Um Framework para Implementação Declarativa de Sintaxes Concretas Visuais

Félix S. de Souza Neto e Sandro S. Andrade

Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo-Real (GSORT)
Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa)
Av. Araújo Pinho, nº 39 - Canela - Salvador - BA - CEP: 40.110-150

{felixneto, sandroandrade}@ifba.edu.br

Abstract. *The use of models in software development projects is well known for its benefits in improving the process productivity and the quality of generated artifacts. Such software models are usually defined by using some modeling language, which provides the needed constructs and enables the systematic manipulation of models. Visual concrete syntaxes allow for easily creating model elements, configuring their attributes, and defining required elements relationships. This paper presents the design and implementation of a framework for the declarative implementation of visual concrete syntaxes in a metamodel-independent way. The framework has been used to implement the concrete syntax for basic UML constructs in an open-source tool for software modeling.*

Resumo. *A utilização de modelos em projetos de desenvolvimento de software é atualmente reconhecida como fator importante para a melhoria da produtividade do processo e da qualidade dos artefatos gerados. Tais modelos são geralmente descritos em alguma linguagem de modelagem, que disponibiliza os constructos necessários e viabiliza a manipulação sistemática de modelos. Sintaxes concretas visuais permitem que a criação dos elementos do modelo, configuração dos seus atributos e definição de relacionamentos sejam realizadas de uma forma muito mais simples. Este artigo apresenta o projeto e implementação de um framework para definição de sintaxes concretas visuais de forma declarativa e independente do metamodelo em questão. O framework tem sido utilizado para implementar a sintaxe concreta de constructos básicos da UML em uma ferramenta open-source de modelagem.*

1. Introdução

A definição das tecnologias básicas para criação de modelos de *software* e para integração destes modelos em processos de desenvolvimento tem sido o foco de diversas pesquisas na área de *Model-Driven Software Engineering* (MDSE) [Brambilla et al. 2012, Stahl et al. 2006]. Atualmente, modelos de *software* exercem papel fundamental nas atividades de síntese automática de artefatos, testes, evolução e implantação [Hebig et al. 2013]. Modelos têm sido ainda utilizados como infraestrutura para análise de propriedades tais como *dependability*, complexidade, previsibilidade temporal e manutenibilidade [Grassi et al. 2007].

Modelos de *software* são descritos em alguma linguagem de modelagem. Tais linguagens variam desde representações matemáticas formais (tais como Redes de Pe-

tri, Redes de Filas em Camadas e Cadeias de Markov) [Kan 2002], passando por linguagens baseadas em metamodelos (tais como UML e BPML) [Brambilla et al. 2012], até notações mais simples e genéricas, como linguagem natural e gráficos informais [Taylor et al. 2009]. Dentre estas, as linguagens baseadas em metamodelos têm sido amplamente utilizadas como infraestrutura básica para abordagens da MDSE [Stahl et al. 2006].

Uma linguagem de modelagem baseada em metamodelo é constituída por três elementos principais: sintaxe abstrata, sintaxe concreta e semântica [Voelter et al. 2013]. A sintaxe abstrata define as metaclasses que representam os constructos disponibilizados pela linguagem, seus diversos atributos e como estes constructos se relacionam. Instâncias destas metaclasses constituem um modelo em particular, criado em conformidade com o metamodelo em questão. A semântica define os significados estruturais e comportamentais dos constructos e como estes deverão ser reificados em ferramentas de modelagem.

A sintaxe concreta viabiliza a criação de um modelo de uma forma mais simples do que utilizar uma linguagem de programação de propósito geral para instanciar diretamente as metaclasses, ajustar atributos e definir relacionamentos. Ela disponibiliza elementos textuais ou visuais de maior abstração, tornando as operações de modelagem mais simples e produtivas, bem como viabilizando o suporte por ferramentas. Uma infraestrutura subjacente de transformação de modelos realiza a conversão de modelos descritos em sintaxe concreta em representações correspondentes da sintaxe abstrata, e vice-versa.

