Explorando o Refinamento de uma DSL para Versões Baseadas em EMF, Eclipse Sirius e XText

Jaqueline Moura[1], **Natiele Lucca**[1], **Fábio P. Basso**[1], **João Pablo S. da Silva**[1], **Elder Rodrigues**[1], **Maicon Bernardino**[1]

[1]Campus Alegrete – Universidade Federal do Pampa (UNIPAMPA) – Alegrete – RS – Brasil

{moura.jak,natielelucca}@gmail.com

{fabiobasso,joaosilva,elderrodrigue}@unipampa.edu.br, bernardino@acm.org

Abstract. *Model Driven Engineering (MDE) allows specify abstract models at high levels and then transform them. Self-adaptive systems are able to assess and change their own behavior at run time. Domain-specific languages (DSLs) can be described to abstract out such complexity. This article presents an exploratory study on the refinement of a DSL in this domain. Three versions are implemented: one textual (in Xtext) and two graphics (based on EMF tree and Eclipse Sirius). Afterwards, transformations of these models are carried out for target platform code. Finally, conclusions from the practice obtained with the apparatus used in the Eclipse ecosystem are exposed.*

Resumo. *A Engenharia Dirigida por Modelos (MDE) possibilita descrever modelos abstratos em altos níveis e então transformá-los. Sistemas autoadaptativos são capazes de avaliar e alterar seu próprio comportamento em tempo de execução. Linguagens específicas de domínio (DSLs) podem ser descritas para abstrair tal complexidade. Este artigo apresenta um estudo exploratório no refinamento de uma DSL desse domínio. Três versões são implementadas: uma textual (em Xtext) e duas gráficas (baseada em árvore EMF e Eclipse Sirius). Após, são realizadas transformações destes modelos para código de plataforma alvo. Finalmente, são expostas conclusões da prática obtida com o aparato utilizado no ecossistema do Eclipse.*

1. Introdução

Um sistema autoadaptativo é capaz de avaliar e alterar seu próprio comportamento, sempre que a avaliação mostrar que o software não está realizando o que se pretendia fazer, ou quando uma melhor funcionalidade ou desempenho pode ser possível. Assim, a investigação de abordagens sistemáticas de engenharia de software é necessária, a fim de desenvolver sistemas autoadaptativos que possam, idealmente, ser aplicados em vários domínios [Macás-Escrivá et al. 2013]. Estes sistemas podem ser aplicados a diversas áreas de conhecimento como: 1) saúde – monitoramento de sinais vitais, pressão arterial e temperatura); 2) aplicações tecnológicas como veículos autônomos; 3) dispositivos móveis como controlar a luminosidade e o acelerômetro; entre outros[1].

[1]Mais exemplos de sistemas auto-adaptativos podem ser encontrados em https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

Sistemas autoadaptativos vem sendo alvo de investigação em pesquisas aplicadas com MDE [Schmidt 2006]. MDE é uma abordagem de desenvolvimento de software que se concentra na criação de modelos que descrevem os elementos de um sistema. Os modelos respeitam um modelo preciso (sintaxe) e um significado (semântica) [Petricic et al. 2008] conforme uma Linguagem Específica de Domínio (DSL) [Voelter 2009].

Pesquisas sobre MDE identificaram a carência literária de estudos exploratórios considerando diferentes tecnologias de construção de DSLs [Neto et al. 2020, Neto et al. 2019]. Em virtude disso, apresenta-se neste artigo um estudo exploratório caracterizando três opções disponíveis no ecossistema do Eclipse: EMF, Eclipse Sirius e XText. Para tal, uma investigação para abstrair os elementos de sistemas autoadaptativos em modelos foi conduzida, tendo como eleita uma versão preliminar e simplificada da DSL denominada SaSML [da Silva 2018]. Nossa contribuição é um relato de experiência sobre o desenvolvimento de três protótipos desta mesma DSL, utilizando estratégias de representação diferentes.

