UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

TIAGO LOPES TELECKEN

Proposta de Suporte XML para Ambientes de Desenvolvimento de Sistemas Visuais Interativos Baseados em Gramáticas

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. José Valdeni de Lima
Orientador

Porto Alegre, julho de 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Telecken, Tiago Lopes

Proposta de Suporte XML para Ambientes de Desenvolvimento de Sistemas Visuais Interativos Baseados em Gramáticas / Tiago Lopes Telecken – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

138 f.:il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Orientador: José Valdeni de Lima.

1.XML. 2.Sistemas Visuais Interativos 3.Ambientes de desenvolvimento de software. I. Lima, José Valdeni de. Título.

AGRADECIMENTOS

Agradeço primeiramente a minha esposa Juliane, aos meus pais Carlos e Osana e aos meus irmãos Lucas e Vanessa pelo apoio, incentivo e compreenção.

Agradeço ao meu orientador José Valdeni de Lima pela atenção, confiança, apoio e ensinamentos. Ao supervisor do meu estágio sanduíche Ethan V. Munson pela receptividade, atenção e orientações dadas enquanto estive na UWM.

Agradeço também a todos que de alguma forma colaboraram com este trabalho seja questionando, sugerindo, discutindo ou incentivando. Em especial vai um agradecimento aos mestres e colegas de disciplinas do PPGC da UFRGS pelos ensinamentos e companherismo. Aos funcionários, professores e alunos que passaram pelo laboratório 251 durante os anos do doutorado e acabaram tornando-se parceiros de viajem e amigos. Aos amigos de Milwaukee sem os quais a adaptação, trabalho e diversão em terras distantes seriam bem mais difíceis. E a todos os bons e velhos amigos e familiares que me acompanham desde tenra idade. Todos de uma forma ou outra me ajudaram nesta longa travesia.

Agradeço ao CNPq pelo apoio financeiro.

E finalmente a Deus pela vida.

Obrigado a todos.

Quando nada parece ajudar, eu vou e olho o cortador de pedras martelando sua rocha talvez cem vezes sem que nem uma só rachadura apareça. No entanto, na centésima primeira martelada, a pedra se abre em duas e eu sei que não foi aquela a que conseguiu, mas todas as que vieram antes.

Jacob Riis

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ADBG	Ambiente de Desenvolvimento Baseado em Gramáticas
API	Application Programming Interface
DTD	Document Type Definition
GXL	Graph eXchange Language
NVDL	Namespace-based Validation Dispatching Language
S	Structure language
SVG	Scalable Vector Graphics
SV-XML	Sentença Visual baseada no XML
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VCARW	Visual Conditional Attributed ReWriting systems
Vitransf	Visual Interface Transformations
W3C	World Wide Web Consortium
XML	Extensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Ambientes de Desenvolvimento Baseados em Gramática (ADBG) utilizam uma rigorosa semântica e sintaxe para prover poderosas ferramentas que são capazes de especificar com precisão as propriedades de um sistema visual interativo. Por intermédio destas ferramentas, os ADBGs auxiliam a edição de gramáticas e a partir destas gramáticas geram o sistema visual interativo especificado.

Estes ambientes utilizam componentes dos mais diferentes espaços tecnológicos e esta tese em particular está focalizada no estudo da utilização de componentes da *eXtended Markup language* (XML) em ADBGs. Componentes XML oferecem inúmeras soluções que visam a interoperabilidade e armazenamento de dados. Porém, apesar de amplamente difundidos a complexidade destes componentes causa uma série de problemas que vão desde o baixo desempenho de processamentos até a inviabilização da implementação de operações mais elaboradas.

Com o objetivo de diminuir esta complexidade, a presente tese propõe um conjunto simplificado de componentes XML. O conjunto dos componentes XML propostos e seus relacionamentos formam um framework que pode ser utilizado para se construir ADBGs. Após a definição do framework, o mesmo foi utilizado para se implementar o protótipo de um ADBG. A implementação e uso deste protótipo demonstrou concretamente a viabilidade e aplicabilidade das propostas desta tese.

Já para demonstrar a referida diferença de complexidade, os componentes XML do framework proposto foram comparados com os componentes XML dos demais ADBGs. A comparação revelou que os componentes XML das demais abordagens têm entidades adicionais que não estão presentes nos componentes propostos. Este conjunto adicional de entidades comprovou a maior complexidade dos componentes XML utilizados nos demais ADBGs. Adicionalmente, a referida comparação demonstrou que mesmo sendo mais simples os componentes propostos mantêm as mesmas funcionalidades que os componentes atualmente utilizados.

Palavras-Chave: ADBG, XML, sistemas visuais interativos, ambientes de desenvolvimento de software, alfabeto visual, sentença visual, gramática, edição.

A XML support propose to development environments of interactive visual systems based on grammars

ABSTRACT

Development Environments Based on Grammar (DEBGs) often rely on rigorous syntax and semantics, which provide powerful tools to fully specify visual interactive systems and its properties. Such environments provide to visual interactive system developers a set of editors to aid the grammar specification. From these grammars specifications the DEBGs can generate the specified visual interactive systems.

Such environments can use components from several technological spaces. However this thesis is focused on the application of *eXtended Markup language* (XML) components in DEBGs. XML components provide many solutions in terms of interoperability and data storing. Despite of its widely application, the complexity of XML components is the cause of problems such as low processing performance and inviability of many implementations.

Aiming to decrease such complexity, we propose a set of simplified XML components. The set of proposed XML components and its relationships compose a framework that can be used to develop DEBGs. After the framework specification, the framework was used to implement a DEBG prototype. Such prototype shows the viability and applicability our proposals.

We demonstrate the referred complexity difference comparing the XML components proposed in this thesis with the XML components applied in others DEBGs. Basically, the XML component of others DEBGs has all entities of correspondent component proposed in this thesis more a set of extra entities. This set of extra entities demonstrates the greater complexity of other approaches. Additionally, the referred comparison demonstrates that despite of its simplifications, the proposed components have the same functionalities than the current components.

Keywords: DEBG, XML, visual interactive systems, software development environment, visual alphabet, visual sentence, grammar, edition.

1 INTRODUÇÃO

Com o objetivo de melhorar o desenvolvimento de softwares, pesquisadores e a indústria, constantemente, procuram por modelos, componentes e artefatos que minimizem a complexidade dos sistemas produzidos, facilitem a sua análise e promovam o reuso de seus processos e produtos. Neste contexto a presente tese propõe um conjunto simplificado de componentes que mesmo sem perder suas funcionalidades originais podem simplificar a estrutura de uma classe de aplicações, facilitar a sua análise e promover o reuso de um grande conjunto de produtos e processos. A classe de aplicações estudadas são os Ambientes de Desenvolvimento Baseado em Gramática (ADBGs) e o conjunto de produtos e processos que podem ser reutilizados pertencem ao espaço tecnológico XML.

ADBGs utilizam uma rigorosa semântica e sintaxe para prover poderosas ferramentas que são capazes de especificar com precisão as propriedades de um sistema visual interativo (COSTAGLIOLA, 2004). Por intermédio destas ferramentas, os ADBGs auxiliam a edição de gramáticas e a partir destas gramáticas geram o sistema visual interativo especificado.

Os sistemas visuais interativos gerados são artefatos eletrônicos que reagem e mudam a sua configuração de acordo com comandos de usuários e sistemas externos, além disto, são artefatos que usam mais de uma dimensão para transmitir semântica i.e. usam notações visuais em duas ou mais dimensões (BARDOHL, 1999). Tais características englobam uma grande variedade de sistemas como, por exemplo, simulações visuais, animações, jogos, planilhas eletrônicas e editores de diagramas (editores UML, ER, de redes de Petri etc).

As gramáticas editadas nos ADBGs definem um sistema visual interativo como um conjunto de sentenças visuais válidas, onde cada sentença é composta de unidades visuais perceptíveis chamadas de símbolos. Os símbolos são tipados sendo que a estrutura de dados comum a todos os tipos de símbolos está definida em um modelo de sentença visual e as características específicas de cada tipo de símbolo estão definidas em um alfabeto (BOTTONI, 2007). Mais precisamente as gramáticas especificam como os tipos de símbolos definidos em um alfabeto podem ser combinados para formar sentenças visuais que são mostradas aos usuários do sistema visual interativo.

Os ADBGs utilizam componentes dos mais diferentes espaços tecnológicos e esta tese em particular está focalizada no estudo da utilização de componentes XML em ADBGs. A tecnologia XML (BRAY, 2006-a) tornou-se hoje em dia um padrão de fato em termos de interoperabilidade e armazenamento de dados. Consequentemente muitos ADBGs utilizam componentes XML. Em ADBGs desenvolvidos sobre a plataforma XML os dados (sentenças visuais, alfabetos e gramáticas) são realizados em documentos XML e os modelos destes dados (modelo de sentença visual, meta-modelo

de alfabeto e meta-modelo de gramática) são realizados em esquemas XML. Este é um padrão amplamente difundido e comum a todos ADBGs que utilizam componentes XML para realizar seus dados e modelos. Entretanto a complexidade das estruturas destes componentes causa uma série de problemas que vão desde o baixo desempenho dos processamentos até a inviabilização da implementação de operações mais elaboradas (NICOLA, 2003).

Com o objetivo de diminuir esta complexidade, a presente tese propõe um conjunto simplificado de componentes XML. Inicialmente foi proposto que uma especificação do modelo XML (BRAY, 2006-a), um componente mais simples que um esquema XML, realize o modelo de sentença visual de um ADBG. Adicionalmente é proposto que todas as demais estruturas de dados, operações e sistemas do ADBG sejam realizadas por componentes XML compatíveis com esta especificação. O conjunto de componentes XML propostos e seus relacionamentos formam um framework de projeto físico que pode ser utilizado para se construir ADBGs. O framework é dito de projeto físico pois é formado por um conjunto de componentes que podem "realizar" o modelo lógico de um ADBG.

Para que se possa utilizar os componentes propostos, há um requisito que deve ser atendido no modelo lógico do ADBG: o modelo de sentença visual precisa ser o modelo XML. O uso do modelo XML como um modelo de sentença visual também é uma proposta desta tese.

O modelo XML interpretado como um modelo de sentença visual bem como o framework de projeto físico de ADBG foram respectivamente chamados de modelo e framework SV-XML (Sentença Visual XML) (TELECKEN, 2008). As hipóteses, contribuições e metodologias desta tese são descritas a seguir.

1.1 Hipóteses

Esta tese propôs, verificou e comprovou as seguintes hipóteses.

A primeira hipótese supõe que o modelo XML possa ser usado como um modelo de sentença visual de um ADBG e que um framework de projeto físico de ADBGs possa ser especificado com componentes XML compatíveis com este modelo de sentença visual. O principal argumento desta suposição é que as entidades e relacionamentos do modelo XML são semelhantes às entidades e relacionamentos de modelos de sentença visual.

Como a especificação de um modelo XML é um componente mais simples que um esquema XML a segunda hipótese desta tese supõem que os ADBGs construídos a partir do framework proposto serão mais simples que os ADBGs que utilizam um esquema XML para realizar o modelo de sentença visual. O principal argumento desta suposição é que a maior simplicidade do componente que realiza o modelo de sentença visual é propagada para os componentes que realizam as demais estruturas, operações e sistemas do ADBG.

A terceira hipótese supõe que os ADBGs construídos a partir do framework proposto aumentam o suporte que operações e sistemas XML independentes de domínio podem dar aos ADBGs. Este suporte possibilita que componentes XML já prontos, amplamente testados e difundidos sejam integrados aos ADBGs sem a necessidade de adaptações nestes componentes. O principal argumento desta suposição é que tanto os

ADBGs construídos a partir do framework SV-XML quanto a maioria das operações e sistemas XML tem como base o modelo XML.

1.2 Contribuições

As principais contribuições desta tese estão relacionadas à utilização de componentes XML em ADBGs. As principais vantagens de se utilizar componentes XML em ADBGs de acordo com as propostas desta tese em relação ao modo como os componentes XML são utilizados atualmente são:

- Desenvolvedores de ADBGs e desenvolvedores de sistemas visuais interativos que utilizam ADBGs poderão administrar componentes XML mais simples. Apesar de suas simplificações estes componentes executarão as mesmas funcionalidades que componentes mais complexos.
- Desenvolvedores de ADBGs e desenvolvedores de sistemas visuais interativos que utilizam ADBGs poderão integrar ao ADBG um maior número de operações e sistemas XML independentes de domínio sem a necessidade de adaptá-los.
- Esta tese propôs que diferentes componentes XML sejam utilizados para realizar sentenças visuais, alfabetos e seus modelos em ADBGs. Tal proposta por si só já é uma contribuição relevante, pois abre novas possibilidades de se explorar os componentes XML em ADBGs.

1.3 Metodologia

Primeiramente as estruturas de dados, operações e sistemas de um ADBG foram categorizados e modelados. Foram utilizados neste processo documentos e artigos científicos da área de ambientes de desenvolvimento baseado em gramáticas.

Depois foi verificado como estes sistemas utilizam componentes XML. Para tanto foi feito um mapeamento das entidades típicas de ADBGs para os componentes XML que realizam estas entidades em ADBGs. A partir da análise destes mapeamentos foram notados e relatados padrões comuns de utilização de componentes XML. Dentre estes padrões, um destacou-se: todos os ADBGs que utilizam componentes XML para realizar o modelo de sentença visual realizam este modelo através de esquemas XML.

E foi exatamente a partir do modelo de sentença visual que a proposta desta tese começou a ser definida. Inicialmente, foi definido que o modelo XML seria o modelo de sentença visual do SV-XML. Um mapeamento entre entidades do modelo XML e entidades típicas de modelos de sentença visual demonstrou que o modelo XML pode apropriadamente ser um modelo de sentença visual. Em outro momento a expressividade do modelo SV-XML foi formalmente demonstrada.

Subsequentemente foi definido o framework de projeto físico SV-XML. O framework SV-XML define que uma especificação XML é o componente XML que realiza o modelo de sentença visual. A partir deste componente foram definidas as demais estruturas, operações e sistemas da proposta. Somente componentes XML foram utilizados para realizar as estruturas de dados, operações e sistemas do SV-XML.

Também através de mapeamentos foi demonstrada a compatibilidade entre as entidades típicas de ADBGs e os componentes XML que as realizam no SV-XML.

O passo seguinte a especificação do framework foi a implementação de um ADBG que utilizou as recomendações do framework SV-XML. O ADBG implementado chama-se Vitransf (TELECKEN, 2008). Esta implementação demonstrou concretamente a viabilidade e aplicabilidade de um ADBG que segue as recomendações do framework SV-XML. Após a implementação do Vitransf, dois estudos de casos foram apresentados. Nestes estudos de casos dois sistemas visuais interativos foram especificados através do ambiente de desenvolvimento do Vitransf e executados em seu ambiente de execução.

Com a disponibilização de todos os dados sobre a aplicação de componentes XML no SV-XML e nos demais ADBGs, a etapa final desta tese foi comparar os dois modos de aplicação de componentes XML. Primeiramente os componentes XML utilizados nas duas abordagens foram comparados item por item. O objetivo foi demonstrar as diferenças entre os dois modos de aplicação de componentes XML e comprovar a originalidade das propostas desta tese.

Para demonstrar a diferença de complexidade entre as duas abordagens, as entidades do componente XML que realiza o modelo de sentença visual do SV-XML foram comparadas com as entidades do componente XML que realiza este modelo nos demais ADBGs. A comparação revelou que o componente XML que realiza o modelo de sentença visual dos demais ADBGs (um esquema XML) têm entidades adicionais que não estão presentes no correspondente componente do SV-XML (uma especificação do modelo XML). Este conjunto adicional de entidades comprovou a maior complexidade dos componentes XML utilizados nos demais ADBGs. Adicionalmente, a referida comparação demonstrou que mesmo sendo mais simples o componente proposto mantem as mesmas funcionalidades que os componentes atualmente utilizados. Abordagens semelhantes foram usadas para se comparar componentes XML que realizam as demais estruturas de dados, operações e sistemas das abordagens analisadas.

Para comparar o uso de operações e sistemas XML independentes de domínio no SV-XML e nas demais abordagens foi verificado quais operações típicas de ADBGs poderiam ser executadas por operações e sistemas XML independentes de domínio em cada uma das abordagens. Nesta comparação foi verificado que no SV-XML mais operações típicas de ADBGs podem ser executadas por operações e sistemas XML independentes de domínio.

Os demais capítulos desta tese estão distribuídos da seguinte maneira. O capítulo 2 faz uma revisão bibliográfica sobre ADBGs, incluindo a categorização de suas estruturas, operações e sistemas. O capítulo 3 apresenta alguns ADBGs existentes. O capítulo 4 faz uma revisão bibliográfica sobre os principais componentes XML utilizados em ADBGs e em especial dos componenetes XML utilizados no SV-XML. O capítulo 5 apresenta as propostas desta tese: o modelo SV-XML e o framework SV-XML. O capítulo 6 apresenta o protótipo Vitransf que foi desenvolvido com base nas especificações e recomendações do framework SV-XML. O capítulo 7 apresenta os trabalhos relacionados, ou seja, como os atuais ADBGs aplicam componentes XML. O capítulo 8 compara e analisa as diferenças da aplicação de componentes XML no SV-XML e nos demais ADBGs. Por fim o capítulo 9 apresenta as conclusões desta tese.

1.4 Notações e conceitos utilizados nesta tese

Nesta seção serão revisados conceitos e notações que serão utilizados ao longo desta tese.

Segundo a UML 2.0 (OMG, 2007), uma classe ou um diagrama de classes define os conceitos ou o modelo lógico de um sistema. Uma implementação em uma plataforma específica deste modelo lógico é descrita através de componentes. **Componente** é uma unidade lógica modular e intercambiável de um sistema que pode ser combinada com outros componentes para formar componentes maiores e também modulares. O funcionamento/comportamento de um componente em um sistema é definido em um modelo lógico (um diagrama de classes, uma descrição em linguagem natural ou formal). Assim é dito que um componente **realiza** a especificação de um modelo em uma plataforma específica. Um componente pode realizar desde a especificação de uma única classe até a especificação de estrutura de dados, operações, subsistemas e sistemas. Componentes fazem parte do modelo físico de um software. Boa parte das contribuições desta tese está relacionada com o modelo físico de ADBGs.

Fisicamente as implementações não manipulam os componentes, mas sim os artefatos que representam estes componentes. Onde **artefato** é um dispositivo físico que pode ser manipulado por um programa. Um arquivo texto, um arquivo XML, um objeto na memória, uma árvore na memória, uma tabela em um banco de dados ou um arquivo executável são exemplos de artefatos. Na linguagem UML 2.0 é dito que um artefato **manifesta** um componente. Enquanto o componente especifica os requisitos/características de uma implementação o artefato especifica uma implementação em particular. Em um modelo, um artefato não manifesta apenas componentes, um artefato também pode manifestar classes, modelos ou pacotes. A UML também usa o termo artefato para designar técnicas de programação, recomendações e qualquer outro artifício ou recurso utilizado na implementação de um software.

