MPS x Xtext: Uma comparação de Languages Workbenches para o desenvolvimento de DSLs

Yury Alencar Lima[1], **Samuel Modesto**[1], **Juliana Mareco Medeiros**[1], **João Batista Pedroso Carbonell** [1], **Elder de Macedo Rodrigues**[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

{yuryalencar19,samuelmodestoes,julianamareco18,

joaocarbonellpc, eldermr}@gmail.com

Abstract. *Due to difficulties related to complexity and communication between members of diverse areas within a team, Domain-Specific Languages (DSL) are developed that can minimize the complexity of some activities. There are several notations and ways to create DSLs, one of them is using a Language Workbench (LW), which develops various types of language. As is the case with textual language which is considered classic and most common. However, choosing LW to make DSL more easily accepted by the customer becomes a problem in the development scenario. Based on this, this study compares two LWs used in the implementation of textual DSLs, named MPS and Xtext.*

Resumo. *Devido às dificuldades relacionadas à complexidade e comunicação entre membros de áreas diversas dentro de uma equipe, são desenvolvidas Linguagens de Domínio Específico (DSL) que podem minimizar a complexidade de algumas atividades. Existem diversas notações e formas de criar DSLs, uma delas é utilizando uma Language Workbench (LW), que desenvolve vários tipos de linguagem. Como é o caso da linguagem textual que é considerada clássica e mais comum. No entanto, a escolha da LW para que a DSL seja aceita com maior facilidade pelo o cliente se torna um problema no cenário de desenvolvimento. Com base nisto, este estudo realiza uma comparação entre duas LWs usadas na implementação de DSLs textuais, sendo elas a MPS e a Xtext.*

1. Introdução

O desenvolvimento de software é uma tarefa não trivial que depende do conhecimento de especialistas de um respectivo domínio [JetBrains 2019a]. A participação destes profissionais pode aumentar a qualidade do serviço ou produto entregue, além de reduzir gastos decorrente a diferentes interpretações dos requisitos especificados [JetBrains 2019a]. Para engajar este especialista ao desenvolvimento, é necessária a padronização da comunicação da equipe com este profissional, utilizando um vocabulário e regras do seu domínio específico [JetBrains 2019a]. O que acaba se tornando uma atividade não trivial por parte dos envolvidos, já que possuem especializações distintas. Com o objetivo de proporcionar uma maior produtividade e melhorar a comunicação com os envolvidos no âmbito profissional podem ser utilizadas Linguagens Específicas de Domínio (DSL) [Fowler 2010].

As DSLs são linguagens que tem como objetivo representar um domínio específico, e muitas vezes ser utilizadas para intermediar a equipe de desenvolvimento dos

especialistas de domínio. Estas linguagens podem fazer o uso da geração de artefatos para facilitar a comunicação como: relatórios, código, modelos gráficos, desde que sejam implementadas para esta finalidade [Fowler 2010]. Existem diversos tipos de DSLs, entretanto as textuais por sua vez são mais eficientes em alguns cenários como em testes de software [Torsel 2011]. Mas a implementação de uma linguagem não é uma tarefa trivial, e com esta finalidade existem as *Languages Workbenches* (LW).

As LW tem o objetivo de facilitar e aumentar a produtividade no desenvolvimento de linguagens específicas de domínio [Fowler 2005]. Com a diversidade de LW existentes, como escolher uma específica acaba sendo um processo difícil que delimita tempo. Tendo em vista que a definição de um LW é um ponto essencial no desenvolvimento de qualquer DSL, este estudo realiza uma comparação entre as LWs Xtext e MPS. Esta escolha é decorrente da possibilidade de criação de DSLs Textuais e Projecionais com base em texto através dos dois LW. Além disto, são *Open Source* e possuem uma documentação completa atualmente.

O presente trabalho foi estruturado conforme segue. A Seção 2 apresenta o referencial teórico utilizado para a criação de uma estrutura de comparação entre os LWs. A Seção 3 apresenta os trabalhos relacionados ao estudo. A Seção 4 apresenta a estrutura de comparação e quais atributos serão comparados de cada LWs. A Seção 5 apresenta os resultados da comparação entre a Xtext e o MPS. A Seção 6 apresenta as considerações finais e trabalhos futuros.