O restante do trabalho está organizado da seguinte maneira: A Seção 2 discute a metodologia adotada nessa pesquisa exploratória. A Seção 4 introduz o metamodelo da SaSML e ambas as versões de DSL criadas. A Seção 3 apresenta brevemente as tecnologias utilizadas. O mapeamento e transformação de modelos é discutido na Seção 5. Por fim, a Seção 6 aborda trabalhos relacionados e a Seção 7 as considerações finais acerca do trabalho.

2. Motivação

Como exemplificação do estudo, a Figura 1(A) apresenta uma versão preliminar dos elementos esperados da DSL SaSML. Ela introduz o elemento AdaptiveBehavior como forma de encapsular um modelo complexo composto de contextos e comportamentos. Este exemplo descreve o identificador do elemento como Comportamento-Adaptativo, as informações de contextos (C) e os comportamentos alternativos(B).

Alguns desafios para a concepção dessa DSL como uma ferramenta importante para a área incluem: 1) uma vez que a transferência de tecnologia de MDE é altamente dependente de experiências, percebe-se que não há um corpo de conhecimento em português que relate sobre a utilização de diferentes tecnologias na construção da mesma DSL; 2) uma vez que algumas DSLs em nossos grupos de pesquisa estão em seu estágio embrionário, não se tem dados práticos sobre qual tecnologia é viável para construí-las; e 3) uma vez que cada domínio de aplicação pode ter maior aceitação por determinados tipos de DSL [Voelter 2009], não se tem dados sobre qual forma de representação (se em árvore, gráfica ou a textual) é mais bem aceita pelos usuários finais na modelagem de sistemas em geral.

Diante destes desafios, buscou-se resolver o primeiro. Nosso foco neste estudo é realizar uma pesquisa exploratória sobre três tecnologias na construção da SaSML: EMF [Steinberg et al. 2008], Eclipse Sirius [Eclipse Sirius 2018] e XText [XText 2018]. Portanto, apresenta-se uma fundação que serve como base para a execução das próximas pesquisas, mais empíricas, sobre os dois desafios subsequentes.

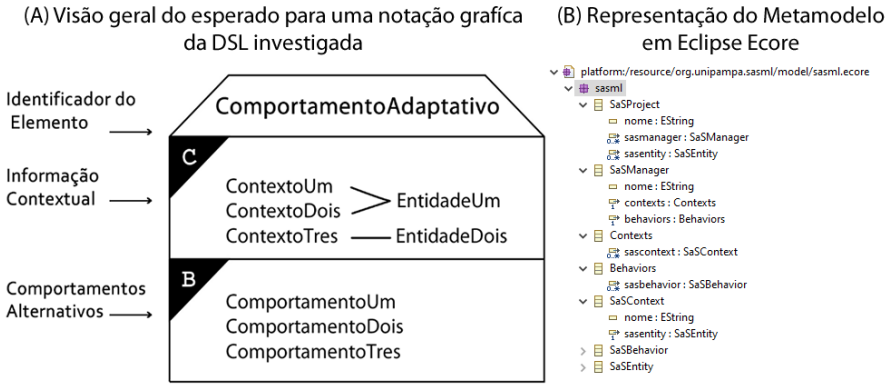

Figura 1. Elementos representacionais da DSL e Metamodelo Ecore

3. Tecnologias Utilizadas

Esta seção descreve brevemente as principais tecnologias que foram integradas para produzir esse trabalho.

3.1. EMF

O Eclipse Modeling Project provém e evolui diversas tecnologias de desenvolvimento baseada em modelo. Seu guarda chuva possui um conjunto unificado de *frameworks* de modelagem, implementações de padrões e fornece o aparato ferramental para sua utilização.

Dentro de um projeto de modelagem, precisamos inicialmente definir a estrutura do modelo de domínio, essa definição é denominada meta modelo. No eclipse, o meta modelo é determinada pela linguagem Ecore por meio do editor EcoreTools. Modelos Ecore possuem suporte de tempo de execução, notificação quando alterações são realizadas e são persistidos em XMI serializado. Sendo, portanto, parte essencial do *framework* EMF.