O termo **espaço tecnológico** é comumente utilizado no lugar dos termos "plataforma" ou "tecnologia", entretanto uma definição mais precisa é dada por Kurtev (2002):

> Espaço tecnológico é um contexto de trabalho com um conjunto associado de conceitos, área de domínio, ferramentas, artefatos, habilidades requeridas e possibilidades. É comumente associada a uma dada comunidade de usuários com suporte educacional, literatura própria, workshops, conferencias e conhecimento compartilhado. Tem ao mesmo tempo uma área já bem estabelecida de conhecimento e pesquisas em andamento. Apesar de ser difícil apresentar uma definição precisa de espaço tecnológico alguns espaços podem ser facilmente identificados. Este é o caso do espaço tecnológico XML e do espaço tecnológico de sistemas de gerenciamento de banco de dados (SGBD).

Esta tese, em particular, estuda a intersecção de dois espaços tecnológicos: o espaço tecnológico XML e o espaço tecnológico de linguagens visuais.

2 REVISÃO BIBLIOGRÁFICA – AMBIENTES DE DESENVOLVIMENTO BASEADOS EM GRAMÁTICA

Esta seção descreve primeiramente alguns conceitos do espaço tecnológico de linguagens visuais relacionados à especificação de sistemas visuais interativos. Em seguida é mostrada uma revisão bibliográfica mais aprofundada sobre ADBGs, incluindo a categorização de suas estruturas, operações e sistemas.

2.1 Modelo de dados e especificação de sistemas visuais interativos

O modelo de sistema visual interativo mostrado nesta seção foi proposto por Bottoni et al. (2007, 2003).

Neste modelo um sistema visual interativo é especificado como uma linguagem visual. Portanto um sistema visual interativo é uma instância concreta de uma linguagem visual. Utilizando-se termos da UML, é dito que o artefato sistema visual interativo manifesta uma linguagem visual.

Uma **linguagem visual** é definida como uma seqüência de sentenças visuais que são alteradas de acordo com comandos. Uma **sentença visual** é uma tela de um sistema visual interativo que é apresentada ao usuário durante um ciclo de interação. Cada sentença visual é composta de unidades lógicas chamadas de símbolos. Normalmente um **símbolo** é uma entidade gráfica percebida pelo usuário. Porém isto não é uma regra, eventualmente um símbolo pode não ser percebido pelo usuário ou simplesmente não tem uma representação gráfica.

Um conjunto de **comandos** pode ser aplicado em cada sentença visual, onde cada comando é composto de um evento que dispara uma reação. O **evento** é disparado pelo usuário ou pelo sistema. A **reação** é um processamento do computador. Dois importantes processamentos são a consulta às entidades da sentença visual e a transformação desta sentença. Portanto o conjunto de comandos especifica quando e como uma sentença visual válida pode ser transformada em outra sentença visual válida. Desta maneira uma linguagem visual tem o seguinte comportamento: uma sentença visual inicial é mostrada ao usuário. Quando um comando previsto na linguagem ocorre, a sentença visual pode ser transformada em outra sentença visual que também é mostrada ao usuário. Nesta sentença podem ser aplicados outros comandos que novamente transformarão a sentença visual formando outro ciclo de interação.

De acordo com Costagliola (2004) e Bottoni (2007), as abordagens para definir linguagens visuais através de gramáticas precisam de três especificações: um modelo de sentença visual, um alfabeto e uma gramática.

O **modelo de sentença visual** especifica uma estrutura de dados. Tal estrutura define as entidades que podem e devem estar presentes em sentenças visuais e os relacionamentos que podem e devem existir entre estas entidades. Dentre as entidades especificadas, uma ou um conjunto delas deve ser capaz de representar um símbolo. Outro conjunto deve ser capaz de representar uma sentença visual. Neste modelo também são definidas as características gerais dos comandos que poderão ser utilizados em sentenças visuais que estão de acordo com o referido modelo de sentença visual.

Costagliola et. al (2002) classificam os modelos de sentença visual em duas principais categorias: os baseados em atributos e os baseados em relações. As estruturas dos modelos baseados em atributos são símbolos que possuem atributos. Todos os símbolos que aparecem em uma sentença visual são descritos por um nome que referencia o tipo do símbolo e um conjunto de atributos que irá descrever todas as características do símbolo, inclusive os seus relacionamentos com outros símbolos.

Já as estruturas de modelos baseados em relacionamentos são formadas por entidades que representam símbolos e entidades que representam os relacionamentos entre estes símbolos. As características de um símbolo estão distribuídas entre a entidade que representa o símbolo e as entidades que representam os relacionamentos deste símbolo.

O anexo A descreve em detalhes o VCARW (BOTTONI, 2007;1999), um modelo de sentença visual baseado em atributos. Já o anexo B descreve em detalhes o GXL (HOLT, 2006), um modelo de sentença visual baseado em relacionamento.

O **alfabeto** acrescenta informações de tipos às entidades definidas no modelo de sentença visual, portanto em uma sentença visual que está de acordo com um alfabeto todas as entidades são tipadas. Um tipo define regras sintáticas que uma entidade deve obedecer para ser considerada entidade deste tipo. Se, por exemplo, no modelo de sentença visual é definido que uma entidade chamada símbolo deve possuir um *nome*, e uma lista de pares "*nome_de_ atributo* : *valor"*. Então no alfabeto vários tipos podem ser definidos com esta estrutura. Cada tipo definiria um *nome* para o tipo de símbolo, os *nomes_de_atributos* que podem fazer parte deste tipo de símbolo e seus respectivos domínios de *valores*. Como uma sentença visual também pode ser uma entidade, sentenças visuais também podem ser tipadas. Na especificação de um alfabeto a aparência gráfica e os comandos também podem ser refinados e associados a tipos de símbolos.

Quando utilizado por uma gramática o alfabeto define o conjunto de tipos de símbolos desta gramática. Ou seja, os tipos de símbolos que podem estar presentes em uma sentença visual que está de acordo com a referida gramática.

A **gramática** informa como entidades tipadas podem ser combinadas para formar uma sentença visual. A gramática também refina e define os comandos que podem disparar cada transformação.

Existem vários tipos de gramáticas (BARDOHL,1999; BOTTONI, 2003; COSTAGLIOLA, 2004; FERRI, 2007), mas de forma geral elas são formadas por: um axioma inicial (que pode ser sentença visual inicial), um conjunto de tipos de símbolos terminais (o alfabeto), um conjunto de símbolos não-terminais (alguns tipos de gramáticas não possuem o conjunto de símbolos não-terminais) e um conjunto de regras que irá definir como as sentenças visuais podem ser transformadas. Deste modo uma gramática é suficiente para especificar sem ambigüidades uma linguagem visual.

A figura 2.1 mostra um diagrama com o modelo de uma linguagem visual bem como as entidades que podem especificá-la e o artefato que a manifesta.

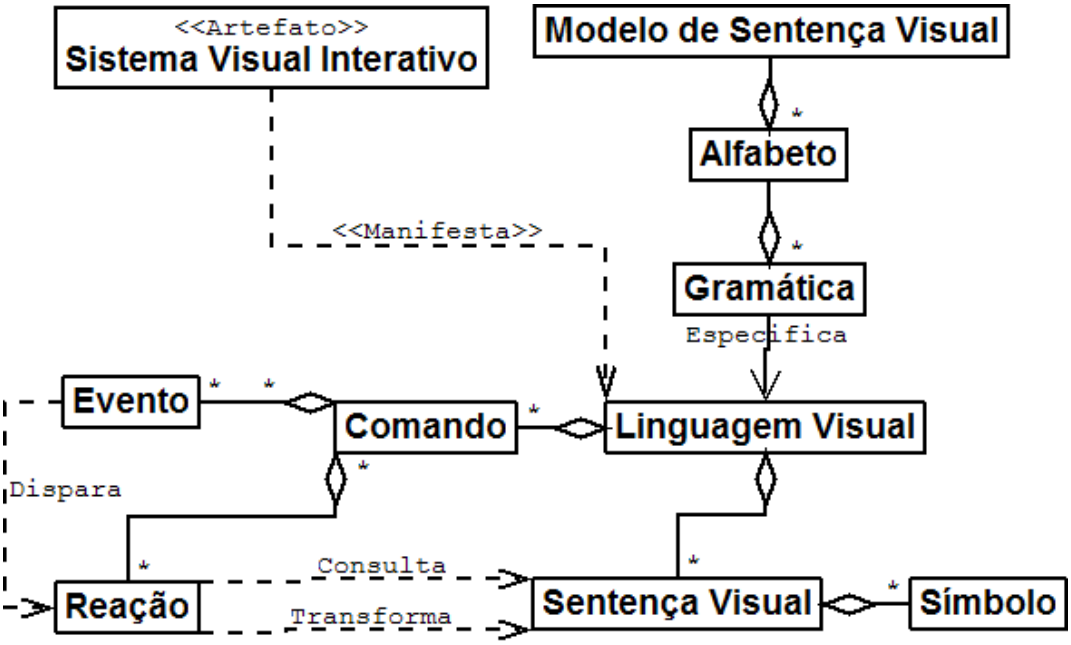


Figura 2.1: Modelo de linguagens visuais

É importante destacar que: toda gramática contém um e somente um alfabeto; todo alfabeto contém um e somente um modelo de sentença visual. Além disto: um modelo de sentença visual pode fazer parte de nenhum ou vários alfabetos; um alfabeto pode fazer parte de nenhuma ou várias gramáticas.

2.1.1 Verificação sintática em sentenças visuais

Verificações sintáticas consistem em verificar se a estrutura de uma sentença visual está de acordo com um modelo de sentença visual, um alfabeto e/ou uma gramática.

Em sistemas de linguagens visuais baseadas em gramáticas são utilizados três níveis de validação (COSTAGLIOLA, 2004):

1. Sentença validada por modelo de sentença visual: uma sentença visual que está de acordo com o modelo de sentença visual.
2. Sentença validada por alfabeto: uma sentença visual que está de acordo com o modelo de sentença visual e de acordo com um alfabeto.
3. Sentença validada por gramática: uma sentença visual que está de acordo com o modelo de sentença visual e de acordo com uma gramática.

Operações sobre sentenças validadas por gramáticas são importantes em sistemas baseados em gramáticas. Porém muitas operações podem ser realizadas em sentenças visuais que precisam estar apenas em um nível intermediário de validação. Por isso sentenças validadas por um alfabeto ou pelo modelo de sentença visual são importantes opções quando um alfabeto/gramática ou os artefatos que suportam este alfabeto/gramática não estão disponíveis ou não são desejados.

Portanto, os níveis intermediários de validação dão mais liberdade e flexibilidade ao processo de edição de sentenças visuais e alfabetos.

2.1.1.1 Validação em múltiplos espaços de nomes

Outro recurso de validação importante é o uso de múltiplos espaços de nomes. Este não é um recurso suportado por todos modelos de sentença visual, mas modelos importantes como o GXL (HOLT, 2006) suportam o gerenciamento de múltiplos espaços de nomes. Este recurso proporciona mais flexibilidade e modularidade na edição e validação de alfabetos e sentenças visuais.

Quando os modelos de sentença visual permitem o uso de múltiplos espaços de nomes, pode-se dividir a estrutura de dados de uma sentença visual em partes. Onde cada parte do documento poderá ser validada de uma maneira diferente. Uma parte pode ser validada por um alfabeto "A", outra parte por um alfabeto "B" e uma terceira parte pode ser validada apenas pelo modelo de sentença visual. Como se pôde notar, cada parte do documento precisa estar de acordo com um espaço de nomes e cada espaço de nomes é definido em um alfabeto. Portanto vários alfabetos podem ser combinados para definirem um conjunto de regras que validam a estrutura de dados de uma sentença visual.

A estrutura que define como alfabetos podem ser combinados varia de modelo para modelo. Esta tese apresenta uma estrutura compatível com o modelo GXL e com o modelo de linguagem visual apresentado na figura 2.1. Tal estrutura é mostrada na figura 2.2.

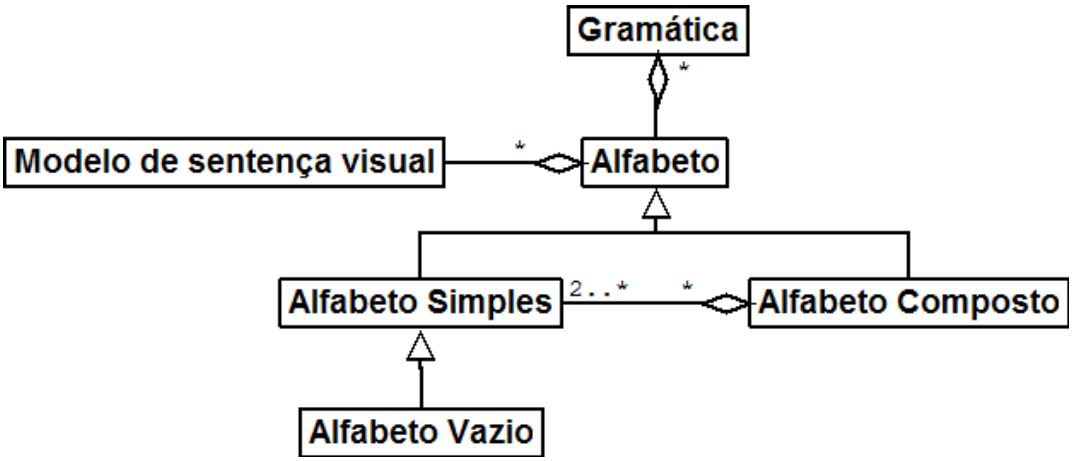


Figura 2.2: Alfabetos com suporte a múltiplos espaços de nomes

Como já foi dito, um **alfabeto** é um conjunto de regras que define os tipos válidos de uma classe de sentenças visuais. Um **alfabeto simples** tem um único espaço de nomes e não pode ser decomposto em sub-alfabetos. Quando uma sentença visual informa que está de acordo com um alfabeto simples, todas as entidades da sentença precisam estar de acordo com os tipos definidos no alfabeto simples.

O **alfabeto vazio** é um alfabeto simples que não acrescenta nenhuma regra além das regras que determinam que um documento seja validado por um modelo de sentença visual. Quando uma sentença visual informa que está de acordo com um alfabeto vazio, todas as entidades da sentença precisam ser validadas por um modelo de sentença visual.

Quando uma sentença visual contém mais de um espaço de nomes, ela precisa estar dividida em partes e deverá informar o **alfabeto composto** que definirá as regras gerais da sentença. Adicionalmente cada parte da sentença deverá informar o alfabeto simples que conterá as regras específicas desta parte da sentença.

Um alfabeto composto basicamente especifica que a estrutura de dados de uma sentença visual deverá ser dividida em diversas partes e que cada parte deverá ser despachada para o seu correspondente alfabetos simples. Cada alfabeto simples irá validar a sua correspondente parte da sentença visual sem ter nenhuma informação sobre o restante da sentença. Antes de despachar as partes de uma sentença visual o alfabeto composto verifica algumas condições. Se estas condições forem atendidas o despache pode ser feito. Ao final de um processo de validação, se todas as partes da sentença visual forem despachadas e validadas a sentença como um todo será válida.

Tanto a validação das condições para o despache quanto as validações feitas nos alfabetos simples são verificações de tipos. A diferença é que a validação das condições para o despache faz uma verificação de tipos de dados numa granularidade maior (como já foi dito a própria sentença visual é uma entidade tipada). Já as validações feitas nos alfabetos simples são validações mais refinadas, i.e. numa granularidade menor.

A estrutura da figura 2.2 evidencia que independentemente da composição do alfabeto, uma gramática continua tendo um e somente um alfabeto. Ou seja, o alfabeto continua sendo um conjunto de tipos de símbolos, embora estes tipos estejam definidos em espaços de nomes diferentes. Adicionalmente, um alfabeto continua tendo um e somente um modelo de sentença visual. Estas duas condições mantêm a compatibilidade desta estrutura com o modelo de linguagem visual já apresentado.

2.2 Ambientes de execução e desenvolvimento de sistemas visuais interativos

Em sistemas que utilizam gramáticas, as gramáticas são especificadas em um ambiente de desenvolvimento e os sistemas visuais interativos descritos nestas gramáticas são executados em ambientes de execução (BARDOHL, 1999; COSTAGLIOLA, 2004).

No **ambiente de desenvolvimento,** um sistema visual interativo é especificado. Nestes ambientes um desenvolvedor especifica o alfabeto e/ou a gramática que descreve um sistema visual interativo. O modelo de sentença visual não é editado pelo desenvolvedor, pois este modelo é uma característica fixa e não configurável do ambiente. Este ambiente também é chamado de ambiente de desenvolvimento baseado em gramática (ADBG).

Já o **ambiente de execução** executa o sistema visual interativo que foi especificado. Nestes ambientes o usuário final pode interagir com o sistema visual interativo projetado.

A figura 2.3 mostra duas alternativas para o relacionamento entre estes ambientes. A figura 2.3.a mostra o caso onde o ambiente de desenvolvimento gera uma especificação que é interpretada e/ou executada pelo ambiente de execução. Esta especificação pode ser uma gramática ou qualquer outro artefato que descreva uma linguagem visual e que possa ser interpretado pelo ambiente de execução. Neste caso é dito que o ambiente de execução transforma-se num sistema visual interativo. Já a figura 2.3.b mostra o caso onde o ambiente de desenvolvimento gera o ambiente de

execução. Neste caso é gerado um ambiente de execução para cada linguagem visual especificada. Nesta situação é dito que o ambiente de execução é o sistema visual interativo.

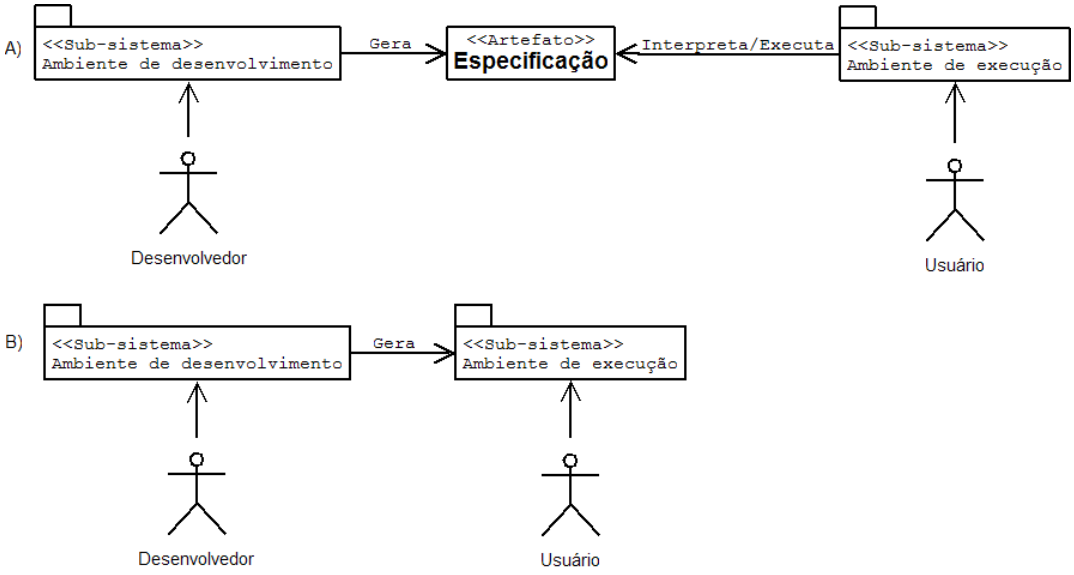


Figura 2.3: Sistemas automatizados que utilizam gramáticas

O que mais caracteriza um sistema que utiliza gramáticas é o seu ambiente de desenvolvimento, pois, é no ambiente de desenvolvimento que um desenvolvedor utiliza e cria modelos de sentença visual, alfabetos e gramáticas. Um ambiente de execução ou o que ele executa é apenas o resultado final do desenvolvimento de um sistema visual interativo, resultados similares podem ser obtidos por outras formas de desenvolvimento.