Embora iniciativas tais como o *Eclipse Modeling Framework* (EMF) [Steinberg et al. 2009], *Graphical Editing Framework* (GEF) [Rubel et al. 2011] e *Xtext/Xtend* [Bettini 2013] facilitem consideravelmente a construção de novas linguagens de modelagem, a implementação de sintaxes concretas e de mapeamentos em elementos da sintaxe abstrata é ainda uma tarefa desafiadora. O esforço de codificação necessário para implementação das representações visuais e definição dos mecanismos de *binding* entre sintaxe concreta e abstrata é geralmente alto. Adicionalmente, a solução deve ser escalável ao manipular grandes modelos.

Este artigo apresenta o projeto e implementação de um *framework* para definição facilitada de sintaxes concretas visuais para linguagens de modelagem. O *framework* é independente do metamodelo utilizado na definição da linguagem e combina uma série de tecnologias que viabilizam: *i)* a utilização de uma linguagem declarativa para definição da sintaxe concreta; e *ii)* renderização suportada por *hardware* das representações visuais utilizadas. Tais características contribuem significativamente para a produtividade e escalabilidade, respectivamente, da solução aqui proposta.

O *framework* proposto foi implementado utilizando a linguagem C++ e o *toolkit* Qt [Qt Community 2014], como parte integrante do DuSE-MT[1] [Andrade and Macêdo 2013] – ferramenta *open-source* para criação e manipulação de modelos de *software*. O *framework* implementa a exportação dos objetos da sintaxe concreta para o interpretador do QML – linguagem declarativa para criação de interfaces gráficas, disponibilizada pelo Qt. Adicionalmente, um mecanismo genérico para mapeamento entre sintaxe abstrata e sintaxe concreta é também disponibilizado pela solução aqui proposta, fazendo com que a implementação de sintaxes concretas se resuma à es-

[1] `http://duse.sf.net`

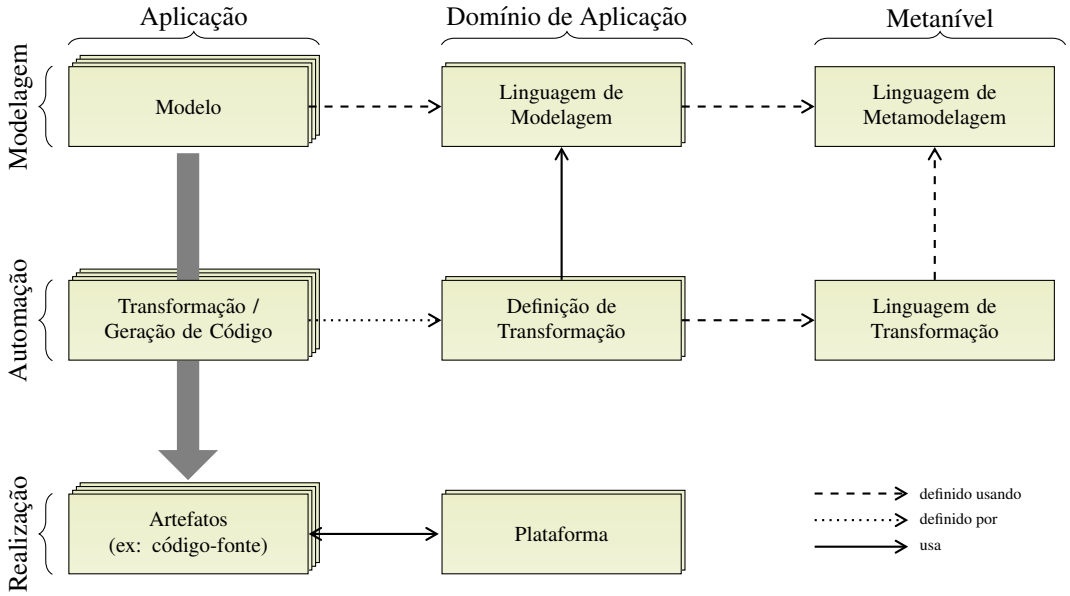

Figura 1. Visão geral das atividades de metamodelagem e transformação de modelos [Brambilla et al. 2012].

crita de código declarativo em conformidade com algumas convenções definidas pelo *framework*.