2. Estado da Arte

Com o objetivo de realizar a comparação entre as *Languages Workbenches*, é necessária a compreensão de alguns conceitos do domínio. Com base nisto, esta seção apresenta as definições essenciais para o estudo como: Linguagens de Domínio Específico e as LWs Xtext e MPS.

2.1. Linguagem de Domínio Específico

Diferente das Linguagens de Propósito Gerais (GPL), as Linguagens Específica de Domínio (DSL) são linguagens de computador que possuem o objetivo de fornecer suporte à um determinado domínio [Mernik et al. 2005]. Muitas vezes as diferenças de uma GPL e uma DSL não são nítidas, pois a linguagem pode ter apenas alguns recursos específicos para o domínio, podendo ser amplamente utilizada [Fowler 2010]. Como uma Linguagem Específica de Domínio é destinada especificamente à resolver um problema de um respectivo domínio, ela geralmente não é capaz de resolver os problemas fora dela, sendo parte da Engenharia de Domínio [Fowler 2010]. Com base nisto, o seu nível de abstração é maior do que uma GPL, facilitando o uso por parte dos especialistas do domínio, e os aproximando da resolução dos problemas [Živanov et al. 2008].

Mesmo que uma DSL possa aumentar a produtividade e engajamento dos especialistas de domínio, sua implementação não é uma atividade trivial. Além disto, os especialistas de domínio que irão utilizar a linguagem pode influenciar na sua implementação. Com a finalidade de implementar uma DSL podem ser utilizados os seguintes tipos de Linguagens: 1. **Linguagem Interna**: Ou linguagem embarcada é uma linguagem implementada dentro de outra já existente [Fowler 2006]. Apresenta como principal benefício o conhecimento prévio dos envolvidos na linguagem na qual a DSL foi implementada. Um

exemplo seria uma linguagem para testes unitários desenvolvida em JAVA, onde o programador JAVA tende a ter mais facilidade de adesão e compreensão da DSL. 2. **Linguagem Externa**: É independente de outra linguagem, possui seu próprio analisador e gerador, quando possuir [Fowler 2010]. Com base nisto, o principal benefício é que o desenvolvedor pode criar uma sintaxe livre e especializada para um respectivo domínio, aumentando o nível de abstração para o determinado contexto, quando comparada à uma GPL. 3. ***Language Workbench***: É uma ferramenta que apoia o desenvolvimento de uma linguagem, fornecendo um ambiente de trabalho que possui um esquema para a definição de um modelo semântico, um ambiente para a edição da DSL e realização de uma semântica comportamental por meio de interpretação e geração de código, o que facilita e reduz o custo de desenvolvimento de uma DSL [Fowler 2005].

Além dos tipos de desenvolvimento as DSLs também podem ser classificadas com base no seu *design*. Isto pode fornecer uma maior compreensão, produtividade e inclusão do especialista de domínio dentro da resolução do problema. A seguir são apresentados os tipos de DSL: 1. **Textual**: Este tipo é considerado uma linguagem clássica, onde toda a semântica e sintaxe são expressadas no formato textual. Geralmente são as mais utilizadas e possui como principal vantagem a facilidade de suporte e criação para diversos contextos. Entretanto, um editor customizado é recomendado para aumentar sua produtividade [Fowler 2010]. 2. **Gráfica**: Neste tipo de DSL toda a semântica e sintaxe são expressas por meio de elementos gráficos, o que geralmente pode facilitar o entendimento, mas são menos flexíveis do que as textuais o que pode dificultar a manutenção da linguagem [Fowler 2010]. 3. ***Projectional Editing***: Este tipo de DSL possui recursos extremamente poderosos, entretanto são mais desconhecidas. Estas linguagens são compostas de um editor que apresenta uma projeção aos usuários com base na sintaxe e semântica previamente definidas na forma de um modelo. Esta projeção pode ser tanto gráfica quanto textual. Com base nisto, o especialista do domínio interage com a projeção e o editor traduz essas interações para o formato do modelo especificado, aumentando assim sua flexibilidade em relação às linguagens gráficas [Fowler 2008].