Para o escopo desta tese não será necessário maiores detalhes sobre o ambiente de execução. Basta saber que ele é um sistema que interpreta e/ou executa a especificação de uma linguagem visual, fornecendo ao usuário um sistema visual interativo. Já os ambientes de desenvolvimento serão detalhados na próxima seção.

2.3 Ambientes de desenvolvimento baseados em gramática (ADBG)

Esta seção apresenta os principais subsistemas e operações de ambientes de desenvolvimento baseados em gramática (ADBGs). A definição dos referidos subsistemas/operações foram retiradas de trabalhos que analisam e classificam ADBGs e suas características (BARDOHL,1999; BOTTONI, 2003; COSTAGLIOLA, 2004; FERRI, 2007). Para comprovar a composição dos subsistemas e operações dos ADBGs também foram consultados artigos científicos sobre ADBGs específicos (ASCHENBRENNER, 2003; BOTTONI, 2004; COSTAGLIOLA, 2005; DREWES, 2004; ERMEL, 2004; ERMEL, 2007; FURNAS, 2003; GUERRA, 2007; MINAS, 2004; REPENNING, 1995; SMITH, 2000; ZHU, 2007).

2.3.1 Subsistemas de ADBGs

Em síntese os ADBGs são formados por um, dois ou três dos seguintes subsistemas: editor de sentença visual, editor de alfabeto e editor de gramática.

Em um **editor de sentença visual** o desenvolvedor realiza operações que utilizam as entidades de um modelo de sentença visual. Todo editor de sentença visual contém um **modelo de sentença visual**. Este modelo é uma entidade fixa do editor, vem previamente implementada e não pode ser alterada pelo desenvolvedor. O modelo de sentença visual contém um conjunto de operações. Tal conjunto especifica operações que utilizam as entidades do próprio modelo de sentença visual.

Típicas operações que utilizam entidades do modelo de sentença visual são: adicionar entidades em uma sentença visual, editar entidades de uma sentença visual, visualizar entidades, pesquisar entidades, verificar se uma sentença visual ou uma entidade está de acordo com o modelo de sentença visual etc.

Em um **editor de alfabeto** o desenvolvedor realiza operações que utilizam as entidades de um alfabeto. Adicionalmente o editor permite que o desenvolvedor realize meta-operações em alfabetos, ou seja, operações que permitem a criação e edição de alfabetos.

Um editor de alfabeto contém um **meta-modelo de alfabeto** que por sua vez contém um conjunto de operações controladas pelo meta-modelo de alfabetos. O meta-modelo de alfabeto define a estrutura dos alfabetos, ou seja, as entidades de um alfabeto e seus relacionamentos. As suas operações permitem que o desenvolvedor utilize as entidades do meta-modelo de alfabeto. Típicas operações do meta-modelo de alfabeto são: criar/excluir/editar um alfabeto, criar/excluir/editar um tipo, definir como um tipo é visualizado, etc.

Um alfabeto e as operações que podem utilizar este alfabeto podem ser geradas e especificadas pelo desenvolvedor através das operações do meta-modelo de alfabeto. Todo **alfabeto** gerado contém um conjunto de operações. Tal conjunto especifica operações que utilizam o próprio alfabeto. Típicas operações que utilizam um alfabeto são: adicionar uma entidade tipada numa sentença visual, editar entidades tipadas de uma sentença visual, visualizar uma entidade tipada etc.

O meta-modelo de alfabeto é uma entidade fixa do editor, vem previamente implementada e não pode ser alterada pelo desenvolvedor. Já a entidade alfabeto pode ser criada, editada e utilizada pelo desenvolvedor.

Em um **editor de gramática** o desenvolvedor realiza operações que utilizam uma gramática. Adicionalmente o editor permite que o desenvolvedor realize meta-operações em gramáticas, ou seja, operações que permitem a criação e edição de gramáticas.

Um editor de gramática contém um **meta-modelo de gramática** que por sua vez contém um conjunto de operações controladas pelo meta-modelo de gramática. O meta-modelo de gramática define a estrutura das gramáticas, ou seja, as entidades de uma gramática (uma sentença visual inicial, um alfabeto, um conjunto de regras etc) e seus relacionamentos. As sua operações permitem que o usuário utilize as entidades do meta-modelo de gramática. Típicas operações do meta-modelo de gramáticas são: criar/excluir/editar uma gramática, criar/excluir/editar uma sentença inicial ou uma regra de produção, etc.

Uma gramática e as operações que podem utilizar esta gramática podem ser geradas e especificadas pelo desenvolvedor através das operações do meta-modelo de gramática. Toda **gramática** gerada contém um conjunto de operações. Tal conjunto especifica operações que utilizam a própria gramática. Típicas operações que utilizam uma gramática são: verificar se uma sentença visual pertence a uma gramática, gerar um

ambiente de execução que possa executar a linguagem visual definida na gramática, gerar uma especificação de linguagem visual em um formato diferente do formato da gramática, simular o funcionamento e a aparência do sistema visual interativo que está descrito na gramática etc.

O meta-modelo de gramática é uma entidade fixa do editor, vem previamente implementada e não pode ser alterada pelo desenvolvedor. Já a entidade gramática pode ser criada, editada e utilizada pelo desenvolvedor de sistemas visuais interativos.

A figura 2.4 mostra um diagrama de classes de um ambiente de desenvolvimento.

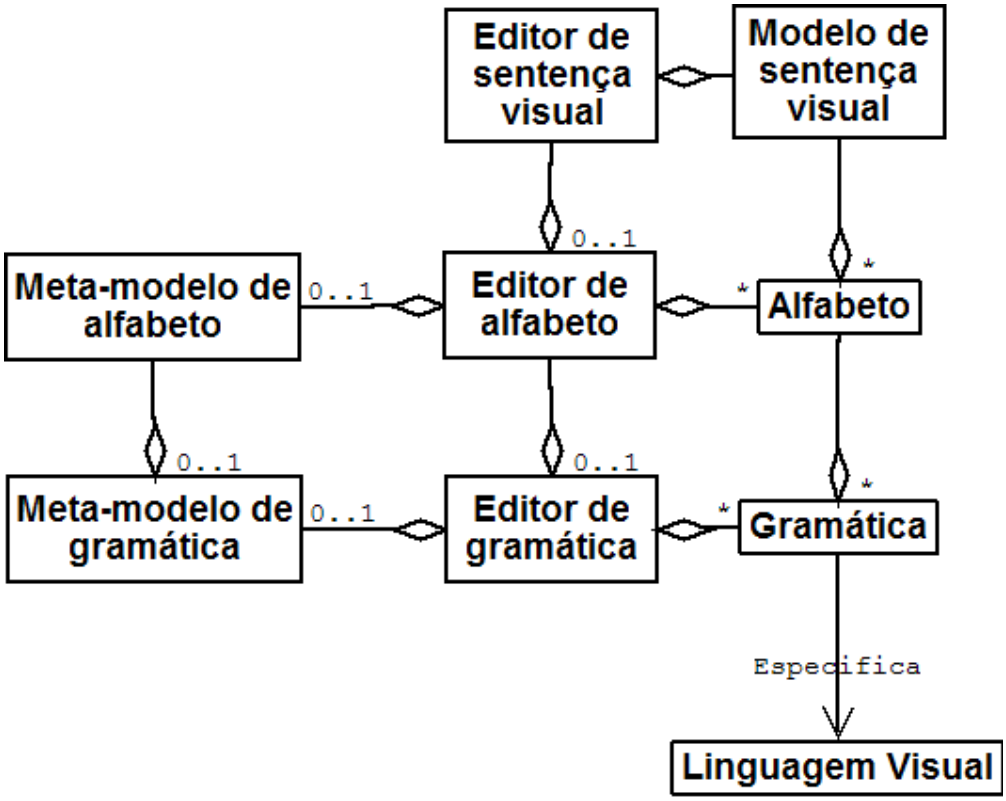


Figura 2.4: Sistemas e modelos de dados de um ADBG

É importante observar que todo editor de gramática contém um editor de alfabeto e todo editor de alfabeto contém um editor de sentença visual. Porém nem todo editor de sentença visual está contido em um editor de alfabeto e nem todo editor de alfabeto está contido em um editor de gramática.

Também se pode notar que nem todo editor de gramática possui um meta-modelo de gramática e que nem todo editor de alfabeto possui um meta-modelo de alfabeto. Estas diferentes cardinalidades evidenciam que podem existir diferentes especializações do modelo de ambientes de desenvolvimento apresentado. A próxima subseção apresenta algumas destas especializações.

2.3.1.1 Variações do ambiente de desenvolvimento

A seção anterior apresentou um modelo de ambientes de desenvolvimento. Esta seção discute como este modelo pode ser especializado para descrever ambientes de desenvolvimento com diferentes características.

A especialização de ambiente de desenvolvimento mais completa é o editor de gramática completo. Ele possui todas as entidades mostradas na figura 2.4. A figura 2.5 mostra o diagrama de classes de um editor de gramática completo.

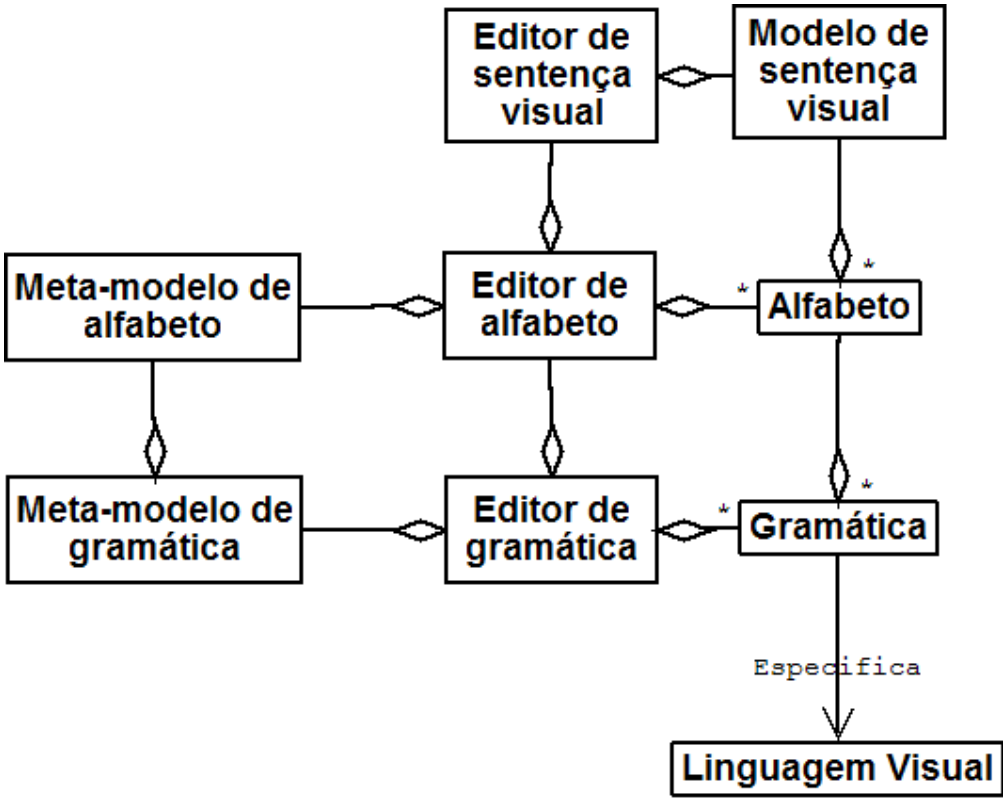


Figura 2.5: Editor de gramática completo

Com um editor de gramática completo o desenvolvedor pode criar e editar gramáticas. Posteriormente pode-se utilizar cada gramática criada e, por exemplo, gerar ambientes de execução. Como este editor também contém um editor de alfabetos, o desenvolvedor pode criar e editar alfabetos. Subsequentemente ele pode utilizar os alfabetos criados para adicionar entidades tipadas em uma sentença visual entre outras operações. O editor de alfabetos possui um editor de sentenças visuais. Por isso o desenvolvedor também pode utilizar operações do modelo de sentenças visuais como, por exemplo, editar uma sentença visual sem os recursos associados aos tipos definidos em alfabetos, verificar se uma sentença visual esta de acordo com o modelo de sentença visual, etc.

Uma pequena variação do editor de gramática completo é um editor de gramática que não possui um meta-modelo de alfabeto. Este tipo de editor possui todas as funcionalidades do editor de gramática completo exceto as operações controladas pelo meta-modelo do alfabeto. Este tipo de editor tem um alfabeto fixo e disponibiliza operações que podem utilizar este alfabeto. Consequentemente o editor de gramáticas só pode criar gramáticas que utilizam o alfabeto previamente fixado. Na seção 6 será apresentado um ambiente de desenvolvimento com estas características.

Outra especialização do ambiente de desenvolvimento é o editor de alfabeto sem o editor de gramática. Este é um ambiente de desenvolvimento que contém apenas um editor de alfabeto e um editor de sentença visual. Por isso ele pode ser utilizado para a

edição de sentenças visuais e alfabetos, mas não pode ser utilizado para edição de gramáticas. A figura 2.6 mostra um diagrama de classes deste tipo de editor. Há ainda o editor de sentença visual sem o editor de alfabeto nem o editor de gramática. Tal editor só pode ser utilizado para se manipular sentenças visuais.

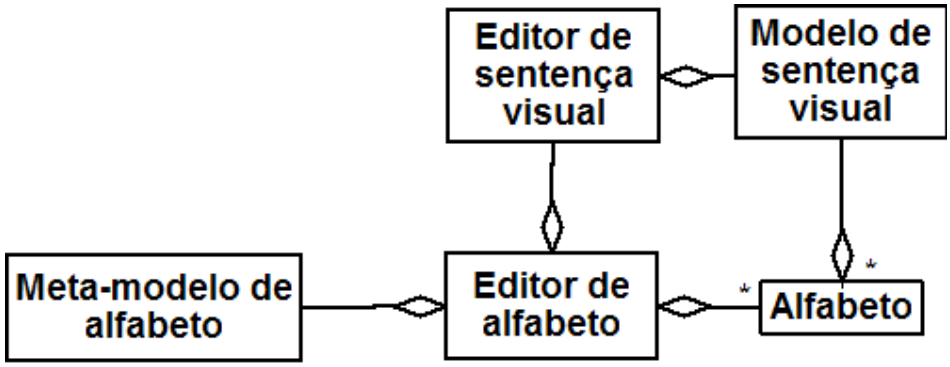


Figura 2.6: Editor de alfabeto

Até este ponto foi discutido variações do modelo de ambientes de desenvolvimento que podiam conter ou não conjuntos inteiros de operações e modelos. Mas as variações de ambientes de desenvolvimento também ocorrem em granularidades mais finas. Dois editores de gramática completos podem variar por conterem operações diferentes (um permite a validação de alfabetos outro não, um gera ambientes de execução outro apenas exporta especificações, etc). Outra variação importante são as variações decorrentes da escolha de diferentes modelos de sentença visual, meta-modelos de alfabeto e meta-modelos de gramáticas.

2.3.2 Operações de ADBGs

Até este ponto foram descritos em detalhes os modelos e sistemas de um ADBG. Nesta seção serão dados maiores detalhes sobre as operações que podem ser feitas nestes ambientes.

As operações de um ADBG são essencialmente operações de edição. Operações de edição mostram para um desenvolvedor uma visão dos dados que estão em uma sentença visual, em um alfabeto ou em uma gramática. Muitas destas visões são acompanhadas de operações de interação entre o desenvolvedor e o sistema. Tais interações permitem que o desenvolvedor analise e altere os dados de uma sentença visual, alfabeto ou gramática.

Muitas operações são utilizadas nestas visões e em suas interações. Para organizá-las e categorizá-las esta tese aplicou os critérios de categorização de operações em sistemas de edição de Quint e Vatton (2004).

Quint et al. (2004) classificaram as operações de edição de documentos XML conforme o modelo de dados que controla as operações. Cada conjunto de operações que é controlado por um mesmo modelo é chamado de modo de edição.

Um modo de edição mostra ao desenvolvedor uma visão da estrutura de dados de um documento destacando as entidades do seu modelo. Cada entidade do modelo encontrada no documento (i.e. instância da entidade) é tratada como uma unidade lógica que pode ser excluída, inserida ou alterada desde que estas alterações transformem o referido documento em um novo documento que também esteja de acordo com o modelo de dados.

Cada modo de edição pode conter dois tipos de visões: visões sintáticas e visões semânticas. As visões sintáticas mostram representações da estrutura de dados de uma entidade. Já as visões semânticas mostram o significado, a interpretação, a semântica das entidades. No caso das sentenças visuais, a interpretação da entidade *símbolo* é a de um gráfico na tela, a interpretação de uma *sentença visual* é a de uma tela com vários gráficos e assim por diante. Para uma melhor compreensão pode-se dizer que visões semânticas de sentenças visuais equivalem a interfaces do tipo WYSIWYG (*What You See Is What You Get*) (Quint, 2007) de editores de texto. Nestas interfaces a aparência do que está sendo editado é similar ou igual à aparência do produto final, ou seja, é igual a sua interpretação.

Quint especificou que a visão sintática controlada por um modelo é um modo de edição. Já a visão semântica controlada pelo mesmo modelo é outro modo de edição. Para evitar uma grande quantidade de modos de edição e para agrupar visões controladas pelo mesmo modelo a proposta desta tese agrupou visões sintáticas e semânticas de um mesmo modelo em um único modo de edição (mais detalhes sobre o trabalho de Quint et. al. estão relatados na seção 4 desta tese).

Estes critérios ainda não haviam sido aplicados para classificar as operações de ADBGs. Portanto a aplicação dos critérios de Quint et. al. (2004) para classificar as operações de ADBGs é uma contribuição desta tese.

