Universidade Federal do Espírito Santo
Programa de Pós-Graduação em Informática – PPGI
Aspectos Teóricos de Ontologias de Fundamentação

Nome: Antognoni Fundão de Albuquerque

Professor: Giancarlo Guizzardi

Relatório Final

1 Introdução

O objetivo do presente documento é relatar as atividades desenvolvidas ao longo da disciplina "Aspectos Teóricos de Ontologias de Fundamentação", ministrada no período letivo 2011/2, que teve como resultado a criação de um editor integrado de OntoUML.

2 Objetivos

Os objetivos na disciplina foram integrar diversas contribuições geradas por trabalhos anteriores em torno da linguagem OntoUML [1] e familiarizar o aluno com as tecnologias envolvidas em tais contribuições. Inicialmente foi considerada a integração dos seguintes componentes/funcionalidades no escopo do trabalho:

I. Editor de OntoUML Desktop por Alessander Benevides (OntoUML Editor) [2]
II. Transformação de OntoUML para OWL por Veruska Zamborlini (OntoUML2OWL) [3]
III. Transformação de OntoUML para Alloy por Bernardo Braga (OntoUML2Alloy) [4]
IV. Transformação de OntoUML para Alloy considerando diferentes mundos possíveis por Alessander Benevides (OntoUML2Alloy) [5]
V. Editor de OntoUML Web por Alex Pinheiro (OntoUML Web Editor) [6]
VI. Aplicação de Design Patterns na criação de Modelos OntoUML por Alex Pinheiro (DP Model) [6]

3 Desenvolvimento do Trabalho

Para alcançar os objetivos estabelecidos, o primeiro passo foi definir por onde iniciar a integração dos trabalhos. Com exceção das ferramentas de modelagem (itens I e V da listagem anterior) as demais contribuições foram implementadas como funcionalidades ou serviços a serem consumidos por outras aplicações. Logo, foi constatado que o ponto de partida para

integração poderia ser um dos editores já desenvolvidos. Para determinar qual editor utilizar foi feita uma análise de cada ferramenta segundo requisitos não-funcionais essenciais para editores OntoUML. É importante observar que os requisitos foram determinados levando em consideração o feedback recebido pelos usuários e desenvolvedores dos editores OntoUML existentes. Os requisitos foram classificados segundo a estrutura da norma ISO/IEC 9126 [7]:

a) **Integração de funcionalidades (Interoperabilidade)**: O editor deve integrar em uma única ferramenta as diversas funcionalidades/componentes já desenvolvidos em torno da OntoUML para que o usuário final possa efetivamente utilizá-los.

b) **Facilidade de utilização (Inteligibilidade):** O editor precisa ser fácil de usar e intuitivo para que usuários com pouca ou nenhuma familiaridade com ferramentas de modelagem possam editar e validar modelos OntoUML.

c) **Qualidade gráfica dos diagramas gerados (Conformidade):** Os diagramas gerados pelo editor precisam ser bem acabados, isto é, ter qualidade gráfica semelhante aos editores mais populares de UML. Verificou-se que alguns modeladores realizavam a modelagem e validação com as ferramentas OntoUML disponíveis e, uma vez corretos sintaticamente, recriavam os diagramas em ferramentas UML (como Astah [8]) por possuírem melhor diagramação.

d) **Compatibilidade com UML 2.0 (Conformidade):** Como a OntoUML é basicamente um estereótipo para a UML, os modelos gerados pelos editores precisam estar em conformidade com o a UML de forma a ser compatível com o formato XMI para intercâmbio de modelos que possam ser importados em outras ferramentas.

e) **Transparência na simulação do com Alloy (Operacionalidade):** O editor deve permitir o usuário abstrair dos aspectos técnicos da simulação com Alloy [9], como por exemplo, o carregamento e a execução manual da simulação usando o Alloy Analyzer.

f) **Redução do vínculo com plataformas de terceiros (Manutenibilidade):** A ferramenta deve possuir o menor vínculo possível com plataformas de

terceiros para que seja capaz de evoluir de maneira independente, não sendo limitados por estas.

g) **Facilidade de Acesso/Instalação/Deployment (Portabilidade):** O editor deve ser fácil de instalado (se aplícável) e acessado pelos seus usuários finais.

h) **Facilidade para manter e evoluir a ferramenta (Manutenibilidade):** O editor precisa ser fácil de manter para que outros desenvolvedores (principalmente aqueles que possuem apenas conhecimento em Java – perfil da maioria dos estudantes de graduação/mestrado) possam dar continuidade e evoluir a ferramenta.