A solução aqui apresentada tem sido utilizada para implementação de sintaxe concreta para constructos básicos da UML e de linguagens de domínio específico. Experimentos preliminares indicam que a solução viabiliza a implementação produtiva de sintaxes concretas e apresenta boa escalabilidade na visualização de grandes modelos.

O restante deste artigo está organizado como segue. A seção 2 apresenta os fundamentos sobre metamodelagem e sintaxes concretas. A seção 3 apresenta os requisitos, arquitetura e aspectos de implementação do *framework* proposto. A seção 4 apresenta os experimentos preliminares de avaliação da produtividade e escalabilidade da solução proposta. Por fim, a seção 5 discute os trabalhos correlatos e a seção 6 apresenta as conclusões e possibilidades de trabalhos futuros.

2. Metamodelagem e Sintaxes Concretas

Reconhecer os paradigmas e intenções gerais de construção do *software* diretamente a partir do código-fonte é uma atividade extremamente árdua. Modelos de *software* viabilizam a discussão dos artefatos em diferentes níveis de abstração, dependendo dos *stakeholders* envolvidos, estágio atual de desenvolvimento e objetivos da discussão (propósito descritivo). Modelos também podem assumir papéis centrais no desenvolvimento de sistemas computacionais, ao serem considerados não somente como documentação mas também como abstrações (semi-)formais essenciais que guiam a execução das atividades do processo (propósito prescritivo).

Conforme apresentado na Figura 1, a MDSE trabalha com artefatos e operações definidos em duas dimensões ortogonais: implementação (linhas na Figura 1) e conceitualização (colunas na Figura 1). A dimensão de implementação lida com o mapeamento/instanciação de modelos em representações operacionais (ex: artefatos de

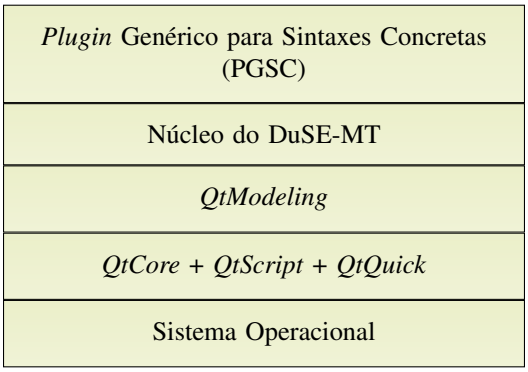

Figura 2. Stack de tecnologias do DuSE-MT.

implementação), enquanto a dimensão de conceitualização busca a definição de modelos conceituais (metamodelos) que representem a realidade sendo descrita.

A dimensão de implementação lida com artefatos em três níveis possíveis: *i)* modelagem, *ii)* realização e *iii)* automação. Artefatos de modelagem são modelos abstratos e compactos que descrevem a estrutura e comportamento do sistema em questão. Artefatos de realização, por sua vez, são as soluções (códigos-fonte, arquivos de dados, etc) que efetivamente implementam o sistema. Os artefatos de automação são responsáveis por executar os mapeamentos entre artefatos de modelagem e artefatos de realização.

A dimensão de conceitualização utiliza artefatos de três categorias: *i)* aplicação, *ii)* domínio de aplicação e *iii)* metanível. Artefatos de aplicação são representações específicas de um sistema pertencente ao domínio em questão, tais como os modelos, execuções de transformações e implementações deste sistema. Artefatos de domínio de aplicação são responsáveis pela especificação de linguagens (metamodelos), transformações e plataformas/ambientes de execução; todos voltados para um determinado domínio de aplicação. Já os artefatos de metanível definem as linguagens utilizadas para a especificação dos artefatos de domínio de aplicação (meta-metamodelos).