Como já foi mostrado, um ADBG pode conter 5 modelos: modelo de sentença visual, modelo de alfabeto, modelo de gramática, meta-modelo de alfabeto e meta-modelo de gramática. Um modo de edição pode ser associado a cada modelo. Um modo adicional de edição, chamado de modo de edição de entidades básicas, está associado a operações que não são controladas por nenhum destes modelos.

A seguir os seis modos de edição de ADBGs e suas visões serão detalhados.

2.3.2.1 Edição de entidades básicas

No seu nível de abstração mais baixo, uma sentença visual, um alfabeto ou uma gramática são armazenados como uma seqüência de caracteres, um conjunto de objetos, um conjunto de elementos ou qualquer outro conjunto de entidades básicas. Um editor de entidades básicas mostra ao desenvolvedor uma sentença visual, alfabeto ou gramática como um simples conjunto de caracteres, objetos ou outro tipo de entidade básica.

Quando os dados de um ADBG são tratados como um conjunto de caracteres, por exemplo, o desenvolvedor pode interagir livremente com estes caracteres inserindo e excluindo caracteres conforme as suas necessidades. Este modo de edição da uma grande liberdade de edição ao desenvolvedor e normalmente suas operações são mais simples que as operações dos outros modos. O preço a pagar por esta simplicidade e liberdade é que o desenvolvedor é o responsável por manter a coerência sintática e semântica da sentença visual, alfabeto ou gramática. Há poucas operações que ajudam o desenvolvedor a não cometer erros e poucas operações que agilizam edições mais complexas.

Analogamente, quando os dados de um ambiente são tratados como um conjunto de objetos, elementos ou qualquer outro tipo de entidade básica também se pode editar os dados do ambiente. Entretanto sentenças visuais, alfabetos e gramáticas não são tratados ou visualizados como sentenças visuais, alfabetos ou gramáticas. Os dados são tratados

como um conjunto de entidades básicas (em um nível de abstração mais baixo). Consequentemente o desenvolvedor tem mais liberdade de edição e é o responsável por manter a coerência sintática e semântica dos dados. Todas as visões deste modo de edição são visões sintáticas, pois só mostram representações dos dados e não o seu significado.

Os editores de entidades básicas fazem parte do editor de sentença visual, embora tenham sido omitidos do modelo de ADBG mostrado na figura 2.4,

2.3.2.2 Edição controlada por modelo de sentença visual

Num nível de abstração mais alto, uma sentença visual é um conjunto de entidades que está de acordo com um modelo de sentença visual. Este modo de edição mostra ao desenvolvedor a estrutura de dados de uma sentença visual destacando as entidades do modelo que estão nesta sentença visual. Cada entidade do modelo de sentença visual encontrada na sentença visual é tratada como uma unidade lógica. Esta unidade lógica pode ser excluída, inserida ou alterada desde que estas alterações transformem a sentença visual em uma nova sentença que também esteja de acordo com o modelo de sentença visual.

Visões comuns deste modo de edição mostram uma sentença visual como um grafo, como uma árvore, como um conjunto de caracteres onde as entidades são diferenciadas por cores e/ou endentações, etc. Neste modo de edição as operações precisam estar de acordo com o modelo de sentença visual e suas regras sintáticas. Tal fato torna estas operações mais complexas que as operações do modo de edição de entidades básicas. Porém esta complexidade pode ser compensada por um número maior de recursos que podem ajudar o desenvolvedor a editar/visualizar sentenças visuais e a evitar a ocorrência de erros relacionados à sintaxe do modelo de sentenças visuais.

Todas as visões deste modo de edição são visões sintáticas pois só mostram representações dos dados e não o seu significado. Este modo de edição pode ser encapsulado em um editor de sentença visual.

2.3.2.3 Edição controlada por alfabeto

Neste nível de abstração, uma sentença visual é um conjunto de entidades tipadas que está de acordo com um alfabeto. Cada entidade tipada é tratada como uma unidade lógica que pode ser excluída, incluída ou alterada desde que estas alterações transformem a sentença visual em uma nova sentença que também esteja de acordo com o respectivo alfabeto.

Este modo de edição possui visões sintáticas semelhantes ao modo de edição controlada por modelo de sentença visual (grafos, árvores, texto, etc). Porém as operações deste modo de edição possuem recursos que facilitam a edição específica de entidades que estão de acordo com o respectivo alfabeto. Por exemplo, através de menus, caixas de diálogo e outros recursos estas operações permitem que somente entidades definidas no alfabeto sejam inseridas na sentença visual. Quando possível, estas operações mostram quais são as opções de nomes e valores que podem ser inseridas em cada parte de uma entidade.

Até este ponto foram descritas as interações das visões sintáticas deste modo de edição. Porém a partir deste modo de edição já é possível a disponibilização de visões semânticas. Quando uma entidade *símbolo* é tipada, já é possível associar uma aparência, um gráfico a este tipo de *símbolo*.

Assim nas visões semânticas deste modo de edição o desenvolvedor vê uma sentença visual como uma sentença visual, ou seja, uma tela com vários símbolos gráficos. Nestas visões normalmente é disponibilizado ao desenvolvedor uma paleta com os tipos de símbolos definidos no alfabeto. Através desta paleta o desenvolvedor pode inserir os símbolos permitidos na sentença que está editando. Tais visões também disponibilizam ao desenvolvedor os recursos gráficos para alterar propriedades gráficas dos símbolos como, por exemplo, redimensionar e reposicionar símbolos graficamente (sem acessar sua estrutura de dados). É importante lembrar que as visões semânticas também são controladas pelo alfabeto, por isso só é possível inserir símbolos definidos no alfabeto e alterar propriedades das entidades dentro do domínio de valores definidos no alfabeto.

Todas as operações e visões deste modo de edição podem ser encapsuladas em um editor de alfabeto.

2.3.2.4 Edição controlada por meta-modelo de alfabeto

Neste modo de edição os alfabetos são criados, visualizados e editados. O que equivale a dizer que tipos de entidades são criados, visualizados e editados.

Nas visões sintáticas deste modo de edição a estrutura de dados do alfabeto é visualizada por uma representação de dados como, por exemplo, grafos, árvores ou texto. Cada tipo de entidade do alfabeto é tratado como uma unidade lógica que pode ser criada, excluída e alterada desde que estas alterações transformem o alfabeto que está sendo editado em um novo alfabeto que também esteja de acordo com o meta-modelo de alfabeto.

Nas visões semânticas deste modo de interação o desenvolvedor pode ver e editar o gráfico que está associado à entidade tipada que está sendo criada. Opcionalmente algumas propriedades destas entidades tipadas também podem ser alteradas graficamente.

As operações e visões deste modo de edição também podem ser encapsuladas em um editor de alfabeto.

2.3.2.5 Edição controlada por gramática

O modo de edição controlada por gramática permite a edição de uma sentença visual guiada por uma gramática. Portanto somente sentenças visuais previstas na gramática podem ser editadas. Sentenças visuais não previstas na gramática são ditas inválidas quanto à gramática, para editá-las deve-se utilizar outro modo de edição. Adicionalmente em cada sentença visual que está sendo editada só podem ser aplicados os eventos previstos na gramática. Eventos não previstos devem ser ignorados. Quando os eventos previstos ocorrem, as reações previstas na gramática são disparadas.

Como se pode notar, neste modo de edição o funcionamento e a aparência do sistema visual interativo descrito em uma gramática são simulados. De fato, as mesmas operações que são usadas para executar um sistema visual interativo em um ambiente de execução podem ser usadas para simular o sistema visual interativo em um ambiente de desenvolvimento. Em ambientes de desenvolvimento estas operações servem para o desenvolvedor analisar ou testar o resultado final ou parcial do sistema que está sendo projetado.

Visões semânticas deste modo de edição simulam a aparência e o funcionamento do sistema visual interativo que foi projetado. O que um desenvolvedor pode ver e executar nestas visões deve ser a mesma coisa que um usuário verá e executará no sistema visual interativo projetado. Por isso muitas vezes o próprio ambiente de execução pode ser usado como uma visão semântica do modo de edição controlada por gramática.

Visões sintáticas deste modo de edição permitem a visualização das estruturas de dados de um sistema visual interativo durante a sua simulação. Permite que os eventos previstos na gramática sejam simulados através de alterações nos dados. Só podem ser simulados os eventos previstos na gramática. Pode-se mostrar o código fonte da sentença visual que está sendo mostrada ao usuário, uma tabela com o nome dos atuais atributos de um símbolo e seus respectivos valores ou qualquer outra forma de representar o atual estado da estrutura de dados do sistema visual interativo que está sendo simulado.

Além da visualização e simulação do sistema visual interativo projetado, o modo de edição controlada por gramática pode verificar se uma sentença visual pertence a uma gramática; pode exportar uma gramática para um formato externo; pode gerar um ambiente de execução, etc. Todas estas operações são controladas pela gramática.

Estas operações e visões podem estar encapsuladas em um editor de gramática. Eventualmente uma visão semântica controlada por gramática pode estar encapsulada a parte em um ambiente de execução.

2.3.2.6 Edição controlada por meta-modelo de gramática

Neste modo de edição as gramáticas são criadas, visualizadas e editadas. Visões sintáticas deste modo de edição variam muito de meta-modelo para meta-modelo. Muitas delas são muito semelhantes a visões de gramáticas tradicionais como, por exemplo, visões que descrevem gramáticas livre do contexto, gramáticas sensíveis ao contexto, gramáticas irrestritas através de linguagens textuais. Uma das linguagens textuais mais utilizadas para descrever gramáticas é a *extended Backus–Naur form* (EBNF). Outras visões sintáticas utilizam representações mais sofisticadas como, por exemplo, as gramáticas de grafos (BARDOHL, 1999), que mostram as regras de produção e toda a gramática através de grafos.

Visões semânticas da gramática como um todo são mostradas nas visões do modo de edição controlada por gramática. Já visões semânticas de partes isoladas da gramática podem ser mostradas no modo de edição controlada por meta-modelo de gramática. Porém, também variam muito de meta-modelo para meta-modelo. Gramáticas que definem sentenças visuais iniciais frequentemente tem visões semânticas destas sentenças (que é uma tela com gráficos). Algumas gramáticas permitem que suas regras de produção sejam expressas graficamente através de exemplos concretos de símbolos; este é o caso das regras de reescrita visual (BOTTONI, 2007). Muitos outros exemplos podem ser dados, porém uma descrição completa e extensiva das possíveis visões sintáticas e semânticas de gramáticas foge ao escopo e objetivos desta tese.

As operações deste modo de edição também podem ser encapsuladas em editores de gramáticas.

2.3.2.7 Considerações finais sobre modos de edição

Uma sentença visual, alfabeto ou gramática pode ter várias visões de cada um dos seis modos de edição. Cada visão deve mostrar diferentes aspectos ou partes de uma sentença visual/alfabeto/gramática e todas as visões devem estar sincronizadas.

Pode-se utilizar alternadamente as operações de diferentes modos de edição. Por exemplo, pode-se editar uma sentença visual no modo de edição de entidades básicas e após utilizar o verificador sintático do modo de edição controlado por modelo de sentença visual para verificar se as mudanças efetuadas estão de acordo com o modelo de sentença visual.

Devido ao recurso de múltiplos espaços de nomes, em uma mesma sentença visual pode-se aplicar operações que pertencem a diferentes modos de edição. Ou seja, uma parte da sentença pode estar de acordo com um alfabeto que esteja disponível para uso e outra pode pertencer a um alfabeto desconhecido ou indisponível. Assim em algumas partes da sentença visual podem ser aplicadas operações do modo de edição controlado por alfabeto e em outras partes da mesma sentença são aplicadas apenas operações do modo de edição controlado por modelo de sentença visual.

A primeira coluna da tabela 2.1 mostra os editores ADBG. A segunda coluna mostra os modos de edição e visões que estão encapsulados em cada editor. E por fim, a terceira coluna mostra o documento que pode ser editado em cada modo de edição. A tabela 2.1 bem como a descrição de cada modo de edição de ADBGs aqui apresentados são contribuições desta tese.

Tabela 2.1: Editores, modos de edição e documentos de ADBGs

Editores ADBG	**Modos de edição de ADBG**		**Documento**
Editor de sentença visual	Entidade básica		sentença visual, alfabeto, gramática
	Controlada por modelo de sentença visual		sentença visual
Editor de alfabeto	Controlada por alfabeto	Visão Sintática	sentença visual
		Visão Semântica	sentença visual
	Controlada por meta-modelo de alfabeto	Visão Sintática	alfabeto
		Visão Semântica	alfabeto
Editor de gramática	Controlada por gramática	Visão Sintática	sentença visual
		Visão Semântica	sentença visual
	Controlada por meta-modelo de gramática	Visão Sintática	gramática
		Visão Semântica	gramática

Em síntese, em um editor de sentença visual um desenvolvedor pode realizar as seguintes edições:

- Editar uma sentença visual, um alfabeto ou uma gramática através de uma visão do modo de edição de entidades básicas.

- Editar uma sentença visual através de uma visão do modo de edição controlada por modelo de sentença visual.

Através de um editor de alfabeto um desenvolvedor pode:

- Editar uma sentença visual através de uma visão do modo de edição controlada por alfabeto.
- Editar um alfabeto através de uma visão do modo de edição controlada por meta-modelo de alfabeto.

Por fim, através de um editor de gramática um desenvolvedor pode:

- Editar uma sentença visual através de uma visão do modo de edição de controlada por gramática.
- Editar uma gramática através de uma visão do modo de edição controlada por meta-modelo de gramática.

É importante notar que somente, sentenças visuais, alfabetos e gramáticas podem ser editadas em um ADBG. O modelo de sentença visual, o meta-modelo de alfabeto e o meta-modelo de gramática são características/modelos fixos de um ADBG e não podem ser alteradas pelo usuário do ADBG (o desenvolvedor de sistemas visuais interativos).

3 ADBGS EXISTENTES

O objetivo desta seção é mostrar exemplos concretos das visões, modos de edição e editores apresentados na seção anterior. Também pretende-se apresentar os ADBGs mais importantes, relevantes e representativos da área. Tais ADBGs foram escolhidos conforme as suas citações nos principais documentos e artigos científicos da área de ambientes de desenvolvimento baseados em gramáticas (BARDOHL,1999; BOTTONI, 2003; COSTAGLIOLA, 2004; FERRI, 2007).

O **Diagen** (*Diagram editor Generator*) (MINAS, 2004; BRIELER 2008) é um ADBG capaz de gerar editores gráficos a partir da especificação de uma gramática, mais precisamente a partir de uma gramática de hipergrafos. Um desenvolvedor especifica a gramática (incluindo o alfabeto da gramática) com a ajuda do ADBG. O ADBG provê os editores de entidades básicas, de sentença visual, de alfabeto e de gramática integrados em um único editor, o VisualDiagen. Após o desenvolvedor especificar a gramática o Diagen gera o editor gráfico projetado.

Na figura 3.1 pode-se ver telas do VisualDiagen. Na esquerda das figuras 3.1 (a) e 3.1 (b) pode-se ver uma visão sintática de toda a gramática. Todos os dados da gramática são armazenados em uma estrutura hierárquica e esta estrutura pode ser visualizada nesta visão. Esta é uma visão sintática controlada por meta-modelo de gramática. Ela permite uma visualização dos dados de uma gramática, contudo estes dados não podem ser alterados através desta visão. Ao selecionar um item desta visão, uma segunda visão é mostrada a direita. A segunda visão mostra detalhes do item selecionado em um modo de edição apropriado para a edição do item selecionado.

Na figura 3.1 (a) foi selecionado um tipo de símbolo do alfabeto. Assim a direita foi chamada uma visão sintática controlada por meta-modelo de alfabeto que permite a edição do tipo de símbolo selecionado. Neste exemplo estão sendo definidos os atributos de um tipo de símbolo (cor de borda, cor interna, forma etc). Para alguns atributos são definidos valores fixos. Para outros é definido um domínio de valores.

A figura 3.1 (b) mostra uma visão semântica controlada por alfabeto. Esta visão mostra uma sentença visual onde símbolos de um alfabeto podem ser inseridos, excluídos e alterados. Nesta visão só são válidas as regras de um alfabeto, ela serve para o desenvolvedor visualizar ou testar os tipos de símbolos criados quando as regras de uma gramática ainda não estão disponíveis ou não são desejadas.

A figura 3.1 (c) mostra a esquerda uma visão sintática controlada por meta-modelo de gramática e a direita outra visão sintática controlada por meta-modelo de gramática.

Enquanto a visão da esquerda mostra uma visão de toda a gramática, a visão da direita mostra detalhes de uma regra de produção. É importante notar que os gráficos que aparecem nesta visão são uma representação dos dados e não a aparência dos dados em um ambiente de execução.

A figura 3.1 (d) mostra um sistema visual interativo gerado a partir de uma gramática do Diagen. O sistema gerado neste exemplo é um editor de redes de Petri. Este editor também pode ser considerado como uma visão semântica controlada por gramática.