3.2. Eclipse Sirius

O eclipse Sirius utiliza as tecnologias mencionadas anteriormente para produzir um ambiente de modelagem integrado. Com ele é possível definir uma bancada de trabalho com todas as ferramentas necessárias para a produção de modelos gráficos personalizados, como diagramas, tabelas e arvores [**?**]. Ele provém facilidade de especificação de todas as características do modelo, comportamentos da linguagem modelada e regras validação sincronizada dinamicamente. Além disso, ele permite a especificação de diversas representações *(viewpoint especification)* para um único modelo. Podendo então, prover diversas camadas de visualização de acordo com as diferentes preocupações de cada tipo de usuário.

3.3. Xtext

Xtext é uma estrutura do Eclipse para implementar linguagens de programação e DSL textuais. Permite implementar idiomas rapidamente e, acima de tudo, abrange todos os aspectos de uma infra-estrutura completa de linguagem, desde o analisador, gerador de código, ou intérprete, até uma integração completa do Eclipse IDE com

todos os recursos típicos da IDE [Bettini 2016]. Ele também depende do EMF e utiliza seus modelos como representação na memória de qualquer arquivo de texto analisado.

4. Execução do Estudo

O objeto de estudo é uma versão preliminar da SaSML, uma extensão UML que mapeia as abstrações e restrições de nível mais alto de sistemas autoadaptativos ao introduz um novo elemento chamado AdaptiveBehavior para fornecer suporte adequado à modelagem conceitual desses sistemas[2]. Para desenvolver esta DSL, primeiro, temos que modelar o conceito, os relacionamentos e as restrições do domínio. Depois, temos que identificar os elementos UML com a semântica adequada para as nossas necessidades, implementar as extensões e verificar a integridade do modelo.

4.1. Metamodelo em EMF

Um metamodelo Ecore comporta a criação de DSLs em EMF, Xtext e Eclipse Sirius. Portanto, realizamos a definição do metamodelo da linguagem exibido na Figura 1(B)) a partir da identificação dos elementos que compõe o AdaptiveBehavior. A seguir os componentes do metamodelo são detalhados.

Como componente raiz denominou-se o SaSProject. Um SaSProject é composto de zero ou muitos SaSManager e zero ou muitas SaSEntity. Um SaSManager deve levar o nome do Identificador do Elemento e é composto de um Contexts e um Behaviors, compartimentos demarcados na parte superior com um triangulo preto contendo as letras C e B, respectivamente, na Figura 1(A).

Através do Contexts obtém-se acesso às informações de contexto, ele é composto pelos SaSContext e deve ser associado a SaSEntity respectiva. Os comportamentos que o sistema pode adotar em tempo de execução são acessados a partir de Behaviors, que está associado a zero ou muitos SaSBehaviors.

A partir do metamodelo gerou-se três plug-ins para o Eclipse, discutidos logo mais. A apresentação da versão em árvore do Eclipse EMF foi omitida por motivos de espaço. Ela é parte necessária para a geração do *plugin* em Eclipse Sirius.

4.2. Versão SaSML em Modo Textual com Xtext

A versão desenvolvida em Xtext possui elementos compostos por uma descrição e um identificador - nome. Onde são determinadas regras, o início de um projeto deve possuir o estereótipo SaSProject em seguida deve haver uma cadeia identificadora e um símbolo "{" . Em sequência, podem ser inseridos SaSEntity ou SaSManager.

A regra para o elemento SaSEntity é apenas um identificador seguido dos símbolos "{}", que posteriormente pode evoluir e possuir atributos. Já o SaSManager possui as mesmas regras que o anterior, além dos atributos SaSConext e SaSBehavior. O SaSContext possui um nome um simbolo " : " e uma referencia aos elementos SaSEntity já criados, caso esteja vazio (ainda não descritos) será apresentado erro sintático. O SaSBehavior possui um identificador e uma opção de inserir ou não o

[2]Para fins de simplificação do estudo a especificação restringiu-se ao novo elemento

tipo de retorno (caso não seja informado não mostra erro ou inconsistência). Ao fim é necessário informar o(s) "}".