As sintaxes concretas oferecem uma forma alternativa de utilização dos metamodelos definidos pela Linguagem de Modelagem apresentada na Figura 1. Sintaxes concretas podem ser especificadas com base em metamodelo ou com base em gramáticas EBNF. Metamodelos padronizados para definição de sintaxe concreta, como por exemplo o *Diagram Definition* (DD) do OMG (*Object Management Group*) [OMG 2014], definem os elementos básicos para especificação de sintaxes concretas visuais. Sintaxes concretas especificadas via gramáticas EBNF geralmente utilizam geradores automáticos de *parsers* para a implementação do interpretador que processa um código descrito na sintaxe concreta em questão. A maior dificuldade, nestes casos, é realizar o mapeamento da árvore sintática abstrata, gerada pelo *parser*, na instância do metamodelo correspondente à sintaxe abstrata.

3. O Framework Proposto

O *framework* proposto neste artigo foi implementado utilizando a linguagem C++ e o *toolkit* Qt, como parte integrante da ferramenta DuSE-MT. Conforme apresentado na Figura 2, o DuSE-MT é uma aplicação *cross-platform* que utiliza os módulos *QtCore*, *QtScript* e *QtQuick* do Qt para implementar os serviços básicos de manipulação de modelos via

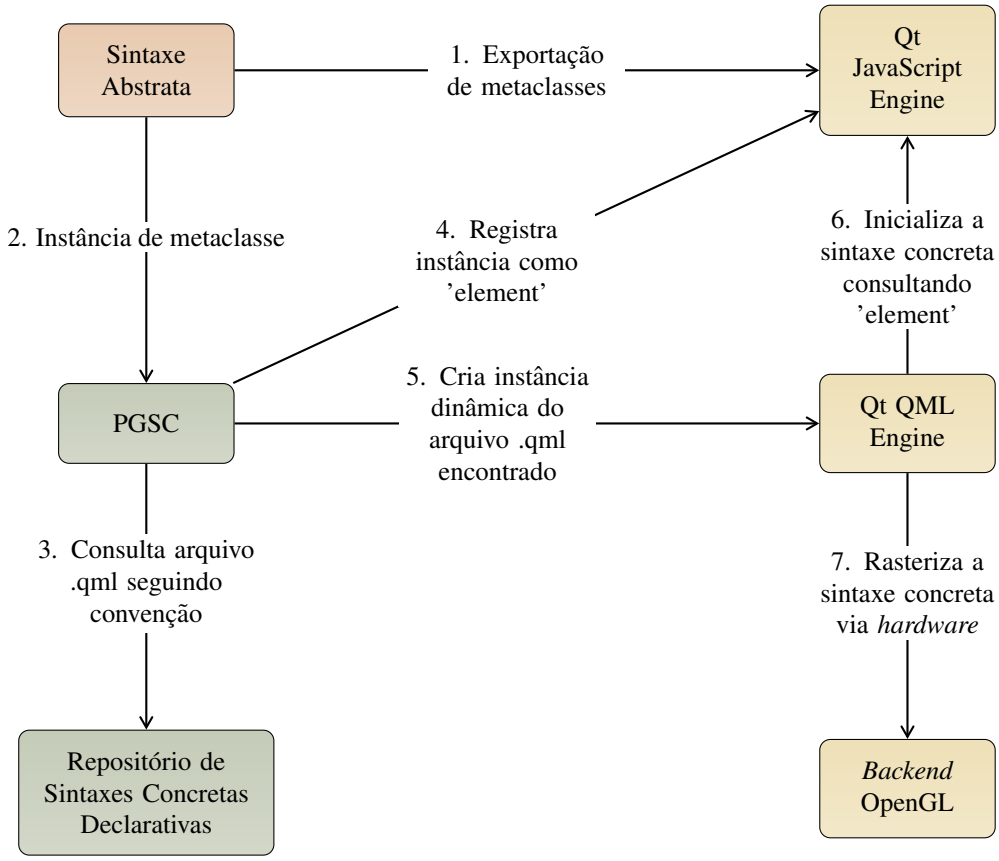

Figura 3. Principais componentes utilizados no *framework* proposto.

scripts e viabilizar a utilização da linguagem declarativa QML para renderização das sintaxes concretas.

Todas as funcionalidades para criação, manipulação e serialização de modelos de *software* estão presentes no módulo *QtModeling*. O núcleo do DuSE-MT define uma infraestrutura que, via mecanismos de reflexão computacional, permite a manipulação de modelos de forma independente do metamodelo utilizado. Uma arquitetura baseada em *plugins* permite a extensão facilitada da ferramenta e foi o mecanismo básico para implementação do *framework* aqui descrito.