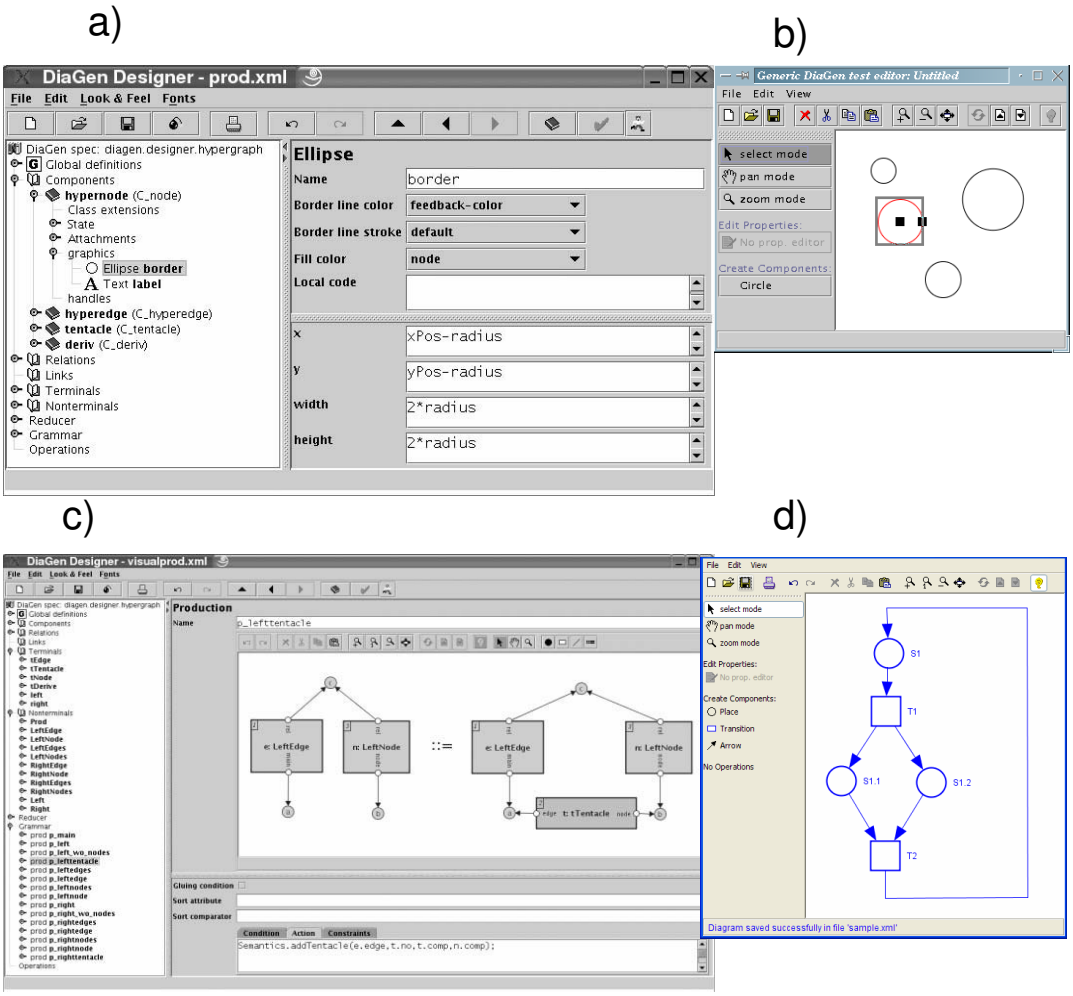


Figura 3.1: Editores do Diagen (Minas, 2004)

Uma sentença visual também pode ser editada levando-se em conta apenas o modelo de sentença visual (No caso do Diagen, uma sentença visual é representada como um hipergrafo). Um dos editores do Diagen permite a edição de sentenças visuais desta forma. A figura 3.2 mostra uma sentença visual através de uma visão sintática controlada por modelo de sentença visual do Diagen. Nesta visão o desenvolvedor pode alterar dados de uma sentença visual mesmo que a aparência da sentença visual, o seu alfabeto ou a sua gramática não estejam disponíveis ou não sejam desejados.

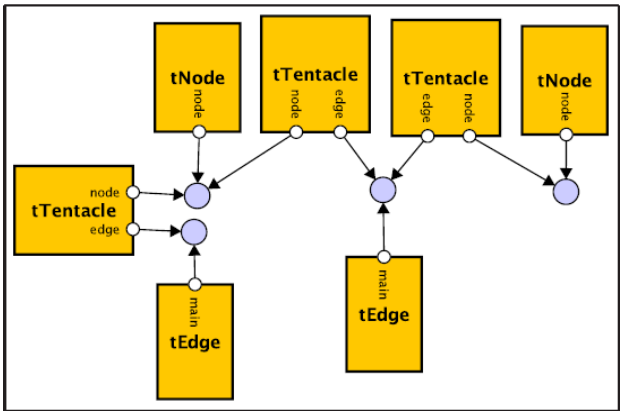


Figura 3.2: Visão sintática controlada por modelo de sentença visual do Diagen (MINAS, 2004)

É importante notar que nesta seção já foram mostradas três maneiras de se editar uma sentença visual no Diagen. A figura 3.2 mostrou o modo controlado por modelo de sentença visual. A figura 3.1 (b) mostrou o modo controlado por alfabeto. Finalmente a figura 3.1 (d) mostrou o modo controlado por gramática. Independentemente do modo como a sentença é editada ao final do processo de edição ela é armazenada em um documento XML.

De fato, todas as informações da gramática, alfabeto e sentenças visuais são armazenadas em um documento XML. O Diagen permite a edição do código fonte destes documentos. Pode-se editá-los conforme as regras de um editor XML ou como um texto comum. As duas alternativas são visões do modo de edição de entidades básicas.

Como foi demonstrado, o Diagen possui todos os 3 editores e os 6 modos de edição típicos de ADBGs.

O **VLDesk** (COSTAGLIOLA, 2005) é um ADBG que gera um sistema visual interativo a partir da especificação de uma gramática do tipo *eXtended Positional Grammar.*

A figura 3.3 mostra parte da arquitetura do VLDesk. O *symbol editor* é um editor de alfabeto onde os alfabetos são criados, editados e analisados conforme os modos de edição controlados por alfabeto e controlado por meta-modelo de alfabeto. É neste editor que são criados os símbolos terminais e não terminais utilizados nas gramáticas.

O *visual grammar editor* é um editor de gramática que auxilia a especificação e análise de *eXtended Positional Grammars* de acordo com os modos de edição controlados por gramática e por meta-modelo de gramática.

O *VPE generator* são as rotinas do VLDesk que irão gerar um sistema visual interativo a partir de uma *eXtended Positional Grammar.* O *visual programming environment* é o sistema visual interativo gerado.

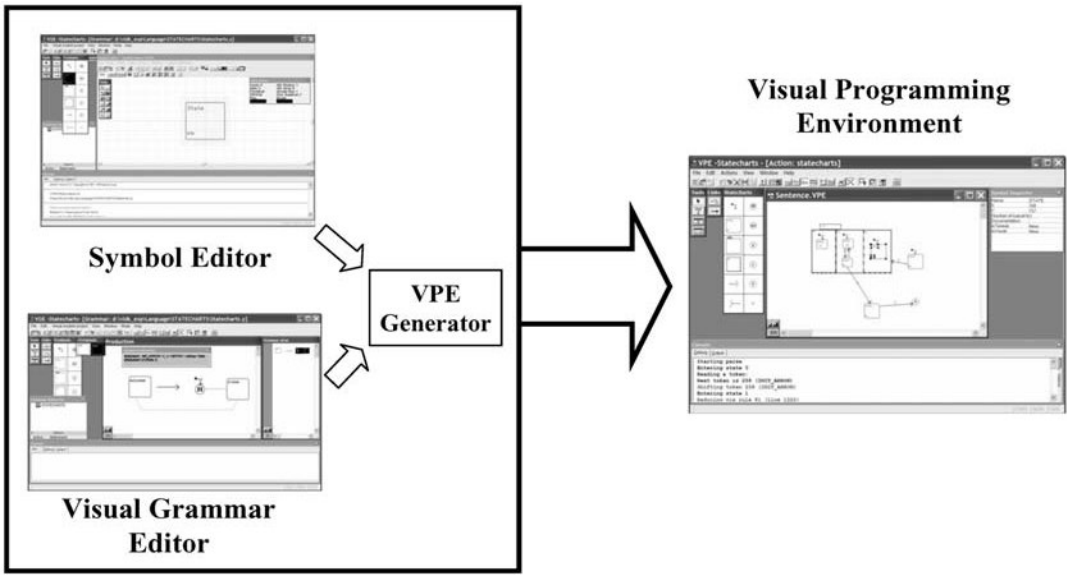


Figura 3.3: Arquitetura do VLDesk (COSTAGLIOLA 2005)

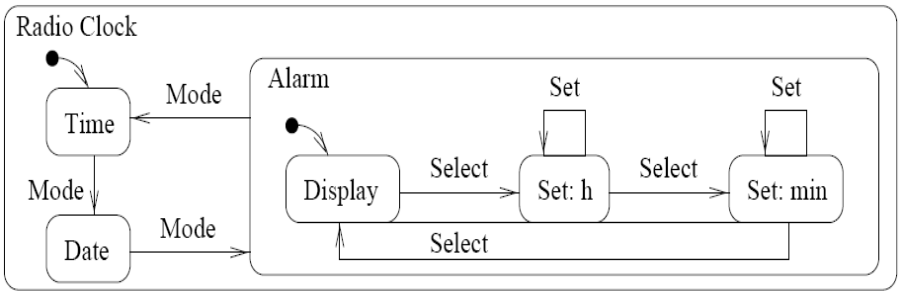
O modelo de sentença visual do VLDesk define as sentenças visuais como grafos. Logo as sentenças visuais são armazenadas como grafos. O VLDesk não disponibiliza uma visão dos dados que mostre a estrutura de grafos de uma sentença visual. Muitos ADBGs não disponibilizam este tipo de visão quando suas gramáticas não utilizam sentenças visuais em suas regras. Entretanto o VLDesk pode armazenar as sentenças visuais no formato GXL (HOLT, 2006). Existem muitos editores GXL que mostram documentos GXL como grafos. Portanto o modo de edição controlado por modelo de sentença visual pode ser feito através de editores GXL que neste caso seriam ferramentas externas (que não pertencem ao VLDesk).

As sentenças visuais, alfabetos e gramáticas podem ser armazenadas em arquivos GXL. Tais arquivos podem ser editados por editores XML ou editores de texto. Este tipo de edição corresponde ao modo de edição de entidades básicas. Portanto, tal modo de edição também pode ser feito utilizando-se ferramentas externas.

Os ADBGs **Genged** (ERMEL, 2004), **Tiger** (ERMEL, 2007) e **Atom3** (GUERRA, 2007; LARA, 2004) têm arquiteturas semelhantes ao Diagen e ao VLDesk. Ou seja, possuem editores de sentença visual (externos em alguns casos), editores de alfabeto e editores de gramática. Tais editores auxiliam a especificação de uma gramática que posteriormente é utilizada para gerar um sistema visual interativo.

A figura 3.4 (a) mostra uma sentença visual. A figura 3.4 (b) mostra uma visão controlada por modelo de sentença visual. Esta é uma visão do editor de sentença visual do Genged que descreve a sentença da figura 3.4 (a). O modelo de sentença visual do Genged descreve uma sentença visual como um conjunto de grafos com atributos.

a)



b)

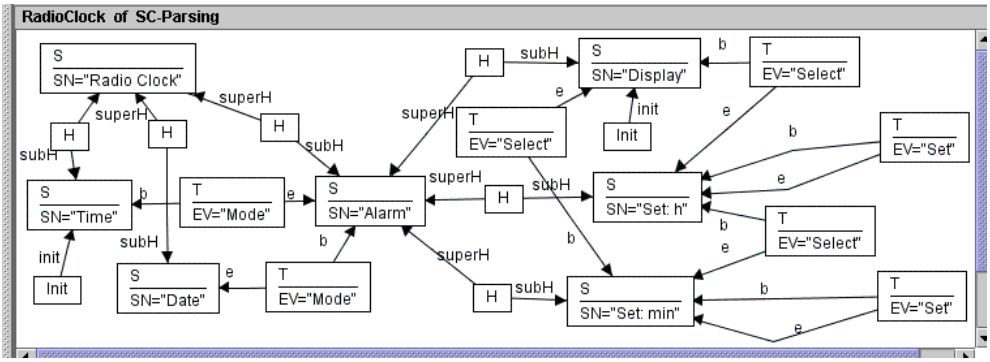


Figura 3.4: Sentença visual (a) e correspondente visão controlada por modelo de sentença visual do Genged (b) (BARDOHL, 2003)

A figura 3.5 mostra uma visão semântica controlada por meta-modelo de alfabeto de um editor de alfabeto do Atom3. Nesta figura um tipo de símbolo está sendo definido.

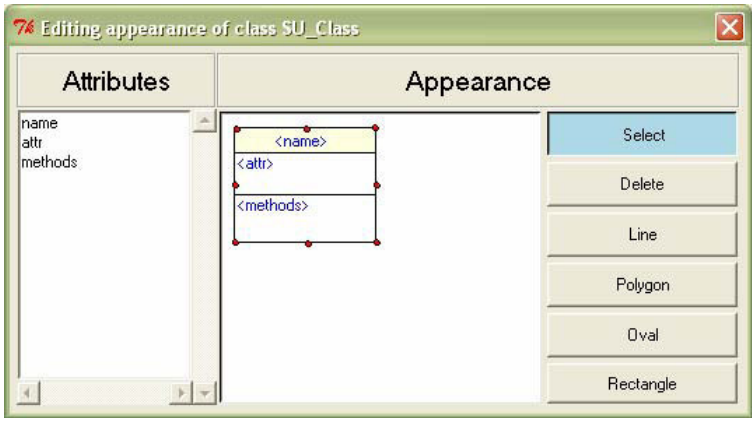


Figura 3.5: Visão semântica controlada por meta-modelo de alfabeto do Atom3 (GUERRA, 2007)

Já no centro da figura 3.6 é mostrada uma visão semântica controlada por meta-modelo de gramática do Tiger. Nesta visão as regras de uma gramática são formadas por símbolos que tem a aparência dos símbolos que irão aparecer no ambiente de execução do sistema gerado. As demais visões da figura 3.6 são visões sintáticas controlas por meta-modelo de gramática do editor de gramática do Tiger.

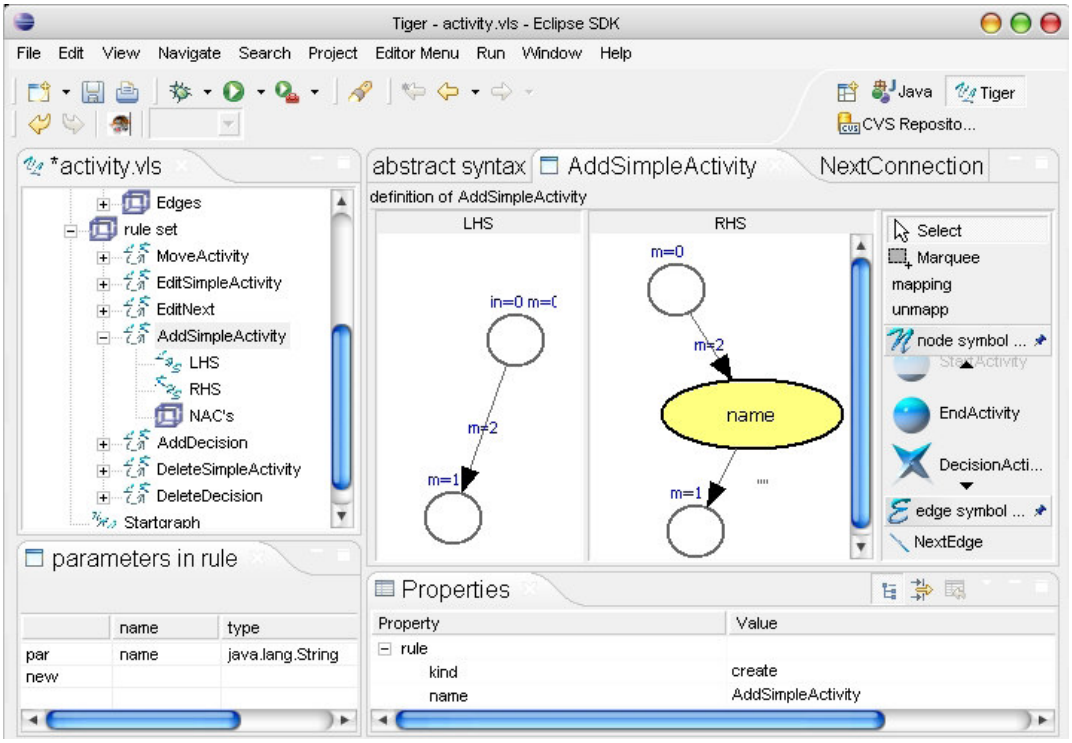


Figura 3.6: Editor de gramática do Tiger (TAENTZER,2006)

O VLDesk, Tiger, Genged e Atom3 possuem os 3 editores e os 6 modos de edição típicos de ADBGs.

O ADBG **Genial** (BOTTONI, 2004) também tem uma arquitetura semelhante ao Diagen, porém o seu editor de gramática ainda não tem uma visão semântica. Bottoni et al. (2004) demonstrou que é possível traduzir as regras de uma gramática VCARW para o formato interno do Genial. Entretanto a visão que traduz e auxilia a edição de regras VCARW ainda não foi implementada. Por enquanto estas regras são informadas através de texto em um formato apropriado para facilitar a edição de gramáticas. Tal visão é uma visão sintática controlada por meta-modelo de gramática.

Especificar gramáticas mais complexas não é uma atividade trivial, por isso em alguns ADBGs o desenvolvedor precisa especificar um meta-modelo antes de especificar uma gramática. O meta-modelo é uma especificação UML ou ER que define algumas características do sistema que está sendo projetado. Estas especificações são então usadas como parâmetros dos editores do ADBG. Assim um "esqueleto" ou "template" da gramática é disponibilizado ao desenvolvedor que terá então o trabalho de especificar as partes da gramática que estão faltando. Quando a especificação da gramática é terminada o ADBG gera o sistema projetado a partir da gramática. O VLDesk e o Atom3 são ADBGs com estas características. Estes ADBGs combinam recursos de ambientes baseados em gramáticas com recursos baseados em meta-modelos.

Em alguns ADBGs crianças e usuários sem conhecimento em programação podem especificar alfabetos e gramáticas, gerando, desta maneira, seus próprios jogos, simuladores e animações. Exemplos de ADBGs deste tipo são: o **Agentsheets** (REPENNING, 1995), o **Stagecast Creator** (antigo KidSim)(SMITH, 2000) e o **BitPict** (FURNAS, 2003). O ponto mais importante nestes ADBGs é proporcionar visões

semânticas adequadas para o seu público alvo. Devido a sua simplicidade estes ADBGs tem uma expressividade menor que os ADBGs relatados anteriormente. Porém, apesar de sua simplicidade estes ADBGs possuem editores de sentença visual, de alfabeto e de gramática.

Os editores de sentença visual dos três ADBGs não possuem o modo de edição de entidades básicas, pois o seu público alvo não tem a habilidade ou o interesse em editar códigos fonte de linguagens de programação. O editor de alfabeto do BitPict não possui visões controladas por meta-modelo de alfabeto. Por isso o alfabeto do BitPict é fixo. Pode-se especificar várias gramáticas com o BitPict, porém em todas gramáticas é usado o mesmo alfabeto. O editor de gramática do BitPict possui visões dos modos de edição controlado por gramática e controlado por meta-modelo de gramática.

Os editores de alfabeto e de gramática do Stagecast creator e Agentsheets possuem visões dos 4 modos de edição típicos destes editores.

Na figura 3.7 é mostrada, bem à esquerda, uma animação gerada no Stagecast Creator, ao lado é mostrado um editor de alfabeto onde os personagens da animação são criados (desenhados). Bem a direita é mostrado um editor de gramática onde as regras de uma gramática especificam os movimentos dos personagens na animação/simulação.

Figura 3.7: Editores do Stagecast Creator (STAGECAST, 2007)

Como pôde ser visto os ADBGs podem ser desde ambientes complexos onde programadores e analistas profissionais desenvolvem editores de diagramas e ferramentas CASE (Costagliola, 2004) até ambientes mais simples onde crianças e usuários finais desenvolvem jogos, animações e simuladores (SMITH, 2000).

4 REVISÃO BIBLIOGRÁFICA - XML

Esta seção faz uma revisão bibliográfica sobre o modelo XML. Adicionalmente serão revisados os principais componentes XML (BRAY, 2006-a) utilizados em ADBGs e em especial os componentes XML utilizados no SV-XML (o framework proposto nesta tese).