Os elementos podem ser repetidos, apenas não é o caso do SaSProject. Logo, para que fosse possível inserir um novo elemento sem sobrescrever o anterior, foi necessário informar a estrela de uma linguagem, ou seja, inserir o símbolo " * " no bloco correspondente as repetições. A Figura 2(B) representa o elemento ControladorAnticolisao no modo textual no *framework* Xtext. Essa linguagem é baseada em conceitos de compiladores, como podem ser vistos na Figura 2(A) como uma gramática elaborada com o Xtext que define a sintaxe da DSL.

A) Gramática da linguagem no Xtext, gerada pela transformação do modelo Ecore

```
grammar org.edm.projmde.ProjMde with org.eclipse.xtext.common.Terminals

generate projMde "http://www.edm.org/projmde/ProjMde"

Model:
    {Model} ('SaSProject' nomeProjeto=ID '{' ( entidades+=Entity)*
                    ('SaSManeger' (nameElemento=ID) '{'
                                (elements+=Manager)*
                    '}')*'}'    )
;

Manager:
    AbstractContext | AbstractBehavior ;

Entity:
     'SaSEntity' name=ID  '{}';

AbstractContext:
    'SaSContext' nomeContext=ID ':' ref=[Entity];

AbstractBehavior:
    'SaSBehavior' nomeBehavior=ID (retornoBehavior=ID)?;
```

B) Representação textual de um modelo conforme a DSL SaSML, construída em Eclipse Xtext

```
SaSProject SaSML{

        SaSEntity Veiculo {}
        SaSEntity  Objeto {}

        SaSManeger ControladorAntiColisao {
            SaSContext Velocidade : Veiculo
            SaSContext Distancia : Objeto
            SaSBehavior Desaceleracao void
            SaSBehavior Frenagem void

        }

}
```

Figura 2. Gramática da linguagem em Xtext, gerada na transformação do Ecore

4.3. Versão SaSML em Modo Gráfico com Eclipse Sirius

No eclipse Sirius, as notações gráficas são definidas em projeto de especificação de ponto de vista. Portanto, o primeiro passo foi cria-lo. A partir do modelo especificação de ponto de vista *(Viewpoint Specification Model)*, temos acesso ao editor em árvore Sirius, ele permite determinar a respectiva representação de cada elemento Ecore. Fornecendo portanto, todas as ferramenta necessárias para definir cada elemento que compõe o modelo.

Os componentes que compõe a DSL são definidos de acordo com a especificação do metamodelo conforme explicado a seguir: 1) Cria-se o elemento gráfico (contêiner, nó ou aresta); 2) Determina-se sempre a qual classe de domínio ele corresponde e quais os candidatos semânticos irão ser representados naquele elemento especifico; 3) Define-se de que modo ele será exibido, para tal, isso são fornecidas diversas personalizações de acordo com o elemento selecionado; 4) Opcionalmente, define-se customizações e regras de validação adicionais.

Esse processo, aplicado a SaSML, teve como resultado a Figura 3(A), que é aprofundado a seguir. Na tentativa de uma representação mais fiel à notação gráfica, optou-se pela representação em diagrama. Ele utiliza o metamodelo o Ecore SaSML discutido anteriormente tem como classe de domínio a raiz SaSProject.

Previsivelmente, o elemento AdaptiveBehavior foi definido como um contêiner que é composto por seus três compartimentos de identificação, de controle e

comportamental, que também são contêineres e são detalhados hierarquicamente a seguir.

No topo, encontra-se o compartimento identificador que possui seu estilo característico, a figura de trapézio, para facilitar o alinhamento do rótulo do SaS-Manager, um nó foi criado para exibi-lo.