2.2. Xtext

O Xtext é um *framework Open Source* para o desenvolvimento de DSLs textuais [Behrens et al. 2008]. Em oposição aos geradores de analisador padrão, ele gera um modelo de classe para uma árvore de sintaxe abstrata e, também, possui integração com o Eclipse IDE [Behrens et al. 2008]. Com o intuito de criar uma linguagem, o desenvolvedor precisa definir uma gramática dentro do Xtext, dessa forma, o gerador de código escrito baseado em *Parser* utiliza o ANTLR internamente e gera as classes para o modelo de domínio [Behrens et al. 2008]. A ideia principal é descrever a sintaxe da linguagem de forma concreta, ou seja, da mesma forma que ela é mapeada para uma representação na memória, isto significa representar o modelo semântico da sintaxe. Além disso, o Xtext traz facilidade na criação de linguagens específicas de domínio para a *Java Virtual Machine* (JVM) com uma integração sólida no *Java Development Toolkit* do Eclipse, o que proporciona a integração com outros idiomas. Outro aspecto positivo do *framework* é a disponibilização de um gerador de código, que pode ser escrito em utilizando o Xtend, que é uma linguagem de programação estaticamente tipada, que produz código fonte na linguagem JAVA [Behrens et al. 2008].

2.3. MPS

O *Meta Programming System* (MPS) é uma ferramenta *Open Source* para projetar DSLs, sendo considerada a mais consolidada atualmente e que faz uso do recurso *Projectional Editing* [Fowler 2010]. Este recurso permite que os desenvolvedores criem seus próprios editores para projetar sua linguagem especificada e proporciona que a projeção seja textual e gráfica ao mesmo tempo se necessário [JetBrains 2019b]. A ferramenta é desenvolvida pela JetBrains e inclui uma linguagem universal pronta para usar, chamada BaseLanguage, que é uma reprodução da linguagem Java. Com o tempo, houve o acréscimo de diferentes linguagens, como XML, C e JavaScript. Para definir uma DSL é preciso seguir alguns passos como estabelecer os conceitos de linguagem, determinar o editor para tais conceitos e por último estipular um gerador [JetBrains 2019b]. O MPS é baseado em árvore de sintaxe abstrata, ou seja, cada atributo deve ser escolhido em um menu suspenso que aparece para o usuário, dessa forma o elemento selecionado passa a ser projetado [JetBrains 2019b]. O MPS ainda disponibiliza um conjunto de personalizações para o editor no qual é possível simular o comportamento de IDEs convencionais [JetBrains 2019b].

3. Trabalhos Relacionados

Dentre os estudos relacionados foi encontrada uma comparação realizada com acadêmicos e profissionais da indústria, com o intuito de analisar qualitativamente e quantitativamente abordagens que utilizam LW [Erdweg et al. 2015]. Para realizar a comparação foi extraída e analisada a notação, semântica, validação, teste e tipo de edição, e foram atribuídas pontuações entre 0 e 1 para as diversas DSL's comparadas [Erdweg et al. 2015]. Outro estudo relacionado é o de Kosar [Kosar et al. 2010] que realiza uma comparação entre DSLs e bibliotecas de aplicativos através de um experimento. O experimento contou com 36 programadores que responderam um questionário contendo mais de 100 páginas referente às duas abordagens. Além disso foi utilizado o mesmo domínio de aplicação para as duas abordagens, onde as linguagens usadas foram XAML para DSL e C# *Forms* para biblioteca de aplicativos[Kosar et al. 2010].

Entretanto, em uma revisão sistemática [da Costa Araújo et al. 2016], é possível verificar que as ferramentas de suporte ao planejamento na condução de experimentos e comparações, ainda são limitadas. Isto implica diretamente no desenvolvimento de uma DSL, onde os experimentos e comparações são importantes tanto para os pesquisadores quanto para os projetistas da linguagem, tendo em vista que estas métricas podem apoiar a escolha de uma LW adequada para seu determinado contexto [da Costa Araújo et al. 2016]. Com base nisto, este estudo tem o intuito de fornecer suporte para a decisão na escolha da LW utilizando as funcionalidades atuais da Xtext e do MPS.