4.1 O modelo XML RELAX NG

Nesta tese foi utilizado o modelo de dados XML Relax NG (VLIST, 2003) para descrever a estrutura de dados de documentos XML. Por brevidade, no restante desta tese, o modelo de dados XML Relax NG será simplesmente chamado de modelo XML. O modelo XML descreve um documento XML em um nível abstrato sem descrever detalhes da sintaxe do XML.

Neste modelo de dados um documento XML é representado por um *elemento*. O *elemento* que contém todos os demais *elementos* é chamado de *elemento raiz*. Um *elemento* consiste de:

- Um *nome*.
- Um *contexto*.
- Um conjunto de *atributos*.
- Uma seqüência ordenada de zero ou mais filhos; cada filho é uma *string* ou um *elemento*.

Um *nome* é composto de:

- Uma *namespace URI*. *Namespace URI* é uma *string* que especifica um endereço único na internet. Desta maneira uma *namespace URI* pode identificar de maneira única um espaço de nomes (um esquema XML). No lugar de uma URI completa a *namespace URI* pode ser apenas uma referência a um *prefixo* definido no *contexto* do documento. A inexistência de uma *namespace URI* é uma referência ao espaço de nomes *default* do documento.
- Um *nome local*. *Nome local* referencia um nome do espaço de nomes definido na *namespace URI*.

Um *contexto* consiste de:

- Uma *base URI*. Um documento XML pode referenciar recursos externos através de endereços relativos. Os endereços relativos partem de um

endereço (a *base URI*) para encontrar recursos; quando a *base URI* não é informada, a *base URI default* é o atual endereço do documento.

- Um *namespace map*. O *namespace map* pode ter vários mapeamentos, cada um deles associa um *prefixo* a uma *namespace URI* (uma identificação única de um espaço de nomes); o *namespace map* também define o espaço de nomes *default* do documento; no restante do documento as entidades *nome* podem utilizar os *prefixos* para referenciar as *namespace URIs* associadas. A ausência de *prefixo* referencia o espaço de nomes *default*.

Um *atributo* consiste de:

- Um *nome*.
- Um *valor*. O *valor* é uma *string* que pertence a um domínio de valores. Eventualmente um *valor* pode referenciar um ou mais *elementos* de um *documento*.

Os relacionamentos entre as entidades do modelo XML são basicamente relacionamentos hierárquicos, as exceções são as referências: de *valores* para *elementos* e de *namespace URIs* para *prefixos*. O modelo XML pode ser visualizado no diagrama de classes da figura 4.1.

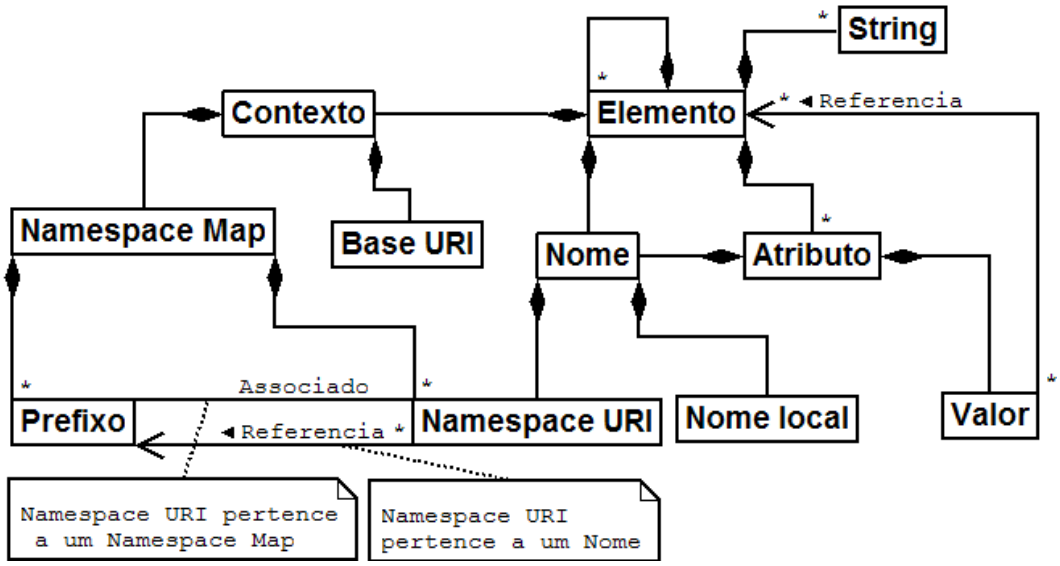


Figura 4.1: Diagrama de classes do modelo XML

Uma especificação com a sintaxe completa do XML é descrita na recomendação XML 1.1 da W3C (BRAY, 2006a) que é complementada com a recomendação Namespace 1.1 da W3C (BRAY, 2006b). É dito que a especificação XML 1.1 da W3C realiza o modelo XML Relax NG. Em (VLIST, 2003) é mostrado o mapeamento entre as entidades do modelo XML Relax NG e as entidades da especificação XML 1.1 da W3C. Para isto foi utilizada a recomendação *Information Set* (COWAN, 2004) da W3C. O *Information Set* é um modelo com um grau de abstração intermediária entre o modelo XML Relax NG e a recomendação XML 1.1 da W3C. Tais mapeamentos demonstram que qualquer artefato XML que está de acordo com a recomendação XML 1.1 da W3C também está de acordo com o modelo XML Relax NG.

4.1.1 Exemplo e convenções

Nesta seção será mostrado um exemplo de documento XML e as entidades do modelo XML que podem ser extraídas deste documento.

É importante notar que muitas entidades do modelo XML que são extraídas de um documento XML não estão explicitamente declaradas no documento, porém são definidas por convenção. Algumas entidades são declaradas implicitamente, este é o caso do o *prefixo* "xml" associado a *namespace URI* "http://www.w3.org/XML/1998/namespace" que é um *namespace map* implicitamente definido para todo documento XML. É também o caso da *base URI*. *Se a base URI* de um documento XML não é explicitamente definida, o seu valor (implicitamente definido) é a localização física do documento XML.

Outras entidades herdam valores dos *elementos* pais, o *contexto* de um *elemento* pai, por exemplo, é passado para seus *elementos* filhos, assim se os filhos não definirem um novo *contexto* eles terão o mesmo *contexto* de seus pais.

Há ainda casos onde atributos e valores de atributos são definidos em esquemas XML. Nesses casos, os esquemas definem atributos e valores implícitos. Assim, mesmo que as entidades não estejam declaradas explicitamente, elas implicitamente fazem parte de qualquer documento que esteja de acordo com este esquema. Note que as entidades implícitas definidas em esquemas XML só são adicionadas aos seus documentos se o esquema e/ou o seu suporte estiverem disponíveis.

A seguir será mostrado um documento XML e suas correspondentes entidades. Suponha que o documento localizado na URI `http://www.example.com/doc.xml` seja o seguinte (exemplo retirado de Clark et al. (2001)):

```
<?xml version="1.0"?>
<foo>
   <pre1:bar1 xmlns:pre1="http://www.example.com/n1"/>
   <pre2:bar2 xmlns:pre2="http://www.example.com/n2"/>
</foo>
```

O *elemento* raiz que representa este documento tem:

- um *nome* que tem:
    - uma *string* vazia como *namespace URI*, representando a ausência de qualquer espaço de nome;
    - "foo" como seu *nome local*;
- um *contexto* que tem:
    - "http://www.example.com/doc.xml" como sua *base URI*;
    - um *namespace map* que:
        - mapeia o *prefixo* "xml" para a *namespace URI* "http://www.w3.org/XML/1998/namespace" (o *prefixo* "xml" é implicitamente declarado para todo documento XML);
        - especifica a *string* vazia como a *namespace URI* default;
- um conjunto vazio de *atributos*;
- uma seqüência de filhos consistindo de um *elemento* que tem:
    - um *nome* que tem:

        - "http://www.example.com/n1" como a *namespace URI*;
        - "bar1" como *nome local*;
    - um *contexto* que tem:
        - "http://www.example.com/doc.xml" como *base URI*;
        - um *namespace map* que:
            - mapeia o *prefixo* "pre1" para a *namespace URI* "http://www.example.com/n1";
            - mapeia o *prefixo* "xml" para a *namespace URI* "http://www.w3.org/XML/1998/namespace";
            - especifica a *string* vazia como a *namespace URI* default;
    - um conjunto vazio de *atributos*;
    - uma seqüência vazia de filhos;
- seguido de um *elemento* que tem:
    - um *nome* que tem:
        - "http://www.example.com/n2" como a *namespace URI*;
        - "bar2" como o *nome local*;
    - um *contexto* que tem:
        - "http://www.example.com/doc.xml" como *base URI*;
        - um *namespace map* que:
            - mapeia o *prefixo* "pre2" para a *namespace URI* "http://www.example.com/n2";
            - mapeia o *prefixo* "xml" para a *namespace URI* "http://www.w3.org/XML/1998/namespace";
            - especifica a *string* vazia como a *namespace URI* default;
    - um conjunto vazio de *atributos*;
    - uma seqüência vazia de filhos.

4.2 Os espaços de nomes XML

Por ser um importante conceito do XML, esta seção da uma breve explicação sobre o conceito **espaço de nomes**. Um espaço de nomes é um container abstrato criado para manter um conjunto de identificadores (i.e. nomes) únicos. Em **espaço de nomes simples** todos identificadores de entidades pertencem ao mesmo espaço de nomes. Por isso os próprios identificadores de entidades são suficientes para se evitar ambigüidades de nomes. Um **espaço de nomes composto** pode conter vários espaços de nomes. Nestes casos, um mesmo identificador pode pertencer a dois espaços de nomes e ter significados diferentes em cada espaço. Para se evitar ambigüidades como esta, os espaços de nomes compostos contém regras ou convenções que associam cada

identificador de entidade ao seu espaço de nomes. Deste modo o identificador único de um espaço de nomes composto é o par "identificador de entidade + identificador do espaço de nomes". Para se evitar verbosidade, muitos destes pares são implicitamente deduzidos através de convenções. No caso do espaço de nomes XML (BRAY, 2006-b) o identificador de espaços de nomes é uma URI. E o par "identificador de entidade + identificador do espaço de nomes" é o par "namespace URI + nome local".

4.3 Esquemas XML e verificações sintáticas

As operações de verificação sintática XML (MURATA, 2005) consistem em verificar se a estrutura de um documento XML está de acordo com uma determinada estrutura de dados. Documentos XML podem ter dois níveis de verificação sintática:

1. Documentos bem formados: um documento que está de acordo com o modelo XML.
2. Documento válido: um documento que está de acordo com o modelo XML e de acordo com um esquema XML (DTD, XML schema, Relax NG schema, etc) (MURATA, 2005).

A validação de documentos é importante para se garantir que um documento tenha uma estrutura de dados compatível com requisitos de um determinado sistema ou domínio. Se um documento esta de acordo com um esquema XML ele é compatível com as operações e sistemas que foram criadas especificamente para documentos deste esquema.

Porém, muitas operações do modelo XML podem ser aplicadas em documentos que são apenas bem formados. Por isso um documento bem formado é uma importante opção quando um esquema XML não está disponível ou não é desejado. Neste contexto as operações que verificam se um sistema é bem formado garantem que o documento analisado é um documento XML. Consequentemente garantem que as operações XML independentes de esquema possam ser aplicadas no documento.

O processo e a capacidade de validação podem variar muito dependendo da linguagem ou mecanismo de validação utilizados. Diferentes linguagens têm diferentes expressividades. Ou seja, algumas têm a capacidade de verificar condições que outras linguagens não são capazes. A DTD, por exemplo, é menos expressiva que o XML Schema e a Relax NG schema (MURATA, 2005).

4.3.1.1 Verificações com múltiplos espaços de nomes

Um dos pontos que mais varia de verificação sintática para verificação sintática é o tratamento que é dado a documentos com múltiplos espaços de nomes (ISO/IEC, 2004). Os documentos com múltiplos espaços de nomes podem ser divididos em partes. Cada parte contém um contexto que informa o endereço URI de um esquema XML. Esta informação tem duas funções, a primeira é evitar a ambigüidade de nomes. Neste caso a URI só precisa ser um identificador único. Não importa se no endereço indicado existe ou não um esquema XML. A segunda função é definir um esquema XML. Neste caso, no endereço URI indicado precisa existir um esquema e a parte do documento associada a esta URI precisa estar de acordo com este esquema.

Em uma verificação pode-se ignorar o conteúdo do contexto e não utilizar nenhuma das duas funções do contexto. Alternativamente, pode-se utilizar apenas a primeira função ou as duas funções do contexto. Adicionalmente em diferentes partes do

documento pode-se utilizar o contexto de maneira diferente. Todas estas possibilidades permitem a existência de verificações muito variadas (ISO/IEC, 2004). Apenas para demonstrar esta variedade de opções a seguir será apresentada uma lista de opções de verificações que podem ser utilizadas. As opções estão listadas em ordem de complexidade, começando pelas mais simples:

- Verificar se o documento é bem formado. Neste caso o conteúdo de todos os contextos do documento é ignorado. Cada contexto, assim como qualquer outra parte do documento, só precisa estar bem formado.
- Verificar se um documento está de acordo com um esquema XML. Se uma parte do documento tem um contexto que não aponta para este esquema, esta parte do documento é ignorada e só precisa estar bem formada. Se uma parte do documento tem um contexto que aponta para este esquema, esta parte do documento é validada pelo correspondente esquema.
- Verificar se um documento está de acordo com um esquema XML. Neste caso é utilizada a primeira função de todos os contextos. Entretanto só é utilizada a segunda função do contexto *default* (todo documento tem um contexto *default*). O esquema do contexto *default* deve definir como validar elementos e atributos de outros contextos. Tal esquema define precisamente quais elementos/atributos externos podem ser inseridos no documento e o local no documento onde podem ser inseridos.
- Uma das mais expressivas e modulares validações é proporcionada pela *Namespace-based Validation Dispatching Language* (NVDL) (MURATA, 2004). O NVDL é um padrão ISO/IEC para validar documentos contra múltiplos esquemas. A idéia básica é que um documento seja dividido em partes. Onde cada parte pertence a um contexto. Após a divisão do documento, cada parte é despachada para ser validada pelo esquema informado em seu contexto. Se todas as partes são válidas o documento como um todo é válido. Também se destacam os seguintes recursos: (i) Pode-se despachar para diferentes linguagens de validação. Uma parte do documento pode ser despachada para uma DTD e outra para um XML Schema, por exemplo. (ii) Pode-se verificar algumas condições antes do despache. Tais condições podem verificar, por exemplo, que elementos e que contextos podem estar contidos em elementos de outros contextos. Como isto pode ser feito antes do despache, os esquemas XML que recebem os pedaços de documentos despachados permanecem inalterados. Portanto podem ser reusados em um grande número de situações.

A escolha de qual verificação será feita dependerá das intenções de quem está utilizando os documentos XML e do suporte tecnológico que estará disponível.

O uso de múltiplos espaços de nomes em documentos XML ampliou consideravelmente a capacidade de reuso de esquemas e de artefatos que executam operações em documentos XML. Para exemplificar, note o exemplo de um desenvolvedor que está criando um novo esquema XML. O desenvolvedor pode reutilizar, dentro de seu novo esquema, elementos e atributos criados em outros esquemas. Adicionalmente muitos artefatos que podem executar operações sobre artefatos de outros esquemas também podem executar estas operações no novo esquema. A capacidade de reuso de esquemas e artefatos é ampliada quando um padrão

amplamente difundido como o XML é explorado. Existem centenas de linguagens XML disponíveis para reuso e com um grande suporte acadêmico e industrial.

4.4 Especificação de eventos e ações em documentos XML

Em documentos XML pode-se definir comandos onde cada comando é composto de um **evento** que dispara uma **ação**. O evento é a ocorrência de um fato sobre o documento XML que pode ser disparado por um usuário ou sistema. O evento ocorre durante um ciclo de interação entre um usuário e um sistema ou entre dois sistemas. A ação é um processamento do sistema que é disparado por um evento. As ações podem ser customizadas por desenvolvedores de documentos XML e armazenadas em documentos XML ou em artefatos como scripts e applets. Já os eventos que podem ser utilizados são características das linguagens XML e das aplicações que interpretam documentos destas linguagens.

Há muitas maneiras de um desenvolvedor associar (*binding*) eventos, ações e entidades em documentos XML. Esta seção descreve brevemente duas destas opções.

4.4.1 Associação in-line

Uma das opções de associação mais simples e difundidas é o uso de atributos específicos de linguagens XML específicas (associação *in-line*). Algumas linguagens como o XHTML e o SVG têm em sua definição uma associação entre seus elementos e eventos que podem ocorrer nestes elementos. No SVG, por exemplo, está previamente definido que no elemento *circle* (que é visto pelo usuário como um circulo em uma tela) pode ocorrer o evento "clique do mouse". Este evento ocorre quando um elemento *circle* é clicado pelo usuário. Para se monitorar este evento e disparar uma ação quando o usuário clicar um determinado *circle* basta colocar o atributo *onclick* no referido elemento *circle*. A ação que deve ser disparada por este evento deve ser colocada no valor deste atributo. O código a seguir especifica que quando o circulo descrito pelo elemento abaixo for clicado o script *DeletaCirculo(evt)* deve ser executado.

<circle onclick="DeletaCirculo(evt)" cx="100" cy="50" r="40">

O SVG define uma série de eventos que podem ser aplicados em cada um de seus elementos. Todos os eventos têm um nome de atributo associado que por sua vez podem ser colocados em seus respectivos elementos. A ação pode ser qualquer script interpretado pela aplicação que está mostrando o documento. Alternativamente pode ser uma referência a um artefato executável ou a um código fonte que será interpretado. Maiores detalhes sobre o SVG e seus eventos podem ser vistos em (FERRAIOLO, 2003). Esta é uma maneira simples de se definir eventos e ações. Entretanto o desenvolvedor de um documento está limitado a usar somente linguagens e eventos predefinidos.

4.4.2 Associação através da linguagem XML Events

Para evitar as limitações do método in-line a W3C define na sua recomendação *XML Events* (McCARRON, 2003) meios mais flexíveis e modulares de se definir eventos e ações. Tal recomendação descreve como especificar eventos e ações, além de especificar como associar tais eventos/ações a qualquer elemento de qualquer linguagem XML.

A definição de eventos, ações e a associação (*binding*) entre evento, ação e elementos são feitas através de um conjunto específico de elementos e atributos da linguagem XML Events. Estas entidades devem ser colocadas em um documento de uma linguagem hospedeira (*host language*). A linguagem hospedeira pode ser qualquer linguagem XML (SVG, GXL, XHTML, etc). O documento da linguagem hospedeira deverá ter a seguinte declaração de espaço de nomes: *xmlns:ev="http://www.w3.org/2001/xml-events*.