Na parte central, está o compartimento contextual. Como pode ser visto no elemento proposto, ele é composto de duas colunas verticais, a esquerda estão os contextos (SaSContext) propriamente ditos, e a direita as entidades (SaSEntity) que estão sendo mapeadas. Para que a representação aconteça de maneira correta, foram criados dois contêineres de alinhamento vertical, subordinados ao contêiner de alinhamento horizontal de contexto que mapeia a classe Contexts. Embora uma entidade possa estar relacionadas a mais de um contexto, pela notação ela deverá ser representada apenas uma vez e terá múltiplas arestas ligando os contextos a ela. Para evitar essa duplicação, para SaSEntitys de eleição de candidatos semânticos foi definido um método, na classe de serviços *(Services)*, que recebe como parâmetro o Contexts, e por meio dele acessa os seus SaSContexts buscando suas respectivas SaSEntity e os adicionada em um conjunto (Set) e por fim o retorna, portanto, sem repetições.

Na parte inferior, o compartimento de comportamentos foi definido, ele uma coluna com todos os comportamentos alternativos. Seu estilo foi definido utilizando uma figura de retângulo, que contem a marcação na ponta superior (um triangulo preto com a letra B), processo semelhante foi realizado no compartimento anterior, porém, com sua letra respectiva, a C.

Finalizando, foi criada uma aresta *(Edge)* que representa o relacionamento entre SaSContext e seu respectivo SaSEntity, mapeando os como fonte *(Source)* e alvo *(Target)* respectivamente. Por não ser o objetivo do estudo a criação de um editor SaSML, não foram criadas ferramentas, filtros ou outras camadas de visualização para manipular esses elementos.

Um exemplo da representação gráfica é exibido na figura 3(B). Ela exibe um fragmento de um modelo de carro autoadaptativo, responsável por realizar seu controle anticolisão.

5. Mapeamento e Transformações de Modelos

Uma vez que as DSLs foram desenvolvidas, modelos de cada versão foram criados e deu-se seguimento ao processo de desenvolvimento para testar as mesmas. Nesta seção são discutidas as transformações que assistem o processo, permitindo a manipulação do modelo SaSML de EMF (XText, Eclipse Sirius) em nível independente de plataforma para a plataforma alvo.

5.1. Mapeamento da SaSML para Plataforma Alvo

O modelo SaSML proposto em nível independente de plataforma, apresentado na Figura 3 pode ser mapeado para plataformas específicas, como linguagens e frameworks de desenvolvimento de sistemas auto-adaptativos. A plataforma alvo testada corresponde ao conjunto de características presentes no domínio expresso na Figura 4, um

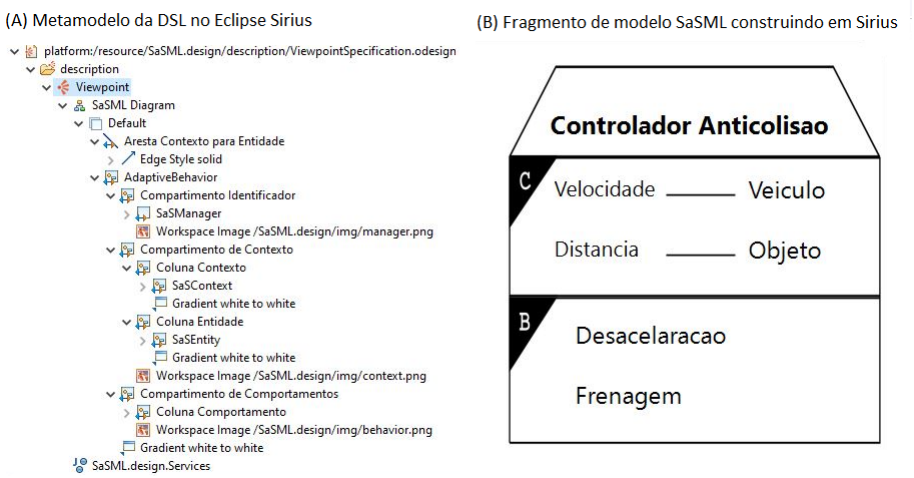

Figura 3. Construção da DSL em Eclipse Sirius

Framework contendo as associações entre as classes AbstractEntity, AbstractBehavior, AbstractManager, AbstractContext. Cada componente do modelo foi mapeado com a devida herança para a super classe respectiva.

