o requisito de abstrair na linguagem os conceitos do padrão MVC, a *EngenDSL* utiliza diretamente o conceito de "*Controllers*" que representam, de acordo com este padrão, componentes que intermediam a entrada e saída de dados tanto para o "*Model"* do MVC, quanto para a "*View*".

Em algumas implementações de sistemas utilizando este padrão, encontramos diferentes comportamentos para este conceito, principalmente no que se refere a como o "*Controller*" é acionado. A *EngenDSL*, em seu nível conceitual, não se interessa como ele é acionado - a linguagem define que existe um controlador que intermedia uma requisição, utilizará um elemento do modelo e uma *View,* ou outro *Controller*, que realizam a interface *LogicalPath,* e que será acionado em resposta a requisição.

A Figura 3 mostra como um *Controller* é definido utilizando a *EngenDSL*.

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

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

ControllerBodyElements:
        namespaceDeclaration=NamespaceDeclaration?
        moduleDeclaration=ModuleDeclaration?
        (
            (target=ControllerTargetDeclaration)?
            (label=LabelPropertyDef)?
            (paths+=LogicalPathDeclaration)+
        )
;
```

Figura 3 - Definição da sintaxe da declaração de um Controller na EngenDSL.

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

Views são utilizadas para apresentar, ou capturar, algum dado do usuário segundo o padrão MVC. Na linguagem *EngenDSL*, 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*. Estruturalmente as *Views,* na linguagem, podem possuir seções. Tais seções representam uma parte de uma página *Web* e podem ser de diversos tipos como uma tabulação de elementos na tela, divisões ou formulários *Web* que contém dados que devem ser enviados para a aplicação. É importante notar que, no conceito da *EngenDSL,* 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 4 é mostrado um trecho da definição da *EngenDSL* para *Views*. Alguns esclarecimentos sobre os conceitos apresentados nesta definição são discutidos a seguir.

```
ViewDef:
    "view" name=ID (":" type=ControllerType ":" targetEntity=[EntityDef])? "{"
        (
            //These entries are unorderd
            (namespace=NamespaceDef)?
            (label=LabelPropertyDef)?
            (module=ModuleDef)?
        )
        (sections+=SectionDef)*
    "}";
```

Figura 4 - Definição da sintaxe da declaração de uma View na EngenDSL.

A definição de uma *View* engloba aspectos visuais, estruturais e funcionais da mesma. Por exemplo, a definição da propriedade "*module*" da *View* especifica a qual módulo funcional da aplicação esta *View* faz parte. Como visto, estruturalmente *Views* são constituídas de seções. A Figura 5 mostra a especificação da *EngenDSL* para uma seção de

uma *View*. Nela estão definidas as principais características estruturais de uma *View*.