4. Comparação dos *Languages Workbenches*

Com o objetivo de classificar os LWs de acordo com seu apoio ao desenvolvimento de uma linguagem, produtividade e qualidade da linguagem implementada. Por limitações de espaço foram selecionados seis critérios considerados mais impactantes para o desenvolvimento, aceitação e adoção de uma linguagem. Estes critérios foram organizados em:
1. **Versionamento de Código**, que impacta na aceitação e adoção da linguagem, tendo em vista que é uma prática utilizada tanto em grandes quanto em pequenos projetos, fornecendo praticidade na colaboração dentro de um mesmo projeto, podendo este ser um

fator impactante no tempo da resolução de um problema no desenvolvimento. 2. **IDE**, que impacta na aceitação e adoção da linguagem, já que pode afetar diretamente na produtividade no tanto no desenvolvimento quanto no uso de uma linguagem, e pode influenciar significativamente no tempo necessário para a conclusão de uma determinada tarefa. 3. **Interoperabilidade com ferramentas, plugins e outras Linguagens**, que impacta no desenvolvimento da linguagem, tendo em vista que existem diversas outras ferramentas, plugins e linguagens que podem facilitar parte da implementação. 4. **Geração de código ou outros artefatos**, que impactar tanto no desenvolvimento quanto na adoção e aceitação da linguagem, tendo em vista que assim pode facilitar a comunicação entre profissionais de diferentes áreas. 5. **Facilidade na Engenharia Reversa para a linguagem** que pode impactar na aceitação e adoção da linguagem, pois em casos que antes da criação da linguagem outra solução fosse aplicada, a engenharia reversa da antiga solução para a nova pode ser determinante para a aceitação, já que pode ser muito trabalhoso transferir todos as informações do modelo antigo para o atual. 6. **Versatilidade na notação** este critério pode impactar na aceitação e adoção da linguagem, devido a muitas vezes aumentar o entendimento do usuário final da DSL, com este objetivo linguagens textuais por sua vez podem ajudar no entendimento com o uso de símbolos específicos e determinados.

Com o intuito de classificar ambas linguagens com base nestes critérios as documentações de cada linguagem foram utilizadas, juntamente com testes realizados em dois computadores com diferentes configurações. Com o objetivo de analisar e comparar as IDEs a ferramenta WinOMeter [Jelinek 2019] foi utilizada com o objetivo de realizar o monitoramento dos computadores durante a tarefa. A tarefa escolhida para esta comparação foi a especificação de cenários de teste usando o Gherkin [Wynne et al. 2017], que é uma linguagem para descrição de cenários de teste, sua escolha foi decorrente a ambas LWs a possuírem implementada com acesso gratuito. Os principais dados extraídos para a análise do esforço foram a quantidade de cliques no teclado realizadas durante a implementação e o tempo gasto para a conclusão da atividade.

Após a implementação dos cenários de teste em ambas linguagens foi realizado seu versionamento de várias versões dos cenários de teste utilizando o GitHub [GitHub 2019]. A escolha desta tecnologia foi devido a ser o meio de versionamento mais comum dentre a comunidade *Open Source*, o que acaba simulando um possível versionamento de uma DSL implementada em qualquer LW. Os principais dados utilizados na análise foi a facilidade de entendimento e comparação das versões por meio da ferramenta GitHub, já que é o meio mais comum de análise de modificações. Um fator importante foi que para a análise das notações e do seu versionamento, que além do Gherkin outra linguagem teve de ser utilizada, sendo esta uma linguagem matemática, com o intuito de verificar como os símbolos seriam versionados, caso eles existam na linguagem. Analisando assim a notação em conjunto do versionamento.

No entanto, para a realização da análise da geração do código, foi necessária a implementação de uma pequena linguagem em cada LW, com o objetivo da geração de um pequeno código em JAVA. A linguagem implementada tem como intuito a descrição de formas geométricas juntamente com seu tamanho e posição no plano cartesiano. O resultado da geração seria um código JAVA, onde na tela ilustrasse os elementos descritos. Durante o desenvolvimento foi analisada a possibilidade de integração com outras ferramentas, *plugins*, bibliotecas ou linguagens para facilitar o desenvolvimento. Já na

comparação para a engenharia reversa foi criado um *Parser* do código JAVA exportado por meio da linguagem para a notação da linguagem e foi medido o esforço para está tarefa.