3.1 Avaliação do OntoUML Editor

O OntoUML Editor é um plugin para a IDE Eclipse criado por Alessander B. Benevides utilizando a tecnologia Eclipse Modeling Framework (EMF) [10]. Suporta a edição de modelos OntoUML e validação nos modos "batch" (sob demanda) e "live" (na hora) que impede o usuário de cometer erros segundo a sintaxe da linguagem. A tecnologia EMF se baseia na transformação de modelos e geração automatizada de código. A partir da definição de um meta-modelo são realizadas transformações sucessivas pelas ferramentas disponíveis no EMF até que se chegue ao código Java (executável) de um editor gráfico para os diagramas do meta-modelo. A principal vantagem dessa abordagem reside na redução dos esforços de codificação e o aumento a preocupação com o meta-modelo em si.

Se por um lado o EMF facilita a criação do Editor pela geração do código, por outro ele restringe as possibilidades do mesmo, principalmente quando é necessário customizar código gerado pela transformação. Embora seja possível parametrizar as transformações até certo ponto, em alguns momentos são necessárias customizações mais específicas, o que pode ser particularmente difícil, visto que não existe muita documentação disponível. Além disso, os elementos gráficos usados para representar os construtores da meta-modelo são muito genéricos e não possuem a mesma qualidade gráfica esperada de um editor UML. Outro aspecto a ser considerado é que o meta-modelo do editor foi adaptado para facilitar as transformações, resultando em meta-modelo não conforme ao meta-modelo original da UML. O OntoUML Editor também sofre restrições no tocante à plataforma utilizada, pois depende do Eclipse e dos componentes do EMF para funcionar, que nem sempre podem estar disponíveis ou com a versão correta.

Adicionalmente, o OntoUML Editor disponibiliza a funcionalidade de transformação de OntoUML para Alloy, para que sejam realizadas simulações usando o Alloy Analyzer. Entretanto, como resultado da transformação é gerado um arquivo Alloy e é necessário que o próprio usuário realize a entrada na ferramenta Alloy Analyzer para fazer a simulação, que deve ser aberta externamente. Esse foi um trabalho pioneiro e de extrema importância para a comunidade de Engenharia de Ontologias que precisava de uma ferramenta para validar modelos OntoUML e até então não havia editores similares disponíveis. O OntoUML Editor e serviu de base para criação do editor de OntoUML Web que será avaliado na próxima seção.

3.2 Avaliação do OntoUML Web Editor

O OntoUML Web Editor é uma ferramenta criada por Alex P. das Graças e implementada como uma aplicação Web que permite a edição de modelos com o apoio de padrões da linguagem OntoUML. O editor também realiza a captura do Design Rationale, isto é, o processo de decisão que levou ao modelador à escolher determinado construtor da linguagem para representar um conceito do domínio. O editor Web foi construído com base no mesmo meta-modelo do editor anterior (Desktop) e também apresenta o mesmo problema de não conformidade com o meta-modelo original da UML.

Diferentemente do seu antecessor, na criação do OntoUML Web Editor houve uma grande preocupação com arquitetura e a implementação da solução. A aplicação foi estruturada segundo o paradigma cliente-servidor, onde do lado cliente foram empregadas técnicas como AJAX e as bibliotecas Google Web Toolkit (GWT) [11] e SmartGWT [12] para manipulação dos elementos visuais do diagrama. No lado servidor empregadas tecnologias Ecore para manipulação do meta-modelo e Hibernate para persistência dos elementos dos modelos e do Rationale capturado. Apesar dos aspectos visuais dos elementos do editor Web estarem mais próximos da UML, eles ainda deixam a desejar não atendendo ao requisito de qualidade gráfica dos diagramas gerados.

Devido sua natureza, o editor Web traz diversas vantagens como a facilidade de acesso e a centralização dos modelos, constituindo uma espécie de repositório on-line para modelos, o que é bem interessante do ponto de vista colaborativo. Entretanto as aplicações Web ainda enfrentam diversas limitações no tocante à utilização de recursos, como por exemplo, acesso ao sistema de arquivos, acesso ao banco de dados e restrições nos componentes e bibliotecas utilizados na camada cliente. Para contornar essas limitações, no editor Web todo o processamento relacionado à edição do modelo e captura do Design Rationale ocorre do lado servidor, enquanto que apenas a apresentação e diagramação ocorre do lado cliente. Esse

fator aumenta consideravelmente a complexidade da solução, pois além de prover as funcionalidades básicas do editor é necessário prover mecanismos adicionais para garantir o sincronismo entre as partes, uma vez que as informações trafegam por um meio não confiável (HTTP). Os editores foram avaliados segundo cada requisito levantado e classificados segundo os critérios “Atendidos Plenamente”, “Atendidos Parcialmente” e “Não Atendidos” em cada requisito. A seguir pode ser visto na Tabela 1 uma comparação dos editores com os requisitos estabelecidos, resumindo o processo de a avaliação:

Tabela 1 - Editores x Requisitos

Requisito	Editor Desktop (A. Benevides)	Editor Web (A. Graças)
Integração de funcionalidades	P	P
Facilidade de utilização	P	P
Qualidade gráfica dos diagramas gerados	N	N
Compatibilidade com UML 2.0	N	N
Transparência na simulação com Alloy	N	N
Redução do vínculo com plataformas de terceiros	N	P
Facilidade de Acesso/Instalação/Deployment	P	A
Facilidade para manter e evoluir a ferramenta	N	N

Legenda: **A** – Atende Plenamente, **P** – Atende Parcialmente, **N** – Não Atendido

Após analisar as duas ferramentas de modelagem disponíveis, verificou-se que nenhuma delas atendia plenamente os requisitos essenciais para editores OntoUML. Portanto, foi considerada uma terceira abordagem que compreendia a customização de um editor UML open source e independente de plataforma para atender aos requisitos levantados. Nessa abordagem foram considerados alguns editores como Oryx [13], StarUML [14], UMLet [15], TinyUML [16], UmlCanvas [17], GWT UML [18] e ArgoUML [19]. Dentre os editores investigados, o TinyUML se destacou por ser um editor estruturalmente simples, leve (consume poucos recursos) e não necessitar de instalação para uso. As vantagens da customização de um editor já desenvolvido são: controle total do código fonte do editor, permitindo a realização de customizações mais específicas, a redução do vínculo com projetos de terceiros, maior proximidade visual com UML, maior qualidade gráfica dos diagramas gerados dado que é possível alterar os elementos visuais do diagrama e facilidade na utilização e acesso/instalação/deployment uma vez que a ferramenta pode ser distribuída em um único arquivo executável. As desvantagens da customização do editor existente são: maior esforço de desenvolvimento, visto que inicialmente é necessário entender como o editor funciona e para então customizá-lo e maior esforço para integração e atendimento aos requisitos restantes.

3.2 Arquitetura da Solução

Mesmo com as desvantagens apresentadas, optou-se por adotar a terceira abordagem como para realização da integração e a ferramenta escolhida foi a TinyUML. Esta escolha pôde ser justificada principalmente devido à necessidade de adaptação/customização do editor para acomodar os elementos a serem integrados. Sendo mais simples, a codificação da integração e posteriormente a manutenção do editor pode ser feita mais facilmente. Outro ponto considerado é que como a adaptação é inevitável, ela pode ser vista também como uma oportunidade para criação de uma plataforma reutilizável para o desenvolvimento de trabalhos relacionados à OntoUML. Em terceiro lugar, a ferramenta de modelagem UML por si já atendia à alguns dos requisitos levantados sem a necessidade de grandes intervenções. A independência de plataforma (como a IDE Eclipse) e a possibilidade de realizar customizações mais específicas foram fatores decisivos na escolha da abordagem.

Uma vez definido qual editor utilizar como base, deu-se início ao processo de estruturação da ferramenta integrada. Os primeiros passos foram verificar como o editor TinyUML poderia ser adaptado para usar a sintaxe da OntoUML e o processo de validação semântica dos modelos gerados. Verificou-se que a ferramenta possuía um meta-modelo não conforme com a UML e que precisava ser adaptado para realizar a validação semântica. Por outro lado, verificou-se também que já existia um meta-modelo da OntoUML produzido por Roberto Carraretto em seu projeto de graduação em conformidade com a UML 2.0 denominado RefOntoUML [20]. O objetivo do RefOntoUML é justamente prover um modelo de referência para integração de aplicações/funcionalidades relacionadas à OntoUML. Outro aspecto positivo do RefOntoUML é que ele já inclui regras para validação semântica de modelos. Decidiu-se utilizar o RefOntoUML como meta-modelo da ferramenta, substituindo o meta-modelo nativo. Foram necessários diversos ajustes na ferramenta para utilização do RefOntoUML, incluindo nas funcionalidades de serialização e persistência dos modelos gerados. Os próximos passos na estruturação foram acomodar os componentes OntoUML2OWL (Veruska Zamborlini) para geração de OWL e OntoUML2Alloy (Bernardo Braga) para geração de Alloy de forma a serem utilizados na ferramenta. A Figura 1 ilustra a arquitetura da aplicação considerando a abordagem adotada. Cada caixa colorida representa um componentes integrados e as cores representam compatibilidade dos componentes entre si.