```
SectionDef:
    "section" name=ID
        (":" extends=[SectionDef])?
        (":" type=SectionType)? "{"
        (
            //These entries are unorderd
            (namespace=NamespaceDef)?
            ("target:" target=SectionFormTargetDef ";")?
        )
        (subSection+=Subsection)*
    "}"
;
```

Figura 5 - Definição da sintaxe da declaração de uma seção (Section) na EngenDSL.

Conforme esta definição, seções de uma *View* começam com a palavra *"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 *View* pode conter,

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

conforme a definição mostrada a seguir:

Figura 6 - Definição de uma subseções de uma View na EngenDSL.

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

- ***SectionDef***: chamada recursiva ao conceito de uma seção de uma *View*. Significa dizer que uma seção pode conter outras seções.
- ***ViewField***: define um campo de uma *View,* podendo ser campos de entrada de texto, datas, listas de opções, tabelas, botões e rótulos.
- ***ScanDefinition***: definição que permite evitar ter que especificar cada campo que a *View* exibirá para os atributos de uma entidade (*entity*), permitindo que simplesmente seja feita uma referência à entidade. Apesar deste não ser o método preferido, pois entidades podem ter campos que serão mostrados, ou não, em diferentes *Views*, a *EngenyDSL* permite este tipo de construção para efeito de concisão do modelo.
- ***LabelPropertyDefinition***: definição de um rótulo para a seção.
- ***PutFieldsPropertyDefinition***: auxiliar à função *ScanDefinition* que permite definir especificamente quais campos serão mostrados. Esta definição irá colocar na *View* todos os campos definidos pela definição de *ScanDefinition* à qual sucede.

LogicalPaths

O conceito de *LogicalPath* está diretamente associado ao conceito de navegação entre *Views* e *Controllers*. 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, *Views* e *Controllers* representam, e são extensões, de *LogicalPaths* dentro da *EngenDSL*. Assim, um *Controller* poderá definir, de acordo com sua lógica de execução, redirecionar o usuário para um caminho – *LogicalPath*, que no final renderizará uma página *Web* 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 fins de concisão, e atender aos requisitos da linguagem, conforme definido no início 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 *Controlller*. 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*.

4. Transformação entre modelos

A definição de uma linguagem expressiva, para qualquer aplicação Web, envolve a definição de vários detalhes que só podem ser capturados quando todas outras definições e requisitos funcionais, de navegação, execução dos *Controllers* e entidades, estão claros. Por outro lado, requisitos de implantação e apresentação ligados diretamente a uma tecnologia específica de uma determinada plataforma Web como estilos, cores e fontes, não devem estar contemplados na DSL, sob pena de ela repetir, em um nível de abstração mais alto, a representação de outras DSLs, que estão definidas em um nível mais baixo de abstração como o CSS e HTML.

Tendo estes problemas em vista, a abordagem da *EngenDSL* admite uma fase intermediária de transformação modelo-para-modelo, antes da transformação final para o código da plataforma alvo da aplicação, visando permitir que tais requisitos, externos à linguagem, possam ser especificados.

Tal extrapolação dos conceitos presentes na linguagem é importante e deve fazer parte do processo de desenvolvimento de cada solução. Para isto a *EngenDSL* utiliza um modelo intermediário que deve ser utilizado pelas ferramentas de interface com o desenvolvedor do sistema, além de templates específicos para a arquitetura alvo definida para o sistema, que transformarão os dados presentes no modelo em artefatos na plataforma escolhida, permitindo a finalização da configuração do sistema, compilação e implantação da solução final,

5. Aplicação da *EngenDSL*

Nesta seção é mostrada uma parte de uma aplicação modelada com a linguagem *EngenDSL*. Serão mostradas as entidades, *Views* e *Controllers* modelados de acordo com os conceitos apresentados anteriormente e que definem um fluxo completo para as operações de *CRUD* (*Create, Retreive, Update and Delete*) em uma entidade.

A ferramenta utilizada para implementar os conceitos da *EngenDSL* é o *XText* da

fundação *Eclipse*. Esta ferramenta possibilita que uma interface de desenvolvimento de sistemas popular entre desenvolvedores, seja utilizada para implementar os conceitos da linguagem, bem como fornecer utilitários de interface com o usuário como comentários em *HTML*, complementação de código sensível ao contexto e refatoração.

Na ferramenta proposta, o designer especifica uma DSL, define, portanto, o metamodelo que será instanciado para o usuário utilizar a DSL. Quando instanciado, o metamodelo proverá a coerência sintáxica e semântica para a criação do modelo final da aplicação. Nesta ferramenta, por exemplo, o usuário pressionaria um botão e as validações ocorrem mostrando as possíveis inconsistências.

Para este exemplo estamos utilizando uma entidade comum na maioria de sistemas Web, pois representa o papel do próprio usuário que está interagindo com o sistema.

```
/**
* The roles can group permissions, but the main purpose of them is assign authorization
* URLs security. These rules for accessing URLs are defined in the UrlPattern entity.
*/
entity Role : SecurityRole {
    namespace: security;
    name: string;
    permissions: Permission[];
}
```

A entidade *Role* é definida como segue:

Figura 7 - Implementação de uma entidade seguindo a notação da EngenDSL.

A entidade *Role*, conforme definido pelo metamodelo da linguagem estende a entidade *SecurityRole*. São definidos a entidade, seus atributos, referências (apesar de não mostrado aqui a declaração da entidade *Permission*, a propriedade *"permissions"* da entidade *Role* é uma referência a uma coleção de entidades *Permissions*; mais adiante será mostrado como a transformação do modelo para a plataforma final irá interpretar esta informação em forma de uma tabela e um botão permitindo a associação definida no modelo) e escopo, todos de forma declarativa.

Na formalização da entidade "*Role*", já é importante notar o suporte da ferramenta ao reconhecimento da sintaxe e comentários conforme a diferença de cores dos termos evidencia. Outras funcionalidades providas pela plataforma são a refatoração e complementação automática do código, definição de fontes e cores para cada construção da linguagem, criação automática de uma árvore visual dos termos definidos no modelo, dentre outras.

A seguir é mostrado como um *Controller* pode ser definido:

```
//Cria um Role
controller CreateRoleController : Create {
    namespace: rolesControl;
    module: security;
    target: Role to role;
    label: "Criar Role";
    success: SearchRolesView;
    failure: NewRoleView;
    cancel: SearchRolesView;
}
```

Figura 8 - Implementação de um Controller seguindo a notação da EngenDSL.

Controllers podem definir um módulo (*module: security*, neste exemplo) ao qual estão atrelados. Na verdade, módulos na *EngenDSL* são somente nomes lógicos de agrupam *Controllers*. A definição da propriedade "*target: Role...*" do *Controller,* faz uma associação entre este e o model "*Role*". Três declarações de *LogicalPaths* estão presentes referenciando prévias declarações de *Views*. Conforme mostrado na seção 3, as Views são entidades de primeiro nível na linguagem e abstraem o conceito de apresentação de dados para uma aplicação Web. Algumas destas *Views* estão definidas como mostrado no exemplo a seguir:

```
//Tela de entrada de dados para criacao de um Role
view NewRoleView {
    namespace: roles;
    module: security;
    section NewRoleFields : form {
        target: CreateRoleController;
        label: "Criar Role";
        scan: entity Role as roleFields;
        putFields: roleFields;
        field CreateRoleButton : button {
            id: createRoleButton;
            label: "Criar Role";
            target: CreateRoleController;
        }
        field CancelarCriacaoRoleButton : cancel {
            id: cancelarCriacaoRoleButton;
            label: "Cancelar";
            target: SearchRolesController;
        }
    }
}
```