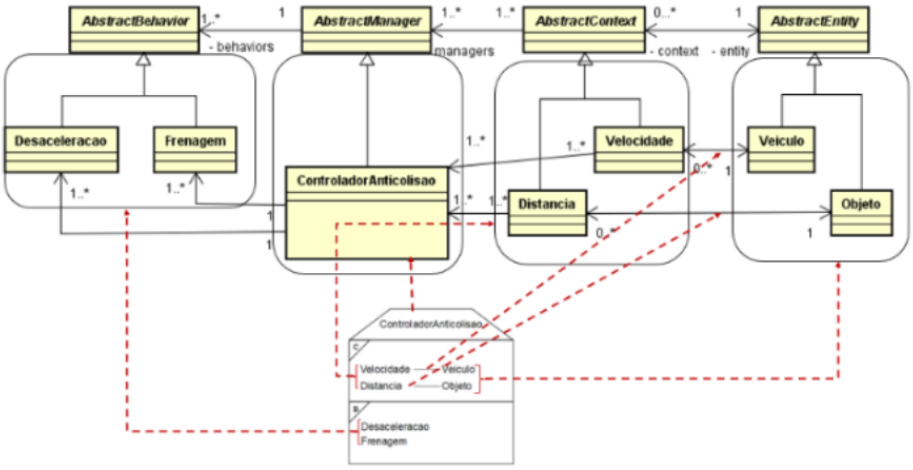

Figura 4. Mapeamento de elemento do modelo "ControladorAnticolisão"para classes específicas de um framework

5.2. Transformação para a Plataforma Alvo

A transformação permite migrar de um modelo a outro a partir de uma estrutura lógica programada de recuperação inteligente. Com auxílio da API WCT, realizamos uma prova de conceito, transformando o modelo gerado para a Plataforma Alvo. Onde identificamos as classes envolvidas (AbstractEntity, AbstractManager, AbstractContext e AbstractBehavior).

A transformação foi realizada por meio da API WCT [Basso et al. 2014]. A WCT possui um aparato robusto para manipulação modelos, permitiu realizar as associações e heranças necessárias entre as classes do modelo.

Inicialmente importou-se o modelo em SaSML, bem como o arquivo com as classes da plataforma. Uma vez que o elemento raiz do metamodelo da SaSML é o

SaSProject, usamos ele para ter acesso aos demais elementos. Para todo SaSEntity presente no SaSProject, foi criado uma classe com seu respectivo nome herdado de AbstractEntity. Para todo SaSManager, presente no SaSProject, foi criado uma classe com seu respectivo nome e realizada a herança com a classe AbstractManager. Por meio de cada AbstractManager, acessou-se Contexts e Behaviors contido nele.

Utilizou-se o Contexts para acessar os SaSContext. Para todo SaSContext, foi criado uma classe com seu respectivo nome que herda de AbstractContext e é associado a sua respectiva SaSEntity e ao SaSManager no qual ele se encontrava. Utilizamos o Behaviors para acessar os SaSBehaviors. Para cada SaSBehavior, foi criado uma classe com seu respectivo nome que herda de AbstractBehavior e também é associado com o SaSManager no qual ele se encontrava. A prova de conceito realizada foi um sucesso, a API Fomda [Basso et al. 2014] gerou o arquivo de saída XMI contendo todas as associações representando o modelo gerado conforme a UML. O fragmento do transformador responsável pela de transformação é exibido em Figura 5(A) Utilizamos a ferramenta WCT para visualização do arquivo e obtivemos o resultado apresentado abaixo exibido na Figura 5(B).