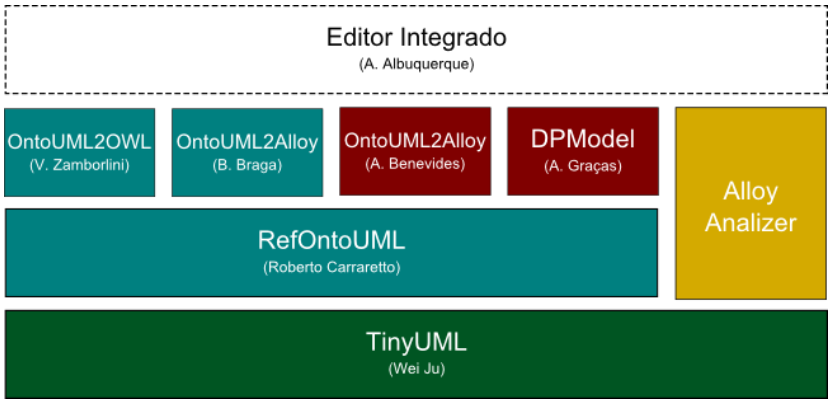

Figura 1 - Arquitetura da solução

Na sequência, o próximo componente a ser integrado foi o Alloy Analyzer, provendo a funcionalidade integrada de geração e simulação de modelos OntoUML com Alloy. Os componentes OntoUML2Alloy (Alessander Benevides) e DPModel (Alex Graças) ainda precisam ser adaptados para a estrutura do RefOntoUML. Na seção a seguir pode ser visto um resumo dos resultados alcançados e posteriormente são detalhados os próximos passos no desenvolvimento do editor.

4 Conclusão e Resultados Alcançados

Este documento relatou as atividades desenvolvidas ao longo da disciplina “Aspectos Teóricos de Ontologias de Fundamentação” que teve como resultado a integração de diversos trabalhos relacionados à linguagem de modelagem OntoUML, dando origem à um editor OntoUML integrado. Pode-se dizer que os objetivos iniciais foram alcançados, visto que foi produzida uma versão inicial funcional do editor e foi adquirida familiaridade com as tecnologias dos trabalhos em questão. Os resultados alcançados são sumarizados a seguir:

- Criação de uma versão funcional do editor de OntoUML, com as funcionalidades de edição, salvamento e validação do modelo.
- Integração das funcionalidades de geração de OWL e Alloy.
- Integração com Alloy Analyzer, permitindo com que o usuário simule e visualize o modelo editado, sem a necessidade de executar o Alloy Analyzer externamente.
- Aprendizado de tecnologias/linguagens como Ecore, Alloy, OCL e OntoUML.

É importante observar que no decorrer do trabalho surgiram diversos questionamentos sobre qual tipo de abordagem seria mais adequada para implementação de

editores no contexto da OntoUML, se aplicações Web ou aplicações Desktop seriam mais adequadas. Estes questionamentos são relevantes, pois existe uma crescente tendência de aplicações Desktop-like na nuvem (tais como Oryx) e, mais do que isso, o caráter colaborativo da Engenharia de Ontologias pode ser mais bem explorado por meio de editores on-line. Dentro do escopo do presente trabalho, foi mais adequada a abordagem de aplicação Desktop, que pode justificada pela principal característica do editor: o reuso dos componentes e funcionalidades já desenvolvidas. No mais, pode-se concluir que as diferentes abordagens podem ser complementares, isto é, espaço para os dois tipos de aplicação. Pode existir um repositório on-line de modelos e um editor desktop que se conecta ao repositório para recuperar modelos, padrões de modelagem e permitir a edição colaborativa de modelos.