5. Resultados

Baseado no modelo de comparação descrito na Seção 4, o Eclipse IDE utilizado em conjunto ao Xtext, tanto para desenvolvimento quanto para o uso da Linguagem implementada, apresenta uma vantagem relacionada ao MPS IDE. Esta vantagem é decorrente aos requisitos mínimos serem mais acessíveis a uma maior quantidade de desenvolvedores e não apresentarem restrições de resolução, tais requisitos estão presentes na Tabela 1. Para realizar a comparação foi utilizado o Eclipse, devido a outras IDEs limitarem o uso de algumas funcionalidades do Xtext [Behrens et al. 2008].

Eclipse IDE com o Xtext		
Requisitos do sistema	**Mínimo**	**Recomendado**
Memória RAM	512 MB	1 GB
Espaço Disponível no Disco	300 MB	1 GB
Resolução de Tela	N/A	N/A
MPS IDE		
Requisitos do sistema	**Mínimo**	**Recomendado**
Memória RAM	3 GB	8 GB
Espaço Disponível no Disco	2.5 GB	SSD
Resolução de Tela	1024x768	N/A

Tabela 1. Requisitos de Sistema para o uso da IDE. Criado pelos Autores.

Os resultados da comparação do uso das IDEs são apresentados na Tabela 2 e são uma média aritmética entre dez execuções de dois desenvolvedores utilizando a linguagem Gherkin em cada linguagem LW. O MPS apresentou uma maior produtividade em relação à Xtext, mesmo que os requisitos para o uso do MPS sejam maiores, o que pode ser uma pequena barreira em hardwares inferiores, tanto para o uso da linguagem implementada quanto para o desenvolvimento de uma DSL. Além disso, diante da Tabela pode ser observado que o MPS, com o objetivo de escrever um cenário de teste, necessitou de uma quantidade significativamente menor de tempo e teclas pressionadas para realizar a mesma tarefa do que no Xtext. Tais resultados podem ser explicados devido a MPS fornecer ao usuário toda a estrutura escrita, enquanto no Xtext é necessário que o até mesmo a estrutura seja inserida mesmo que com o uso de atalhos.

	Número de Teclas Pressionadas	**Percentual de utilização do Teclado** (Em relação ao Total de Acionamentos)	**Cliques do Mouse**	**Tempo** (min)
MPS	151	20%	3	03:46
Xtext	604	80%	2	11:02

Tabela 2. Dados extraídos para avaliação da IDE

Entretanto, na comparação relacionada ao versionamento a Xtext apresentou certa superioridade, decorrente dos arquivos do MPS poderem ser analisados somente pela própria LW , devido a ser do tipo *projectional editing*. Com base nisto, o versionamento não pode ser analisado por profissionais sem conhecimento da IDE, já que estas

informações só podem ser visualizadas através da LW. Além disso, os arquivos utilizados tanto no desenvolvimento quanto no uso de uma DSL no Xtext estão no formato textual, facilitando a análise através da ferramenta GitHub e por profissionais sem conhecimentos da IDE.

O MPS também apresentou certa vantagem em relação à geração de código em comparação ao Xtext, tendo em vista que no MPS esta geração pode ser realizada através da própria LW. Enquanto isto, no Xtext é necessário o uso de uma linguagem auxiliar, sendo esta o Xtend [Behrens et al. 2008], o que implica no aprendizado de outra sintaxe com esta finalidade. Entretanto, a Engenharia Reversa do artefato gerado para a linguagem geradora no Xtext, é uma atividade simples já que seus arquivos são baseados em texto. Então basta criar um analisador simples, enquanto no MPS é necessário a utilização de extensões além de possuir entendimento da estrutura dos arquivos de definição da linguagem.