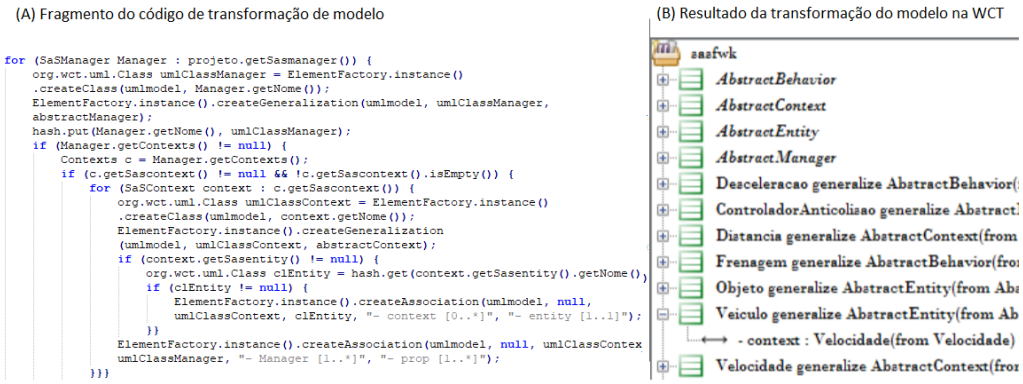

Figura 5. Elementos finais de um processo de transformação de modelos

6. Trabalhos Relacionados

Em [Lima et al. 2019] os autores comparam características de dois construtores de DSLs textuais: o Eclipse *workbench* para XText e *workbench* MPS. Este é um estudo analítico que descreve os pontos positivos e negativos de cada abordagem. A conclusão é que o MPS apresenta características mais interessantes para a construção de DSLs textuais. Em [Favero et al.] os autores apresentam um estudo exploratório para identificar os pontos positivos e negativos do Eclipse *workbench* disponível para a construção de DSLs em ambientes web. O estudo inclui as tecnologias EMF, EMF Forms e Angular, concluindo que apesar de o *workbench* possuir mecanismos de transformações entre estas tecnologias, o refinamento para uma DSL em estágio final necessita de muitos ajustes manuais.

Nosso estudo é complementar e investiga um aparato de tecnologias diferente, dentro do ecossistema do Eclipse para a construção de DSLs. Observou-se que este ecossistema é integrado e permite migrar de uma representação em árvore (EMF) para representação textual e gráfica por meio de transformações. No entanto,

este não é um procedimento linear, pois os artefatos gerados sempre necessitam de refinamento.

7. Conclusão

Uma vez que cada domínio de aplicação pode ter maior aceitação por determinados tipos de DSLs, este estudo exploratório apresenta um relato de experiência no desenvolvimento de três versões de um DSL chamada SaSML. Apoiados no metamodelo em EMF, especificou-se versões em modo árvore (o padrão do EMF), modo textual (Xtext) e em modo gráfico (Eclipse Sirius). Através destas representações, também realizou-se o mapeamento para a plataforma alvo, o que simplificou a compreensão e o processo de transformação entre os modelos.

Sobre a cadeia de ferramentas utilizadas para o desenvolvimento da prova de conceito, destacam-se as seguintes observações: 1) A API WCT permitiu agrupar as informações das versões geradas e transformar o modelo para a plataforma alvo, sendo uma boa opção para o desenvolvimento das provas de conceito futuras; 2) Os *plugins* existentes dentro do Eclipse Modelling Tools (EMT) são integrados, possuem recursos integrados que viabilizam uma simples recuperação e manipulação das informações do metamodelo, e de forma eficiente migram informações do metamodelo para configurações para protótipos executáveis para EMF em Árvore, Eclipse Sirius e XText; e 3) O conjunto ferramental adotado é possível de assimilação por estudantes de graduação sem experiência em MDE, apesar de que observou-se bastante dificuldades na etapa de análise de domínio para extrair as informações gráficas da DSL em metaclasses do metamodelo EMF.