Conforme ilustrado na Figura 3, o componente central do *framework* é o Plugin Genérico para Sintaxes Concretas (PGSC). O fluxo básico de renderização de sintaxes concretas é também apresentado na Figura 3. No passo 1, todas as metaclasses do metamodelo em questão são exportadas para a *engine* de *JavaScript* do Qt, o que permite a consulta e manipulação facilitada de modelos. Ao solicitar, no passo 2, a renderização da sintaxe concreta de uma determina instância de metaclasse (por exemplo, `QUmlClass`), o PGSC consulta um repositório de sintaxes concretas declarativas, buscando por um arquivo-fonte cujo nome segue a convenção `<namespace-do-metamodelo><metaclasse-do-metamodelo>.qml` (`UmlClass.qml` para o exemplo acima).

As informações sobre o *namespace* e metaclasse em questão são obtidas via reflexão computacional, de modo a manter a solução independente de metamodelo. Encon-

```
1: Element {
2:     Slot {
3:         id: nameSlot
4:         Text {
5:             id: label
6:             text: element.name
7:         }
8:     }
9:     Slot {
10:        id: attributeSlot
11:        ListView {
12:            model: element.ownedAttributes
13:            delegate: Text {
14:                text: visibility(modelData.visibility) + modelData.name + ": "+
15:                    (modelData.type ? modelData.type.name:"<no type>")
16:            }
17:        }
18:    }
19:    Slot {
20:        ListView {
21:            model: element.ownedOperations
22:            delegate: Text {
23:                text: visibility(modelData.visibility) + modelData.name +
24:                    operationSignature(model)
25:            }
26:        }
27:    }
28: }
```

Figura 4. Implementação declarativa da sintaxe concreta para a metaclasse `Class` do metamodelo UML.

trada a implementação declarativa da sintaxe concreta referente à metaclasse em questão, o PGSC registra, a instância da metaclasse fornecida, no *engine* de *JavaScript* do Qt, com o identificador `element`. Este identificador é utilizado pelas implementações declarativas para obter as informações necessárias à renderização da sintaxe concreta, conforme ilustrado na Figura 4.

As linhas 2, 9 e 19 utilizam o elemento QML *Slot* para declarar os três compartimentos para representar o nome da classe, seus atributos e métodos, respectivamente. Note como, na linha 6, a instância registrada sob o identificador `element` é utilizada para executar o método acessor que disponibiliza o nome da classe representada pela instância. Na linha 12, `element` é novamente utilizado para obter uma lista de instâncias da metaclasse `QUmlProperty`, representando todos os atributos declarados na instância da metaclasse `QUmlClass` sendo renderizada. O elemento QML *ListView* é utilizado para automaticamente percorrer a lista de atributos e renderizar cada um deles utilizando o *delegate* definido nas linhas 13-16. As linhas 19-27 apresentam a implementação declarativa para renderização dos métodos da classe.