Com isto muitas opções de definição de eventos/ações podem ser feitas. Em (McCARRON, 2003) está a especificação completa de todas as alternativas. Por brevidade, nesta seção será mostrado apenas um exemplo ilustrativo para dar uma idéia geral do funcionamento da recomendação *XML Events* (McCARRON, 2003).

No código abaixo o atributo *ev:event* descreve que o evento *click* que ocorrer no referido elemento *circle* deve ser monitorado. Caso este evento ocorra deve-se disparar a ação definida no recurso apontado por *ev:handler*. O valor de *ev:handler* é uma URI que aponta para um recurso que contém as instruções das ações que devem ser disparadas. O valor de *ev:event* referencia um tipo de evento.

<circle ev:handler="#DeletaCirculo" ev:event="click" cx="50" cy="50" r="40">

4.5 Edição de documentos XML

Operações de edição mostram para um usuário uma visão dos dados que estão em um documento XML. Muitas destas visões são acompanhadas de operações de interação entre os usuários e os sistemas.

Diversas visões e interações de um documento XML podem ser mostradas aos seus usuários, Quint (2004) classifica estas visões e suas respectivas interações em 4 modos denominados **modos de edição**. Cada modo será detalhado nas próximas subseções. Após serão feitas algumas considerações sobre todos os modos de edição.

4.5.1 Edição de código fonte

No seu nível mais baixo, um documento XML é uma seqüência de caracteres. O modo de edição de código fonte mostra ao usuário um documento XML como um simples conjunto de caracteres. O usuário pode interagir livremente com estes caracteres inserindo e excluindo caracteres conforme as suas necessidades. Este modo de edição da uma grande liberdade ao usuário e normalmente as suas operações são mais simples que as operações dos outros modos. O preço a pagar por esta simplicidade e liberdade é que o usuário é o responsável por manter a coerência sintática e semântica do documento que está sendo editado. Há poucas operações que ajudam o usuário a não cometer erros e poucas operações que agilizam edições mais complexas.

4.5.2 Edição estrutural livre

Num nível mais alto, um documento XML é um conjunto de entidades que está de acordo com o modelo XML. O modo de edição estrutural livre mostra ao usuário um documento XML destacando as entidades XML do documento e a sua estrutura em árvore. Cada entidade XML do documento é tratada como uma unidade lógica que pode ser excluída, inserida ou alterada desde que estas alterações estejam de acordo com o modelo XML. Visualizações comuns deste modo de edição mostram o documento como um conjunto de caixas uma dentro da outra, como um grafo, como uma árvore, como

um conjunto de caracteres onde as entidades são diferenciadas por cores e/ou endentações, etc. Neste modo de edição as operações precisam conhecer o modelo XML e suas regras de edição. Isto as torna mais complexas que as operações do modo de edição de código fonte. Porém esta complexidade pode ser compensada por um número maior de recursos que podem ajudar os usuários a editar/visualizar o documento e a evitar a ocorrência de erros relacionados a sintaxe do modelo XML.

4.5.3 Edição controlada por esquema

Neste nível de abstração, um documento XML é um conjunto de entidades tipadas que está de acordo com um esquema XML. O modo de edição controlada por esquemas mostra ao usuário um documento XML como um documento de uma linguagem XML específica. Esta linguagem está definida em uma DTD ou em outro esquema XML. Cada entidade XML tipada é tratada como uma unidade lógica que pode ser excluída, incluída ou alterada desde que estas alterações estejam de acordo com o respectivo esquema. Este modo de edição possui visualizações semelhantes ao modo de edição estruturado livre (caixas dentro de caixas, etc). Porém as operações deste modo de edição possuem recursos que facilitam a edição específica de documentos que estão de acordo com o respectivo esquema. Por exemplo, através de menus, caixas de diálogo e outros recursos estas operações permitem que somente elementos, atributos e valores válidos sejam inseridos no documento. Quando possível, estes recursos mostram quais são as opções de nomes e valores que podem ser inseridas em cada parte de um documento.

4.5.4 Edição controlada por semântica

Neste modo, as entidades XML têm um significado e um papel em uma área de domínio específica. Portanto são visualizadas e editadas de acordo com o seu significado neste domínio específico. Se a entidade XML descreve um gráfico ela é visualizado como um gráfico, se descreve uma fórmula matemática ela é visualizada como uma fórmula matemática e assim por diante. A estrutura de dados não é visualizada ou manipulada. O que é manipulado são representações da semântica dos dados. No caso de um documento XML que descreve gráficos, sua estrutura de dados precisa estar de acordo com o modelo XML e de acordo com um esquema XML que descreve gráficos como, por exemplo, o esquema SVG (FERRAIOLO, 2003). Entretanto o usuário não vê esta estrutura de dados, ele vê a interface de um editor gráfico e desenha gráficos com a ajuda deste editor. O editor então converte este desenho na estrutura de dados XML que o descreve. Este modo de edição pode ser usado por usuários finais, ou seja, usuários que conhecem sua área de domínio mas que não precisam ter nenhum conhecimento sobre o XML e a sua estrutura de dados.

Toda manipulação deste modo de edição é controlada por um conjunto de operações definidas em um esquema XML mais um conjunto de operações extras que ajudarão a manter a coerência semântica das entidades do referido esquema. As operações extras podem ser implementadas em códigos de qualquer linguagem que não seja um esquema XML. Normalmente são utilizadas linguagens de propósito geral como C, Java ou JavaScript.

4.5.5 Mesclando os modos de edição

Um documento pode ter várias visões de cada um dos quatro modos de edição. Cada visão deve mostrar diferentes aspectos ou partes do documento e todas visões devem

estar sincronizadas. Devido ao recurso de espaço de nomes, operações que pertencem a diferentes modos de edição podem ser aplicadas em um mesmo documento. Ou seja, em algumas partes do documento podem ser aplicadas operações do modo de edição controlada por esquemas e em outras são aplicadas apenas operações do modo de edição estrutural livre.

É importante salientar que todos estes modos de edição fazem parte de ambientes de desenvolvimento de documentos. Nestes ambientes os documentos são editados. No final do processo de edição os documentos são disponibilizados para usuários ou aplicações finais em ambientes de execução como, por exemplo, navegadores Web. Alguns sistemas como o Amaya são simultaneamente um ambiente de desenvolvimento e execução.

4.6 Edição de esquemas XML

Quint et al. (2004) classificaram as operações de edição de documentos XML conforme as regras que controlam estas edições. Utilizando estes mesmos critérios esta tese propõe a classificação das operações de edição de esquemas XML em dois modos.

4.6.1 Modo de edição de esquema

Neste modo de edição, um esquema XML é um conjunto de entidades que está de acordo com um meta-modelo de esquema como, por exemplo, o modelo que especifica a estrutura de uma DTD ou a estrutura de um XML schema. O modo de edição de esquema mostra ao usuário um esquema XML destacando as entidades do meta-modelo e a sua estrutura. Cada entidade do meta-modelo é tratada como uma unidade lógica que pode ser excluída, inserida ou alterada desde que estas alterações estejam de acordo com o meta-modelo de esquema.

Neste modo de edição os esquemas XML são criados, visualizados e editados. O que equivale a dizer que tipos de entidades são criados, visualizados e editados.

4.6.2 Modo de edição de esquema controlada por semântica

Editores de esquemas XML normalmente não têm visões semânticas. Normalmente a definição de uma estrutura de dados que pode descrever gráficos não é visualizada como um gráfico. As estruturas que podem descrever fórmulas matemáticas não são visualizadas como fórmulas matemáticas, e assim por diante.

Somente aplicações específicas de áreas de domínio específicas (como a área de especificação de sistemas visuais interativos baseados em gramáticas) é que estendem/adaptam os editores de esquemas para adicionar visões semânticas. Um exemplo destas extensões são esquemas de uma área específica que só podem descrever estruturas gráficas. Nestes casos, como em qualquer outro editor de esquemas XML, pode-se descrever a estrutura de dados dos documentos XML que estão de acordo com o esquema que está sendo editado. Porém pode-se ter uma extensão que associa um gráfico a cada estrutura do documento. A visão que mostra este gráfico seria uma visão do modo de edição de esquema controlada por semântica.

De fato este modo de edição é raro e tem pouca importância para o espaço tecnológico XML como um todo. Porém ele é utilizado em ADBGs. Por isso é importante para a área de foco desta tese (a intersecção entre os espaços tecnológicos XML e de linguagens visuais).

4.7 Editores do XML

Embora as operações dos modos de edição XML possam ser colocadas em diversos sistemas a sua forma mais comum de agrupamento (e mais relevante para esta tese) é o agrupamento ou encapsulamento em editores. Estes editores encapsulam as operações de um modo de edição XML permitindo a edição de documentos de acordo com o correspondente modo de edição. A seguir são citados os seis modos de edição XML e as classes de editores que encapsulam as operações de cada modo de edição:

- As operações do modo de edição de código fonte são normalmente encapsuladas em um **editor de código fonte**.
- As operações do modo estrutural livre são encapsuladas em um **editor XML**.
- As operações do modo de operação controlada por esquema são encapsuladas em **editores XML controlados por esquemas**.
- As operações do modo de operação controlada por semântica são encapsuladas em **extensões de editores XML controlados por esquemas** que editam entidades de um esquema específico de acordo com a semântica de uma área de domínio específica.
- As operações do modo de edição de esquemas são encapsuladas em **editores de esquemas XML**.
- As operações do modo de edição de esquema controlada por semântica são encapsuladas em **extensões de editores de esquemas XML** que editam esquemas de acordo com a semântica de uma área de domínio específica.

Os editores XML citados são padrões do espaço tecnológico XML, por isso são modulares e podem ser combinados de diversas maneiras. Por exemplo, várias extensões de editores de esquemas XML podem ser extensões de um único editor de esquemas XML que por sua vez é uma extensão de um editor XML que por sua vez é uma extensão de um editor de código fonte. Todos editores podem fazer parte de um único sistema que controla e sincroniza todas visões ou alternativamente todos podem existir separadamente. Quint e Vatton (2004) descrevem maiores detalhes e exemplos sobre estes editores. A primeira coluna da tabela 4.1 mostra na os editores XML. A segunda coluna mostra os modos de edição XML encapsulados nestes editores. Por fim na terceira coluna é mostrado o documento que pode ser editado no correspondente modo de edição.

Tabela 4.1: Editores, modos de edição e documentos XML

Editores XML	**Modos de edição do XML**	**Documento**
Editor de código fonte	Código fonte	Documento XML
Editor XML	Estrutural livre	Documento XML
Editor XML controlado por esquema	Controlada por esquema	Documento XML
Extensão de editor XML controlado por esquema	Controlada por semântica	Documento XML
Editor de esquema XML	Edição de esquema	Esquema XML
Extensão de editor de esquema XML	Edição de esquema controlado por semântica	Esquema XML

5 SV-XML – UM FRAMEWORK DE ADBG BASEADO NO MODELO XML

Este capítulo apresenta as principais propostas desta tese: o modelo SV-XML e o framework SV-XML (TELECKEN, 2008). O modelo SV-XML é o modelo XML interpretado como um modelo de sentença visual em ADBGs. O framework de projeto físico de ADBG SV-XML especifica um conjunto de componentes XML que é compatível com o referido modelo. Sem perder funcionalidades, os componentes XML definidos no framework SV-XML são mais simples que os componentes XML utilizados em outras abordagens.

Resultados preliminares e propostas relacionadas com esta tese foram publicados em (TELECKEN, 2005-a; 2005-b; 2004-a; 2004-b; 2004-c). Já parte dos resultados finais foram publicados em (TELECKEN, 2008).

5.1 O modelo de sentença visual SV-XML

Nesta seção será apresentado como o modelo XML pode ser um modelo de sentença visual.

A interpretação original do modelo XML define que suas entidades especificam uma estrutura de dados. Sendo que tal estrutura pode ser utilizada para descrever qualquer tipo de informação de qualquer área de domínio (BRAY, 2006a). Esta tese propõe uma interpretação mais específica do modelo XML. É proposto que as entidades XML especifiquem a estrutura de uma sentença visual. Com esta interpretação o modelo XML pode ser utilizado como um modelo de sentença visual.

Para facilitar a compreensão do texto, no restante desta tese o termo modelo SV-XML (Sentença Visual baseada no modelo XML) será utilizado para referenciar o modelo XML que é usado como um modelo de sentença visual. Já o termo modelo XML será utilizado para referenciar o modelo XML que é usado da maneira originalmente proposta pela W3C, ou seja, uma estrutura de dados que pode descrever qualquer tipo de informação de qualquer área de domínio. É importante novamente salientar que os dois são sintaticamente o mesmo modelo, o que os diferencia é a sua interpretação.

5.1.1 Sintaxe do modelo SV-XML

Ao utilizar o modelo XML como um modelo de sentença visual, a sua sintaxe é exatamente a mesma que a sintaxe do modelo XML que é utilizado em qualquer outro domínio. Em outras palavras o modelo SV-XML tem exatamente as mesmas estruturas, entidades e relacionamentos que o modelo XML. De fato, os dois são o mesmo modelo,

só que são utilizados para diferentes fins, ou seja, com diferentes semânticas/interpretações, como é mostrado na próxima seção.

5.1.2 Semântica do modelo SV-XML

Esta seção descreve a maneira que as entidades XML devem ser interpretadas quando o modelo XML é utilizado como um modelo de sentença visual. Algumas entidades manterão a sua interpretação XML original outras deverão ser interpretadas como entidades típicas de modelos de sentença visual.

No modelo SV-XML, cada *elemento* deve ser interpretado como um símbolo de uma sentença visual. As informações que estão contidas no *elemento* XML são: o *nome*, *atributos*, o *contexto* e seus filhos.

O *nome* de um *elemento* XML corresponde ao nome de um símbolo.

Os *atributos* de um *elemento* XML correspondem a *atributos* de um símbolo. Os *atributos* XML possuem um *nome* e um *valor* que correspondem respectivamente ao nome e ao valor de um atributo de um símbolo. Os *valores* XML podem referenciar um ou vários *elementos* de um documento XML. Tais referências correspondem a referências que um atributo de um símbolo pode fazer a outros símbolos de uma sentença visual.

Um *elemento* XML pode conter outros *elementos* XML formando uma hierarquia de elementos. Tal hierarquia corresponde a uma hierarquia de símbolos onde um símbolo pode conter outros símbolos. O *elemento raiz* do XML representa um símbolo que contém todos os outros símbolos. Portanto o elemento raiz corresponde a própria sentença visual.

Além dos *elementos* normais os filhos dos *elementos* XML podem ser *caracteres* que são chamados de *elementos básicos*. Cada *caractere* corresponde a um símbolo básico.

A definição de um *contexto* (incluindo suas sub-entidades *prefixo*, *base URI*, namespace map, etc) para *elementos* e *atributos* XML corresponde à definição de um *contexto* para símbolos e atributos de uma sentença visual.

Similarmente a definição de um *namespace URI* e um *nome local* para um *atributo/elemento* XML corresponde à definição de um *namespace URI* e um *nome local* para um símbolo/atributo de uma sentença visual.

5.1.2.1 XML como estrutura de dados e como estrutura de sentença visual

Após a apresentação sobre como uma sentença visual do SV-XML deve ser interpretada é importante destacar algumas diferenças entre esta interpretação e a interpretação original de entidades XML (BRAY, 2006).

A interpretação original das entidades do modelo XML define que suas entidades especificam uma estrutura de dados. Onde um *elemento* pode descrever literalmente qualquer tipo de informação. Quando o XML é interpretado como modelo de sentença visual, um *elemento* sempre descreve um símbolo que faz parte de uma sentença visual.

Enquanto na interpretação original o relacionamento entre um *elemento* pai e o *elemento* filho pode ter qualquer significado, no modelo SV- XML este relacionamento sempre descreve que o *elemento* filho é uma parte do *elemento* pai. Em outras palavras, o símbolo representado pelo *elemento* pai pode ser composto por diferentes sub-

símbolos, sendo que cada sub-símbolo é descrito por um *elemento* filho. É importante destacar que um *elemento* filho pode tanto ser um *elemento* normal quanto um *elemento básico* (um *caractere*).

Enquanto no modelo XML o *elemento* raiz pode representar qualquer entidade, no modelo SV-XML o *elemento* raiz representa um símbolo que é a própria sentença visual.

5.1.2.2 SV-XML e outros modelos de sentença visual

Nesta seção as entidades XML interpretadas como sentenças visuais serão comparadas com entidades de outros modelos de sentença visual.

As entidades *elemento*, *atributo*, *nome* e *valor* do SV-XML equivalem semanticamente a entidades comuns à maioria dos modelos de sentença visual. As entidades citadas anteriormente, por exemplo, equivalem respectivamente às entidades símbolo, atributo, nome e valor do modelo VCARW (BOTTONI, 2007).

A descrição dos relacionamentos entre estas entidades variam um pouco mais de modelo para modelo. Pode-se definir o relacionamento através de referências nos valores dos atributos. Através de símbolos que possuem atributos que descrevem relacionamentos. Através da contenção, ou seja quando um símbolo está dentro de outro símbolo formando uma hierarquia de símbolos. Através da ordem e posição em que os símbolos são descritos, etc. No caso do modelo SV-XML pode-se utilizar os valores de atributos, a contenção e a ordem dos elementos. Um modelo de sentença visual que utiliza todos os recursos de descrição de relacionamentos citados neste parágrafo é o GXL (HOLT, 2006).

A entidade elemento básico (caractere) também tem entidades correspondentes na maioria dos modelos de sentenças visuais. Nesses modelos o elemento básico é chamado de elemento folha, símbolo folha, símbolo básico, nodo básico entre outros.

No modelo SV-XML e na maioria dos demais modelos podem existir símbolos que não aparecem na sentença visual. Os motivos são os mais variados: pode ocorrer sobreposição de símbolos, os símbolos podem ser transparentes, imperceptíveis ou simplesmente eles podem conter informações que não tem uma representação gráfica em uma determinada visão dos dados de uma sentença visual.

A divisão do nome em duas partes (*namespace URI* + *nome local*) tem como objetivo definir identificadores únicos em um ambiente com múltiplos espaços de nomes. O modelo GXL usa para o mesmo objetivo a entidade *type*. No GXL a entidade *type* deve conter um endereço URI que pode ser dividido em duas partes. Uma parte identifica unicamente um espaço de nomes (é o identificador do espaço de nomes) e outra o nome local da entidade neste espaço de nomes (o identificador da entidade). Isto é exatamente o que as entidades *namespace URI* + *nome local* do SV-XML fazem. No GXL, a associação entre o identificador e a entidade identificada é feita ao colocar-se a entidade *type* dentro da entidade que se quer identificar. No SV-XML isto é feito colocando-se o par *namespace URI* + *nome local* dentro da entidade *nome* de cada entidade.

O SV-XML ainda tem recursos para evitar a verbosidade dos identificadores. O par *prefixo* + *namespace URI* dentro de uma entidade *namespace map* associa um prefixo a uma URI. Com isso pode-se usar o prefixo (que é bem mais curto) no lugar de uma URI para se identificar um espaço de nomes. Recursos e convenções para se evitar

verbosidade de identificadores também são comuns em vários modelos de sentença visual.