Decorrente ao Xtext ser um *plugin* do ambiente Eclipse, possui uma grande interoperabilidade com outras ferramentas, linguagens de propósito geral e *plugins* do ambiente. Um exemplo é a Sirius um *plugin* para o desenvolvimento de DSLs gráficas [Sirius 2019]. Ao mesmo tempo que o MPS possui uma grande interoperabilidade entre outras linguagens de propósito geral, bibliotecas e DSLs desenvolvidas através do mesmo LW. No contexto de notação a Xtext não possui a possibilidade de utilização de símbolos em sua sintaxe, o que simplifica seu versionamento, mas pode impactar no entendimento em alguns casos, como linguagens matemáticas por exemplo. Entretanto como o MPS possui sua própria estrutura de versionamento e definição da linguagem, a LW proporciona a projeção de símbolos como parte da sintaxe. Na Tabela 3 é possível visualizar os resultados finais sumarizados da comparação apresentada.

LW	Versionamento de Código	IDE	Interoperabilidade	Geração de Código	Engenharia Reversa	Versatilidade
MPS	Difícil	Mais Produtiva	Fácil	Fácil	Difícil	Uso de Símbolos na Linguagem
Xtext	Fácil	Menos Produtiva	Fácil	Depende de outra Ferramenta	Fácil	Somente textos

Tabela 3. Resultados Finais Sumarizados

6. Considerações Finais

Com base nos resultados obtidos cada LW tem suas peculiaridades e benefícios o que as tornam interessantes para cada contexto, no caso de equipes muito grandes e verificação entre as versões independente da IDE utilizada uma DSL implementada em Xtext é a recomendada. Todavia se seu projeto preza produtividade, uso de símbolos juntamente com a notação, menos verificações entre versões ou o responsável pelas verificações conhece a IDE o recomendado é a implementada em MPS. Entretanto o resultado pode ter sido afetado por alguns critérios como a MPS utilizar projeções ao invés de texto padrão e da IDE na qual o Xtext foi utilizado ter sido o Eclipse IDE, mesmo que as outras possibilidades de IDEs para o uso da DSL apresentem limitações elas podem fornecer outras melhorias não detectadas.

Com base nestes problemas detectados, como trabalho futuro foi planejado a realização de um experimento com o objetivo de comparar mais LWs, com diferentes notações em diferentes contextos. Assim, o resultado pode apresentar qual notação ou LW é o recomendado para cada contexto de desenvolvimento.

Referências

Behrens, H., Clay, M., Efftinge, S., Eysholdt, M., Friese, P., Köhnlein, J., Wannheden, K., and Zarnekow, S. (2008). Xtext user guide. *Dostupné z*, page 7.

da Costa Araújo, I., da Silva, W. O., de Sousa Nunes, J. B., and Neto, F. O. (2016). Arrestt: a framework to create reproducible experiments to evaluate software testing techniques. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, page 1. ACM.

Erdweg, S., Van Der Storm, T., Völter, M., Tratt, L., Bosman, R., Cook, W. R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., et al. (2015). Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures*, 44:24–47.

Fowler, M. (2005). Language workbench.

Fowler, M. (2006). Internal dsl style.

Fowler, M. (2008). Projectional editing. *ProjectionalEditing. html*.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

GitHub (2019). Github. https://github.com/.

Jelinek, T. (2019). Winometer 1.30. http://winometer.findmysoft.com/.

JetBrains (2019a). Domain-specific languages. *JetBrains s.r.o.*

JetBrains (2019b). How does mps work?. *JetBrains s.r.o.*

Kosar, T., Oliveira, N., Mernik, M., Pereira, M. J., Crepinsek, M., Cruz, D., and Henriques, P. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *ComSIS–Computer Science an Information Systems Journal*, pages 247–264.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specifi c languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Sirius (2019). Sirius. https://www.eclipse.org/sirius/.

Torsel, A.-M. (2011). Automated test case generation for web applications from a domain specific model. In *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, pages 137–142. IEEE.

Wynne, M., Hellesoy, A., and Tooke, S. (2017). *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf.

Živanov, Ž., Rakić, P., and Hajduković, M. (2008). Using code generation approach in developing kiosk applications. *Computer Science and Information Systems*, 5(1):41–59.