A figura 5 apresenta a sintaxe concreta resultante da implementação apresentada

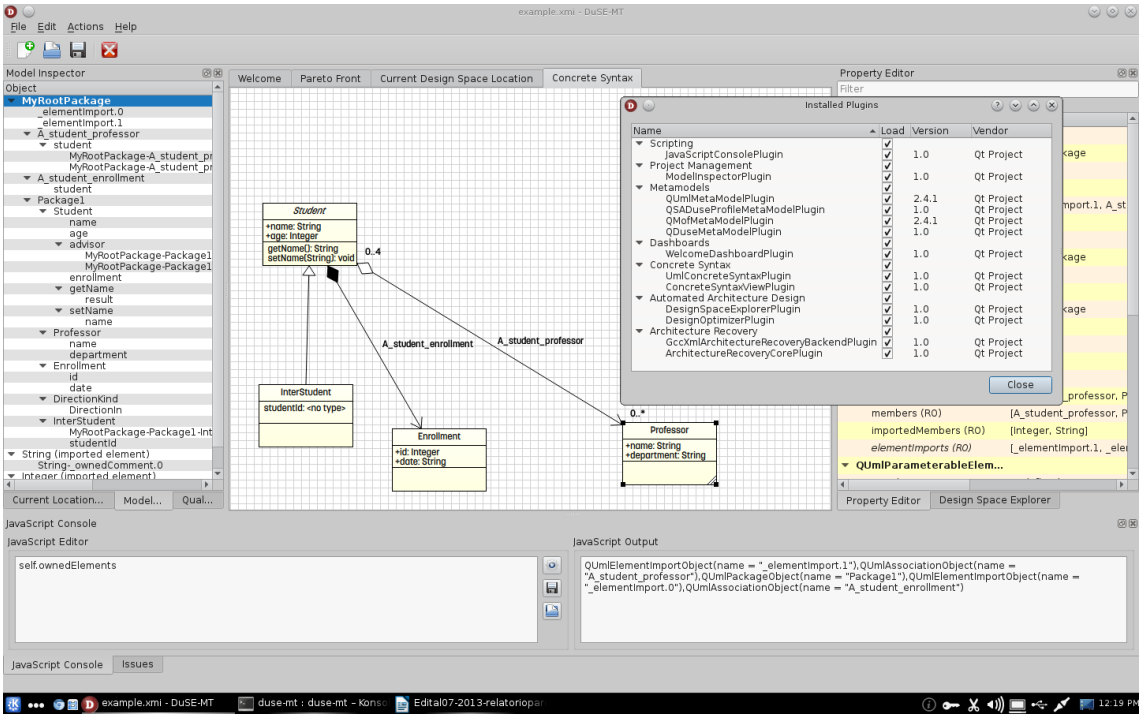

Figura 5. Exemplo ilustrando a sintaxe concreta resultante da implementação declarativa apresentada na Figura 4.

na Figura 4. Vale ressaltar que o *framework* viabiliza a implementação declarativa de sintaxes concretas referentes a outras metaclasses de metamodelos novos ou já existentes simplesmente através da criação de arquivos .qml que seguem a convenção adotada para a nomenclatura do arquivo. Outro aspecto importante é que – em função dos recursos do QML para *binding* de propriedades – modificações realizadas no modelo sendo manipulado (por exemplo, a modificação no nome de um método ou atributo) são automaticamente refletidas na renderização da visualização concreta. Nenhum código adicional é necessário para manter a sincronização entre sintaxe abstrata e sintaxe concreta.

4. Validação

O *framework* proposto neste trabalho é caracterizado pela viabilização da implementação produtiva de sintaxes concretas em função do uso de linguagens declarativas. Adicionalmente, o *backend* OpenGL para renderização de arquivos .qml, disponibilizado pelo Qt, garante excelente desempenho, mesmo na renderização de sintaxes concretas formadas por um número grande de elementos.

Com o objetivo de avaliar a escalabilidade das atividades de renderização de modelos UML pelo *framework* proposto, verificou-se como o tempo de renderização varia à medida em que novas classes são introduzidas no modelo. Conforme observado na Figura 6, embora aparentemente o tempo de renderização inicialmente cresça exponencialmente, este tempo passa a ter um comportamento polinomial para um número alto de classes no modelo. Acredita-se que os mecanismos introduzidos na versão 5.2 do QML tenham trazido benefícios substanciais no desempenho de renderizações.