```
// Mostra a tela de edicao de Roles
view EditRoleView {
    namespace: roles;
    label: "Alterar Role";  //Titulo da tela
    module: security;
    section EditRoleFields : form {
        target: EditRoleController;
        label: "Campos de Role";
        scan: entity Role as roleFields;
        putFields: roleFields;
        field AtualizarRoleButton : button {
            id: atualizarRoleButton;
            label: "Atualizar Role";
            target: UpdateRoleController;
        }
        field ExcluirRoleButton : button {
            id: excluirRoleButton;
            label: "Excluir Role";
            target: DeleteRoleController;
        }
        field CancelarAlteracaoRoleButton : cancel {
            id: cancelarAlteracaoRoleButton;
            label: "Cancelar";
            target: SearchRolesController;
        }
    }
}
```

Figura 9 - Implementação de uma View seguindo a notação da EngenDSL.

Figura 10 - Implementação de uma View seguindo a notação da EngenDSL.

Views são associadas à *Controllers* e possuem formulários e campos. O campo

"*target*" define qual será o *Controller* selecionado quando um campo do tipo "*button*" for acionado pelo usuário final da aplicação. Similarmente outras *Views* e *Controllers* são definidos, provendo o sistema de uma especificação declarativa de suas funcionalidades ao invés de programação e código repetido. A seguir, a tela do sistema gerado automaticamente correspondente à *View NewRoleView* definida acima:

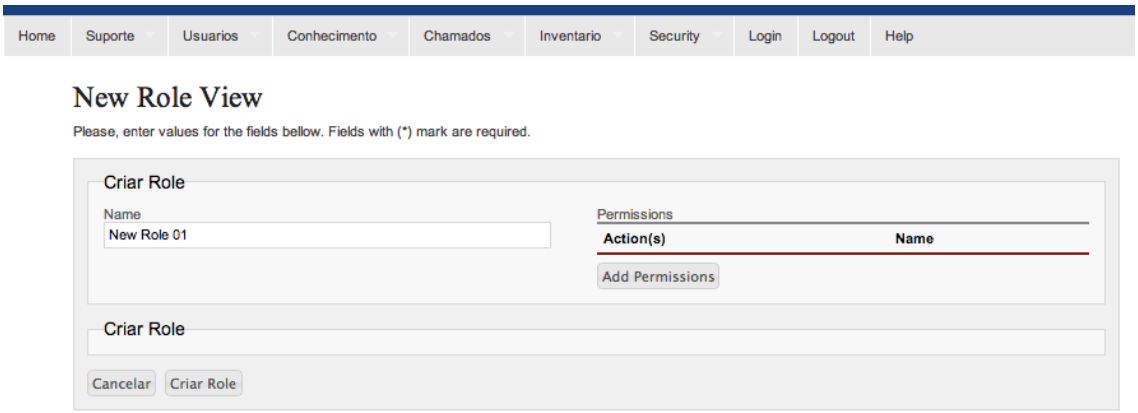

Figura 11 - Demonstração de uma View definida utilizando a EngenDSL em um aplicativo execução.

Como visto, com um metamodelo inicial reduzido, se comparado com outros trabalhos como a *WebML* (Ceri et al, 2000), *WebSA* (Melia et al, 2003), *FrameWeb* (Souza et al, 2007) e *WebDSL* (Visser, 2007), a proposta da *EngenDSL* engloba os principais conceitos de aplicações Web, como os de navegação (realizados pelos *LogicalPaths*), composição da apresentação (realizados pelas *Views* e *Sections* e seus sub-componentes) e o estrutural de dados (realizados por *Entities*).

6. Trabalhos Correlatos

Diversos trabalhos já foram propostos para modelagem e especificação de sistemas Web. WebML (Ceri, et al, 2000), WebSA (Melia et al, 2003), FrameWeb (Souza et al, 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.

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). A WebSA (Melia et al, 2003) propõe a extensão e enriquecimento destes modelos utilizando a abordagem MDA (MDA, 2012) para abranger aspectos arquiteturais de aplicações Web.

Em (Distante et al, 2007) é descrita uma abordagem utilizando o framework UWA (*Ubiquitous Web Application*), com o padrão arquitetural MVC (*Model View Controller*), a

tecnologia JSF - *JavaServer Faces* e o processo de desenvolvimento baseado em modelos – MDD (*Model Driven Development*). Apesar das semelhanças com o presente trabalho 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 *EngenDSL* apresentada por este 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, podem 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.

Em (Knapp & Zhang, 2006) é proposto um método de integração e validação da integração entre os diversos modelos (navegação, aspectos, conteúdo, lógica de negócio e apresentação) para sistemas Web.

A maioria das propostas similares avaliadas neste trabalho baseia-se principalmente no uso de UML – *Unified Modeling Languagem* (uma DSL visual) ou XML para a especificação dos modelos. Outras propostas mais recentes, incluindo a descrita neste artigo, procuram algum tipo de DSL textual (Visser, 2007) para a especificação dos conceitos de aplicações Web. Neste sentido a proposta que guarda mais similaridade com a deste artigo é a *WebDSL* (Visser, 2007). Ela define uma expressiva linguagem para definir aplicações Web, abrangendo seus vários aspectos. Entretanto, em um sentido diferente da abordagem da proposta neste artigo, a linguagem extrapola a simples declaração destes conceitos para especificar estruturas de funções ou métodos, condicionais e inclusive algoritmos, que são utilizados para definir comportamento de certas partes da arquitetura e componentes da aplicação, possivelmente criando uma maior complexidade no processo de desenvolvimento.

7. Conclusão e trabalhos futuros

Linguagens específicas de domínio são uma tendência tecnológica. Sua aplicação em sistemas Web são perfeitamente factíveis e até recomendadas. Não só pelo aumento de produtividade que esta aplicação pode alcançar, mas também por todos outros benefícios que uma abordagem de desenvolvimento baseado em modelos e geração automática de artefatos podem trazer à Engenharia de Sistemas Web: redução de falhas, padronização da arquitetura e solução, abordagem tecnológica adequada e adaptável, além de permitir uma evolução segura entre as diversas tecnologias e frameworks que surgem constantemente, para este tipo de aplicação.

Este trabalho procurou mostrar como uma DSL, desenvolvida para o domínio específico da Web, pode ser definida de maneira declarativa e de fácil utilização.

Foi mostrado como a *EngenDSL* implementa os conceitos do MVC, notadamente com seus conceitos de *Controllers*, *Views* e *Entities* (*Model*), que podem servir como base para a definição e criação de aplicações Web. Tais conceitos estão materializados na linguagem e servem como ponto de partida para a criação de outras extensões.