5 Perspectivas Futuras

Na primeira versão do editor integrado os esforços foram concentrados na adaptação dos aspectos funcionais do mesmo, como por exemplo, funcionalidades básicas de edição e persistência de modelos, adequação para validação semântica e a adequação da estrutura para suportar a integração dos componentes como Alloy Analyzer. É necessário ainda adaptar o editor para suportar relações entre de relações, como por exemplo, especialização de associações e relações de derivação, dentre outras pendências menores relacionadas à questões de usabilidade. A seguir são sumarizadas as perspectivas futuras da ferramenta:

- Aperfeiçoamento da simulação com Alloy baseado em padrões de erros
- Aperfeiçoar a simulação com Alloy a partir de restrições OCL
- Integração do OntoUML2Alloy (Alessander Benevides) e DPModel (Alex P. Graças)
- Exploração de novas formas de visualização de modelos
- Realização de experimentos (abordagens de modelagem, construtores mais utilizados)
- Geração de código ECMA Script a partir de modelos OntoUML
- Integração com Repositório de modelos/padrões
- Melhorias gerais e correção de bugs

6 Referências Bibliográficas

[1] Guizzardi, G. “Ontological foundations for structural conceptual models”. Thesis (Ph.D.) — University of Twente, Enschede, The Netherlands, Enschede, October 2005. Available from Internet: <http://doc.utwente.nl/50826/>. Cited 01/13/2009.

[2] Benevides, A. B.; “A Model-based Graphical Editor for Supporting the Creation, Verification and Validation of OntoUML Conceptual Models”. Dissertação de Mestrado em Informática, Universidade Federal do Espírito Santo, 2010.

[3] Zamborlini, V. “Estudo Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML para OWL Abordagens para Representação de Informação Temporal”. Dissertação de Mestrado em Informática, Universidade Federal do Espírito Santo, 2011.

[4] Braga, B. F. B.; Almeida, J. P. A.; Guizzardi, G.; Benevides, A. B. “Transforming OntoUML into Alloy: Towards conceptual model validation using a lightweight formal method”. Innovations in Systems and Software Engineering (ISSE), Springer-Verlag, London, vol. 6, no. 1, p. 55–63, 2010. (Print). Available from Internet: <http://www.springerlink.com/content/m1715n1220717l58/fulltext.pdf>. Cited 09/21/2010.

[5] Benevides, A.B.; Guizzardi, G.; Braga, B.F.B.; Almeida, J.P.A.;, “Assessing Modal Aspects of OntoUML Conceptual Models in Alloy”, International Workshop on Evolving Theories of Conceptual Modeling (ETheCoM 2009), at the 28th International Conference on Conceptual Modeling (ER 2009), Gramado, Brazil.

[6] Graças, A. P. “Suporte Automatizado Para Construção de Modelos Conceituais Bem Fundamentados”. Dissertação de Mestrado em Informática, Universidade Federal do Espírito Santo, 2010.

[7] ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. Engenharia de software -qualidade de produto: modelo de qualidade: NBR ISO/IEC 9126-1, Rio de Janeiro, 2003. 21p.

[8] Visual Paradigm, “Astah Community”. Disponível em: http://astah.net/editions/community. Data de acesso: 07/12/2011

[9] Jackson, D. “Alloy: a lightweight object modelling notation”. Transactions on Software Engineering and Methodology (TOSEM), ACM, New York, NY, USA, vol. 11, no. 2, p. 256–290, 2002. ISSN 1049-331X.

[10] Steinberg, D.; Budinsky, F.; Patenostro, M.; Merks, E. EMF: Eclipse Modeling Framework 2.0. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0321331885.

[11] Google Inc. “Google Web Toolkit (GWT)”. Disponível em : http://code.google.com/intl/pt-BR/webtoolkit/. Acessado em: 08/12/2011.

[12] SmartClient. “Smart GWT”. Disponível em : http://code.google.com/p/smartgwt/. Acessado em: 08/12/2011.

[13] Business Process Technology Group. “Oryx”. Disponível em : http://bpt.hpi.uni-potsdam.de/Oryx/WebHome. Acessado em: 08/12/2011.

[14] StarUML . “StarUML”. Disponível em: http://staruml.sourceforge.net/en/. Acessado em: 08/12/2011.

[15] Information and Software Engineering Group. “UMLet”. Disponível em: http://www.umlet.com/. Acessado em: 08/12/2011.

[16] Wei Ju. “TinyUML”. Disponível em: http://sourceforge.net/projects/tinyuml/. Acessado em: 08/12/2011.

[17] Van Ginneken, C. “UmlCanvas”, Disponível em: http://umlcanvas.org/. Acessado em: 08/12/2011.

[18] ODLabs. “GWT UML”. Disponível em: http://code.google.com/p/gwtuml/. Acessado em: 08/12/2011.

[19] Tigris.org. “ArgoUML”. Disponível em: http://argouml.tigris.org/. Acessado em: 08/12/2011.

[20] Carraretto, R. “A Modeling Infrastructure for OntoUML”. Trabalho de Conclusão de Curso em Ciências da Computação, Universidade Federal do Espírito Santo, 2010.