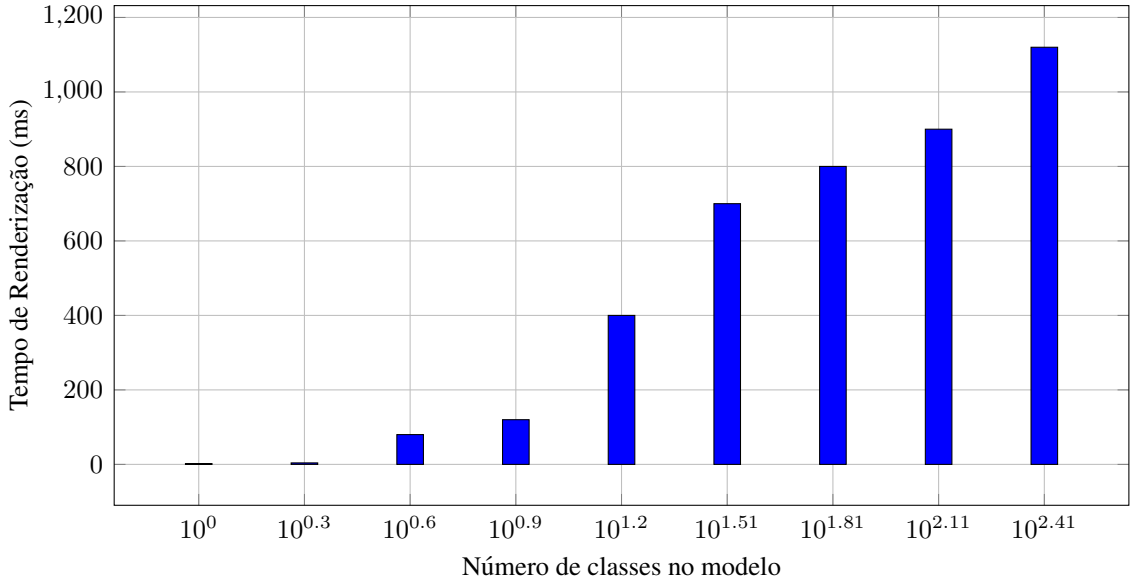

Figura 6. Avaliação do desempenho de renderização do *framework* em função do número de classes no modelo.

5. Trabalhos Correlatos

Uma série de esforços correlatos para implementação de sintaxes concretas podem ser encontrados na literatura. [Ráth et al. 2010] apresentam um modelo de mapeamento entre elementos da sintaxe abstrata e concreta. Adicionalmente, um mecanismo para transformação de modelos permite a sincronização bidirecional e rastreabilidade entre elementos da sintaxe abstrata e concreta.

[Muller et al. 2006] introduzem uma nova forma de especificação de sintaxes concretas textuais através do mapeamento de metamodelos em gramáticas definidas via EBNF. Os autores definem um metamodelo genérico para especificação de sintaxes concretas textuais e regras que mapeiam elementos EBNF em classes deste metamodelo. Um analisador e sintetizador genérico realiza o mapeamento entre o metamodelo da sintaxe concreta e o da sintaxe abstrata.

[Wile 1997] propõe uma solução para o problema oposto ao aqui discutido: derivação de sintaxe abstrata efetiva a partir de especificações escritas em uma sintaxe concreta. Para isso, um mecanismo de mapeamento entre sintaxes concretas definidas em EBNF/YACC e código-fonte C++ descrevendo a sintaxe abstrata é proposto pelo autor.

[Bravenboer and Visser 2004], em sua abordagem denominada MetaBorg, apresentam um método que suporta a definição de sintaxes concretas de forma integrada à linguagem de implementação do metamodelo (sintaxe abstrata). O método integra linguagens de domínio específico em uma linguagem hospedeira de propósito geral. O mapeamento é automaticamente realizado através da análise do código hospedeiro que circunda o código de domínio específico.

[Heidenreich et al. 2009] apresentam um método para derivação automática de sintaxe concreta textual a partir de sintaxes abstratas. Os autores afirmam que os valores *default* para os parâmetros de mapeamento permitem uma geração facilitada de editores textuais já funcionais. Adicionalmente, o método é interessante em casos onde o meta-

modelo muda frequentemente.

[Krahn et al. 2007] apresentam uma abordagem baseada em gramáticas que permite a geração integrada de sintaxes abstratas e concretas, evitando problemas de redundância e necessidade de sincronização.

O *framework* apresentado neste artigo difere dos demais trabalhos no sentido em que baseia-se na utilização de linguagens declarativas e convenções de codificação para aumentar a produtividade da implementação de sintaxes concretas. Adicionalmente, novas sintaxes concretas para metamodelos novos ou já existentes podem ser adicionadas sem a necessidade de nenhuma modificação no núcleo da ferramenta ou na implementação da sintaxe abstrata.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou o projeto e implementação de um *framework* que viabiliza a implementação de sintaxes concretas de forma independente de metamodelo e através da utilização de linguagens declarativas. Foram apresentados os fundamentos sobre as atividades de metamodelagem e definição de sintaxes concretas.