Não foram realizadas avaliações empíricas por meio de grupo focal nas DSLs implementadas. Ficam como pontos em aberto, portanto, os dois outros desafios motivados: 1) uma vez que DSLs em nossos grupos de estudo estão em seu estágio embrionário e não se tem dados empíricos sobre qual tecnologia é viável para construí-las, pretende-se seguir no estudo de outras alternativas como UML Profiles [Schmidt 2006], METACASE [Kelly and Tolvanen 2008] e outras alternativas híbridas [Addazi et al. 2017]; e 2) pretende-se avaliar, com grupo focais de alunos de Engenharia de Software e de Ciência da Computação, qual forma de representação (árvore, gráfica ou textual) é mais aceita para representação de diversos domínios de aplicações investigados em nosso grupo de pesquisa, como: de sistemas de automação e controle (via a DSL M4PIA), sistemas de informações (via a DSL MockupToME) e sistemas de IoT (via a DSL MARTE Profile).

8. Agradecimentos

Este estudo foi parcialmente financiado através da Pró-Reitoria de Pesquisa (PROPESQ), com bolsa do programa AGP (Apoio a Grupos de Pesquisa), e pela FAPERGS, por meio do projeto ARD N. 19/2551-0001268-3.

Referências

Addazi, L., Ciccozzi, F., Langer, P., and Posse, E. (2017). Towards seamless hybrid graphical–textual modelling for uml and profiles. In *Modelling Foundations and Applications*, pages 20–33. Springer International Publishing.

Basso, F. P., Pillat, R. M., Oliveira, T. C., and Fabro, M. D. D. (2014). Generative adaptation of model transformation assets: Experiences, lessons and drawbacks. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1027–1034.

Bettini, L. (2016). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.

da Silva, J. P. S. (2018). Sasml : a uml based domain specific modeling language for self adaptive systems conceptual modeling.

Eclipse Sirius (2018). Disponível em : <https://www.eclipse.org/sirius/doc/> acesso em junho de 2018.

Favero, E. S., de Oliveira, I. A., nez, P. S. Z. N., and Basso, F. P. Estudo exploratório no refinamento de uma dsl para versões baseadas em emf, emf forms e angular.

Kelly, S. and Tolvanen, J.-P. (2008). *Domain Specific Modeling: Enabling Full Code Generation*. IEEE Computer Society - John Wiley & Sons.

Lima, Y. A., Modesto, S., Medeiros, J. M., Carbonell, J. B. P., and de Macedo Rodrigues, E. (2019). Mps x xtext: Uma comparação de languages workbenches para o desenvolvimento de dsls. In *Anais da III Escola Regional de Engenharia de Software*, pages 97–104, Porto Alegre, RS, Brasil. SBC.

Macás-Escrivá, F. D., Haber, R., del Toro, R., and Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267 – 7279.

Neto, A., Carbonell, J., Marchezan, L., Rodrigues, E. M., Bernardino, M., Basso, F. P., and Medeiros, B. (2020). Systematic mapping study on domain-specific language development tools. *Empir. Softw. Eng.*, 25(5):4205–4249.

Neto, V. V. G., Basso, F. P., dos Santos, R. P., Bakar, N. H., Kassab, M., Werner, C., de Oliveira, T. C., and Nakagawa, E. Y. (2019). Model-driven engineering ecosystems. In *Proceedings of the 7th International Workshop on Software Engineering for Systems-of-Systems and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems, SESoS-WDES 2019, Montreal, QC, Canada, May 28, 2019*, pages 58–61. IEEE / ACM.

Petricic, A., Crnkovic, I., and Zagar, M. (2008). Models transformation between uml and a domain specific language. In *Eight Conference on Software Engineering Research and Practice in Sweden (SERPS 08)*.

Schmidt, D. C. (2006). Model-driven engineering. *IEEE Computer*, 39(2).

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional.

Voelter, M. (2009). Best practices for dsls and model-driven development. *Journal of Object Technology*, 8(6):79–102.

XText (2018). Disponível em : <http://www.eclipse.org/xtext/documentation/> acesso em junho de 2018.