Definição de restrições da DSL, processo de depuração de erros, integração de código e componentes desenvolvidos na linguagem alvo do sistema, validação sintáxica e semântica e integração horizontal com outros modelos, estão entre trabalhos que serão desenvolvidos e

documentados futuramente para a linguagem. Também, estão previstos para a *EngenDSL* a criação de várias extensões, que permitam definir, declarar e validar outros aspectos como testes, composição de *Views*, *Web Services*, BPM (*Business Process Modeling*) como meio de definição de processos de negócio, dentre outros.

Referências

Bettin, J. (2003). Best practices for component-based development and model-driven architecture, pp. 1-12.

Ceri, S., Fraternali, P., Bongio, A. (2000). Web modeling language (WebML): a modeling language for designing web sites. Computer Networks 33(1-6): 137-157.

Czarnecki, K., Helsen, S. (2003). Classification of model transformation approaches. In OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, California. http://users.dsic.upv.es/~einsfran/mda/czarnecki_helsen.pdf, pp. 1-13.

Czarnecki, K. (2004). Overview of generative software development. Em J.-P. Bantre et al,, editoras, Unconventional Programming Paradigms (UPP'04), volume 3566 of Lecture Notes in Computer Science, páginas 313–328, Mont Saint-Michel, França.

Deursen A., Klint P., Visser J. (2000). Domain-specific languages: an annotated bibliography, pp. 1-4.

Distante, D., Pedone, P., Rossi, G., Canfora, G. (2007). Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces. In Baresi, L., Fraternali, P., Houben, G.J., eds.: ICWE. Volume 4607 of Lecture Notes in Computer Science., Springer (2007) p.p. 457-472.

Duarte, H., Cardona V. (2011). Interplay among software product lines, model driven, architecture and service oriented architectures for industrial production of software. Em 8o CONTECSI - International Conference on Information Systems and Technology Management, pp. 1166-1192.

Eclipse Project “http://www.eclipse.org/ ”. Acessado em 21-05-2012.

Eclipse XText “http://www.eclipse.org/XText/”. Acessado em 21-05-2012.

Favre, J. (2004). Towards a basic theory to model driven engineering. ADELE Team, Laboratoire LSR-IMAG pp. 1-13.

Fondement, F., Silaghi, R. (2004). Defining Model Driven Engineering Processes. In Proce. of the 3rd Workshop in Software Model Engineering, WiSME pp. 1-11.

Gamma, E., Helm, R., Johnson, R., Vlissides J. (1995). Design patterns: elements of reusable object-oriented software. Addison-Wesley, pp. 4 -6.

Garzotto, F., Paolini, P., Bolchini, D., Valenti, S., (1999). Modelling by patterns of Web Applications. Proc. Int’I Workshop on the World Wide Web and Conceptual Modeling. Springer Verlag, Berlin, pp. 1-14.

Gómez, J., Cachero, C., Pastor, O. (2001). Conceptual modeling of device-Independent web applications. IEEE MultiMedia pp. 26-39.

Java EE 6. Disponível em "http://www.oracle.com/technetwork/java /javaee/tech/index.html". Acessado em 27/01/2012.

Knapp, A., Zhang, G. (2006). Model Transformations for Integrating and Validating Web Application Models. In Proc. Modellierung 2006 (MOD 2006), Vol. P-82, Lect. Notes in Informatics, 115-128, Innsbruck, Austria, pp. 1-14.

Liddle, S. W. (1999). Model-driven software development pp. 1-39.

Lin, Y., Zhang, J., Gray, J. (2005). A testing framework for model transformations. Model-driven Software Development, Springer, Chapter 10.

MDA "http://www.omg.org/mda". Acessado em 21-05-2012.

Melia, S., Cachero, C., Gomez, J. (2003). Using MDA in web software architectures. In *Proc. 2nd OOPSLA Wsh. Generative Techniques in the Context of Model Driven Architecture,* Anaheim, pp. 1-6.

Mernik, M., Heering, J., Sloane, A. (2005). When and how to develop domain-specific languages. ACM Computing Surveys, vol. 37, no. 4, pp. 316-344.

Moreno, N., Romero, J. R., Vallecillo, A. (2007). An overview of model-driven web engineering and the MDA. Departamento Lenguages y Ciencias de la Computación - UMA. Chapter 12.

Pastor O., Abrahão S. M., Fons J. J. (2000). OOWS: An Object-Oriented Approach for WebSolutions Modeling. In Proc. Media in Information Society (MEIS'00), Ljubljana Slovenia, 2000, pp.127-128.

Rivera, J.E., Romero, J.R., Vallecillo, A. (2008). Behavior, time and viewpoint consistency: Three challenges for MDE. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering, pp. 20–24.

Rossi, G., Schwab, D., Lyardet, F. (1999). Web application models are more than conceptual models, pp. 239-252.

Schimidt, D. C. (2006). Model-driven engineering. IEEE Computer, February 2006 (Vol. 39, No. 2) pp. 25-31.

Schwabe, D., Rossi, G., Esmeraldo, L., Lyardet, F. (2001). Engineering web applications for reuse, pp. 1-13.

Souza, V., Falbo, R., Guizzardi, G. (2007). A UML profile for modeling frameworkbased web information systems. In Proceedings of the 12th International Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'07) held in conjunctiun

with the 19th Conference on Advanced Information Systems (CAiSE'07) (Vol 365) pp. 143-152.

Stahl, T., et. all (2006). Model-driven software development technology, engineerring, management, pp. 3-26.

Thiemann, P. (2003). An embedded domain-specific language for type-safe server-side web scripting, pp. 2-8.

Visser, E. (2007). WebDSL: A case study in domain-specific language engineering, generative and transformational techniques in software engineering. GTTSE, Lecture Notes in Computer Science, Springer (2008) Tutorial for International Summer School GTTSE, pp. 1-60.

Walter, T. (2009). Combining domain-specific languages and ontology technologies, pp. 1-6.