A arquitetura definida para o *framework* permite a extensão facilitada de novas sintaxes concretas sem impactos no núcleo da ferramenta de modelagem. Adicionalmente, o registro e acesso automático às propriedades das metaclasses através de *scripts* alavanca um desenvolvimento mais produtivo de sintaxes concretas.

Como trabalhos futuros pode-se citar o desenvolvimento de soluções para alteração de atributos das instâncias de metaclasses diretamente através da sintaxe concreta, a definição de novos elementos QML para facilitar o reuso de elementos gráficos comuns a várias sintaxes concretas e a definição de mecanismos de conversão entre metamodelos de representação de sintaxes concretas.

Referências

[Andrade and Macêdo 2013] Andrade, S. S. and Macêdo, R. J. d. A. (2013). A search-based approach for architectural design of feedback control concerns in self-adaptive systems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, pages 61–70. IEEE.

[Bettini 2013] Bettini, L. (2013). *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd.

[Brambilla et al. 2012] Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers.

[Bravenboer and Visser 2004] Bravenboer, M. and Visser, E. (2004). Concrete syntax for objects: Domain-specific language embedding and assimilation without restrictions. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA'04, pages 365–383, New York, NY, USA. ACM.

[Grassi et al. 2007] Grassi, V., Mirandola, R., and Sabetta, A. (2007). Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software*, 80(4):528–558.

[Hebig et al. 2013] Hebig, R., Giese, H., Stallmann, F., and Seibel, A. (2013). On the complex nature of MDE evolution. In Moreira, A. and Schaetz, B., editors, *Model Driven Engineering Languages and Systems, 16th International Conference, MODELS 2013*, Miami, USA.

[Heidenreich et al. 2009] Heidenreich, F., Johannes, J., Karol, S., Seifert, M., and Wende, C. (2009). Derivation and refinement of textual syntax for models. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ECMDA-FA'09, pages 114–129, Berlin, Heidelberg. Springer-Verlag.

[Kan 2002] Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

[Krahn et al. 2007] Krahn, H., Rumpe, B., and Völkel, S. (2007). Integrated definition of abstract and concrete syntax for textual languages. In Engels, G., Opdyke, B., Schmidt, D. C., and Weil, F., editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 286–300. Springer.

[Muller et al. 2006] Muller, P.-A., Fleurey, F., Fondement, F., Hassenforder, M., Schneckenburger, R., Gérard, S., and Jézéquel, J.-M. (2006). Model-driven analysis and synthesis of concrete syntax. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems*, MoDELS'06, pages 98–110, Berlin, Heidelberg. Springer-Verlag.

[OMG 2014] OMG (2014). Diagram Definition (DD) specification 1.0. `http://www.omg.org/spec/DD/`. Acesso: 29/04/2014.

[Qt Community 2014] Qt Community (2014). Qt project. `http://qt-project.org/`. Acesso: 29/04/2014.

[Ráth et al. 2010] Ráth, I., Ökrös, A., and Varró, D. (2010). Synchronization of abstract and concrete syntax in domain-specific modeling languages - by mapping models and live transformations. *Software and System Modeling*, 9(4):453–471.

[Rubel et al. 2011] Rubel, D., Wren, J., and Clayberg, E. (2011). *The Eclipse Graphical Editing Framework (GEF)*. Eclipse (Addison-Wesley). Addison-Wesley.

[Stahl et al. 2006] Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, and Management*. John Wiley & Sons.

[Steinberg et al. 2009] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

[Taylor et al. 2009] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing.

[Voelter et al. 2013] Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L. C. L., Visser, E., and Wachsmuth, G. (2013). *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org.

[Wile 1997] Wile, D. S. (1997). Abstract syntax from concrete syntax. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE'97, pages 472–480, New York, NY, USA. ACM.