A entidade *base URI* do SV-XML define um ponto em um sistema de endereços. A partir deste ponto pode-se definir caminhos relativos que apontam para recursos externos utilizados ou referenciados em uma sentença visual. Entidades com objetivos semelhantes podem ser encontradas em outros modelos de sentença visual, como por exemplo o GXL.

As entidades *contexto* e *namespace map* são containeres utilizados para organizar e agrupar entidades relacionadas. Containeres de entidades também são recursos comuns encontrados em quase todos os modelos de sentenças visuais.

Na primeira coluna da tabela 5.1 são mostradas entidades ou funções exercidas por grupos de entidades típicas de modelos de sentenças visuais. Na segunda coluna são mostradas as entidades do modelo SV-XML que são equivalentes as entidades da primeira coluna ou que executam as funções descritas nesta coluna.

Tabela 5.1: Entidades de modelos de sentença visual X Entidades do modelo SV-XML

Entidades de Modelos de Sentença Visual		**Entidades do modelo SV-XML**
Símbolo	≅	Elemento
Atributo de símbolos	≅	Atributo
Valor de atributo de símbolo	≅	Valor
Identificador único	≅	Nome
Identificador do espaço de nomes	≅	Namespace URI
Identificador da entidade	≅	Nome local
Container de entidades	≅	Contexto
Ponto em um sistema de endereço	≅	Base URI
Container de entidades	≅	Namespace map
Recurso para se evitar verbosidade	≅	Prefixo

O SV-XML tem todas as entidades obrigatórias de um modelo de sentença visual baseado em atributos (*símbolo*, *atributo*, *nome* e *valor*). Adicionalmente todas as entidades do SV-XML correspondem a entidades ou funções encontradas em outros modelos de sentença visual baseados em atributos. Portanto pode-se concluir que o modelo SV-XML pode apropriadamente ser um modelo de sentença visual baseado em atributos.

Os modelos de gramáticas que podem utilizar o modelo de sentença visual SV-XML são limitados, pois, os modelos de gramáticas precisam ser compatíveis com o modelo de sentença visual. Neste sentido o modelo SV-XML é um modelo de sentença visual baseado em atributos. Portanto o SV-XML é compatível com qualquer modelo de gramática compatível com modelos de sentença visual baseados em atributos. Várias gramáticas com os mais diferentes objetivos e características são compatíveis com modelos de sentenças visuais baseadas em atributos. Esta variedade abrange desde gramáticas simples como a gramática adotado pelo ADBG stagecast creator (SMITH, 2000) até gramáticas mais elaboradas como as gramática VCARW e evCARW (BOTTONI, 2007).

5.1.3 A expressividade do modelo SV-XML

Nesta seção será demonstrado que o modelo SV-XML é pelo menos tão expressivo quanto o modelo de sentenças visuais do VCARW. A especificação formal do VCARW e de sua expressividade é demonstrada no anexo A.

***

Teorema 1: O modelo SV-XML é pelo menos tão expressivo quanto o modelo VCARW (BOTTONI, 2007; BOTTONI, 1999).

Prova: Cada sentença visual VCARW pode ser substituída por um equivalente documento SV-XML quando:

- cada *símbolo* da sentença visual é substituído por um *elemento* XML que é filho do *elemento raiz*, e onde:
    - o *nome* do *símbolo* corresponde ao *nome* do *elemento*;
    - cada *atributo* do *símbolo* corresponde a um *atributo* do *elemento*;
        - cada *nome* de cada *atributo* do *símbolo* corresponde a um *nome* de *atributo* do *elemento*:
        - cada *valor* de cada *atributo* do *símbolo* corresponde a um *valor* de um *atributo* do *elemento*;

No mapeamento acima as entidades *nome* do modelo SV-XML possuem *namespace URIs* vazias e o *nome local* igual aos correspondentes *nomes* das entidades do modelo de sentença visual VCARW. Através deste mapeamento é possível transformar qualquer sentença visual VCARW em um correspondente documento SV-XML. Portanto o modelo SV-XML é pelo menos tão expressivo quanto o modelo de sentença visual VCARW.

***

Corolário 2: O modelo SV-XML é capaz de representar qualquer sentença visual definida por uma gramática livre do contexto, uma *Constraint Multiset Grammar*, uma *Picture Layout Grammar* ou uma gramática VCARW.

Prova: Em (BOTTONI, 1999), foi demonstrado que uma gramática VCARW é pelo menos tão expressiva quanto uma gramática livre do contexto, uma *Constraint Multiset Grammar* (GOLIN, 1991) ou uma *Picture Layout Grammar* (HELM, 1991). Adicionalmente, foi demonstrado que as sentenças visuais VCARW são capazes de representar todas as sentenças visuais definidas por uma gramática VCARW.

Como o modelo SV-XML é pelo menos tão expressivo quanto o modelo de sentença visual VCARW (teorema 1), pode-se deduzir que: o modelo SV-XML é capaz de representar qualquer sentença visual definida por uma gramática livre do contexto, uma *Constraint Multiset Grammar*, uma *Picture Layout Grammar* ou uma gramática VCARW.

***

A expressividade aqui demonstrada é típica dos principais modelos de sentença visual utilizados atualmente. Portanto o SV-XML é tão expressivo quanto os principais modelos de sentença visual atualmente utilizados, como por exemplo os modelos

utilizados nos ADBGs Genial, Diagen, GenGed e VLDesk (BOTTONI, 2004; BRIELER, 2008; COSTAGLIOLA, 2005; ERMEL, 2004;).

5.1.4 Convenções do modelo SV-XML

As convenções de um modelo de sentença visual são características únicas de cada modelo e/ou do sistema que o suporta. Em qualquer modelo o objetivo das convenções é facilitar a edição/interpretação de entidades, prevenir erros ou dar suporte a outras operações.

As convenções do modelo SV-XML são as mesmas do modelo XML. Portanto todas as convenções válidas em documentos XML são válidas em documentos SV-XML. Logo, se um elemento não tem contexto, o contexto é herdado do elemento pai; se a sentença visual não tem uma base URI, a base URI é o endereço URI da sentença visual; atributos implícitos podem estar declarados em esquemas XML; etc.

5.2 A realização das estruturas de dados no framework SV-XML

5.2.1 O modelo de sentença visual

No framework SV-XML o modelo de sentença visual é o modelo XML. O padrão XML determina que a realização do modelo XML seja uma especificação do modelo XML como, por exemplo, a especificação XML 1.1 da W3C (BRAY, 2006a). Por isso foi definido que no framework SV-XML uma especificação XML realiza o modelo de sentença visual.

O uso de uma especificação XML para realizar um modelo de sentença visual é uma proposta original desta tese.

5.2.2 O alfabeto e o meta-modelo do alfabeto

Um esquema XML define o conjunto de elementos que podem ser inseridos em documentos que estão de acordo com este esquema. Além disto, um esquema define os possíveis atributos, valores, contextos e filhos destes elementos (MURATA, 2005). Com estas características um esquema XML pode realizar um alfabeto de uma sentença visual, pois um alfabeto define o conjunto de símbolos e respectivos atributos, valores, contextos e filhos que podem ser colocados em uma sentença visual.

Portanto, no SV-XML um alfabeto é realizado por uma DTD, um XML Schema ou qualquer outro esquema XML. Consequentemente o meta-modelo do alfabeto é realizado em um meta-modelo de esquema XML como, por exemplo, o meta-modelo de DTDs ou o meta-modelo de XML Schema.

É importante salientar que todo alfabeto contém as regras do modelo de sentença visual. Consequentemente todo esquema XML contém as regras definidas em uma especificação XML. Porém as regras sintáticas da especificação XML não ficam explicitamente armazenadas no esquema XML. Como estas regras são fixas e estão contidas em todos os esquemas elas não precisam ser declaradas. Sabe-se implicitamente que fazem parte das regras sintáticas de qualquer esquema.

O uso de um esquema XML e um meta-modelo de esquema XML para realizar respectivamente um alfabeto e um meta-modelo de alfabeto é uma proposta original desta tese.

5.2.3 A gramática e o meta-modelo de gramática

Esquemas XML são capazes de definir a estrutura de documentos que podem armazenar os mais diferentes tipos de informações inclusive as informações de qualquer tipo de gramática. Por isso, para definir a estrutura de uma gramática esta tese propõe que seja adotado um esquema XML cujos documentos sejam capazes de armazenar os dados de um tipo específico de gramática.

Assim, no SV-XML o meta-modelo de gramáticas é realizado em um esquema XML que descreve a estruturas de um tipo de gramática. Já as gramáticas do SV-XML são realizadas em documentos XML que estão de acordo com o referido esquema XML.

A realização de gramáticas e seu meta-modelo apresentada nesta seção não é uma proposta original desta tese. Os demais ADBGs que realizam gramáticas e meta-modelos de gramática em componentes XML também utilizam respectivamente um documento XML e um esquema XML.

5.2.4 Verificação sintática no framework SV-XML

No SV-XML, os três níveis de verificação sintática de sentenças visuais são realizados da seguinte forma.

5.2.4.1 Sentença validada por modelo de sentença visual

Uma sentença validada por modelo de sentença visual é aquela que está de acordo com o modelo de sentença visual. Sabe-se que no SV-XML o modelo de sentença visual é realizado por uma especificação do modelo XML. Logo, no SV-XML uma sentença visual validada por modelo de sentença visual é realizada por um documento XML bem formado. Ou seja, um documento XML de acordo com a especificação XML.

Consequentemente, no SV-XML as operações que verificam se uma sentença visual está em conformidade com o modelo de sentença visual são realizadas por operações que verificam se um documento XML qualquer é bem formado. Tais operações estão normalmente contidas em verificadores sintáticos padronizados como, por exemplo, validadores XML.

5.2.4.2 Sentença validada por alfabeto

No SV-XML um alfabeto é realizado em um esquema XML. Consequentemente as operações que verificam se uma sentença visual está em conformidade com um alfabeto são realizadas em operações que verificam se um documento XML está em conformidade com um esquema XML. Tais operações estão normalmente contidas em verificadores sintáticos XML padronizados como, por exemplo, verificadores de DTDs ou de XML Schemas.

5.2.4.3 Sentença validada por gramática

No SV-XML uma gramática é realizada em um documento XML. Este documento descreve um conjunto de regras sintáticas que irão definir o conjunto de sentenças visuais válidas para esta gramática. Portanto, para verificar se uma sentença visual pertence a uma gramática deve-se interpretar as regras da gramática e verificar se a referida sentença visual está de acordo com esta gramática.

Para interpretar e verificar as regras do documento XML que realiza a gramática não existem verificadores sintáticos XML padronizados. Para esta tarefa deve-se criar novos verificadores/interpretadores e eventualmente utilizar operações XML mais básicas e/ou operações independentes da tecnologia XML.

É importante salientar que os três modos de verificação sintática são acumulativos. Por isso antes de verificar se uma sentença está de acordo com uma gramática deve-se primeiramente verificar se a gramática esta de acordo com o alfabeto informado na gramática. De forma análoga, antes de verificar se uma sentença está de acordo com um alfabeto deve-se primeiramente verificar se a sentença está de acordo com o modelo de sentença visual do ADBG. Somente sentenças visuais que estão de acordo com o modelo de sentença visual de um ADBG podem estar de acordo com um alfabeto deste ADBG. Analogamente somente sentenças visuais que estão de acordo o alfabeto informado em uma gramática podem estar de acordo com esta gramática.

5.2.4.4 Verificação com múltiplos espaços de nomes

A figura 5.1 apresenta o modelo que descreve como alfabetos podem ser combinados para definir a estrutura de dados de sentenças visuais com múltiplos espaços de nomes. No SV-XML este modelo é realizado da seguinte maneira.

Um **alfabeto** é realizado em um esquema XML. Um **alfabeto simples** é realizado em um esquema XML simples que tem um único espaço de nomes especificado em um único esquema XML.

O **alfabeto vazio** é realizado em um esquema XML que não acrescenta nenhuma regra além das regras que determinam que um documento seja bem formado. Assume-se que o alfabeto é vazio quando o esquema xml não é informado, não é encontrado, não está disponível, não contém regras ou quando explicitamente é solicitado que as regras de um esquema XML sejam ignoradas.

O **alfabeto composto** é realizado em um esquema XML composto que define como entidades especificadas em vários esquemas XML podem ser combinadas em um único documento. Um script NVDL pode especificar as regras de um alfabeto composto. Uma convenção de um sistema também pode especificar as regras de um alfabeto composto. A seguir um exemplo de convenção: ”Em um determinado sistema pode-se convencionar que um documento XML deva ser validado por seu esquema *default*. Qualquer contexto que não aponte para o esquema *default* indica que suas entidades precisam ser apenas bem formadas. Pode-se colocar entidades de contextos externos em qualquer parte do documento XML. A validação pelo esquema *default* irá ignorar qualquer entidade que seja de outros contextos”.

Os componentes XML que realizam as classes do modelo de composição de alfabetos do SV-XML são mostrados no correspondente estereótipo de cada classe da figura 5.1

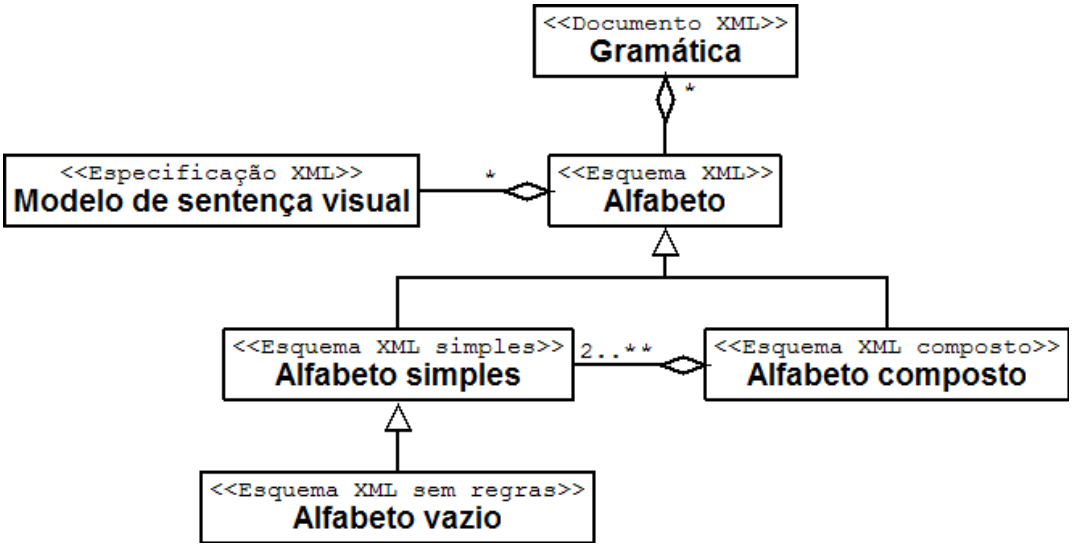


Figura 5.1: Múltiplos espaços de nomes no SV-XML

5.2.5 Comandos, eventos e reação em documentos SV-XML

No SV-XML os comandos das sentenças visuais são realizados através de eventos XML e de ações XML. As associações entre eventos, ações e elementos XML podem ser realizadas por métodos de associação como o método *in-line* ou o uso de elementos e atributos da linguagem *XML Events* (McCARRON, 2003).

Um evento XML é a ocorrência de um fato sobre um documento XML que pode ser disparada por um usuário ou sistema. Já uma ação XML é um processamento do sistema que é disparado por um evento ocorrido em um documento XML.

Já foi dito que no SV-XML uma sentença visual é realizada em um documento XML. Logo, no SV-XML um evento de uma sentença visual é realizado em um evento XML, já uma reação é realizada em uma ação XML.

Em ambientes de desenvolvimento, desenvolvedores associam comandos a símbolos de uma sentença visual. A operação equivalente no SV-XML é a associação entre eventos, ações e elementos XML. Esta associação é realizada no SV-XML através de métodos como o *in-line* ou o uso da linguagem *XML Events*.

5.2.6 Sistema Visual Interativo SV-XML

O componente XML que realiza um sistema visual interativo no SV-XML é um documento XML associado a um conjunto de componentes que armazenam especificações de ações XML. Estas ações podem efetuar as transformações em sentenças visuais previstas na gramática que especificou o referido sistema visual interativo.

Os componentes que armazenam ações podem ser scripts, applets ou qualquer componente que seja capaz de alterar um documento XML.

O documento XML está associado a estes componentes através de associações *in-line*, atributos *XML Events* ou qualquer outro método que associa elementos, eventos e ações XML. Quando um evento monitorado acontece, a correspondente ação é disparada. Esta ação poderá alterar o documento XML. Dentre outras alterações a ação

pode trocar as associações evento/elemento/ação. Desta maneira a cada transformação é definido as ações que podem ser aplicadas no documento que foi transformado.

O documento XML e os componentes associados a este documento são gerados automaticamente pelo ambiente de desenvolvimento SV-XML. Eles estão especificados sem ambigüidades em uma gramática que foi projetada por um desenvolvedor.

O ambiente de execução deste sistema visual interativo é um componente capaz de interpretar os documentos XML gerados e seus respectivos componentes de ações. O ambiente de execução implementado no âmbito desta tese é um Navegador Web capaz de interpretar documentos SVG (FERRAIOLO, 2003) e scripts JavaScripts. Mais detalhes sobre o ambiente implementado podem ser vistos na seção 6.

5.2.7 O modelo de linguagem visual SV-XML e sua realização

Com base no que já foi exposto, esta seção apresenta os componentes XML que realizam o modelo de linguagem visual SV-XML. O modelo de linguagem visual do SV-XML é apresentado na figura 5.2. A realização deste modelo conforme as definições do framework SV-XML é apresentada a seguir:

- O modelo de sentença visual é realizado pelo modelo XML.
- Uma sentença visual é realizada por um documento XML.
- Um símbolo é realizado por um elemento XML.
- Um alfabeto é realizado por um esquema XML.
- Uma gramática é realizada por um documento XML que está de acordo com um esquema XML que descreve documentos capazes de armazenar informações sobre gramática.
- O meta-modelo de alfabeto é realizado por um meta-modelo de esquema XML.
- O meta-modelo de gramática é realizado por um esquema XML.
- Um evento é realizado por um evento XML.
- Uma reação é realizada por uma ação XML.
- Um comando é realizado por um conjunto de eventos XML e ações XML.
- Uma linguagem visual é realizada por um conjunto de documentos XML válido e um conjunto de comandos associado a estes documentos.
- Um sistema visual interativo especificado em uma gramática é realizado por um documento XML associado a artefatos que armazenam a especificação de ações XML.

Os componentes XML que realizam as classes do modelo de linguagem visual do SV-XML são mostrados no correspondente estereótipo de cada classe da figura 5.2.

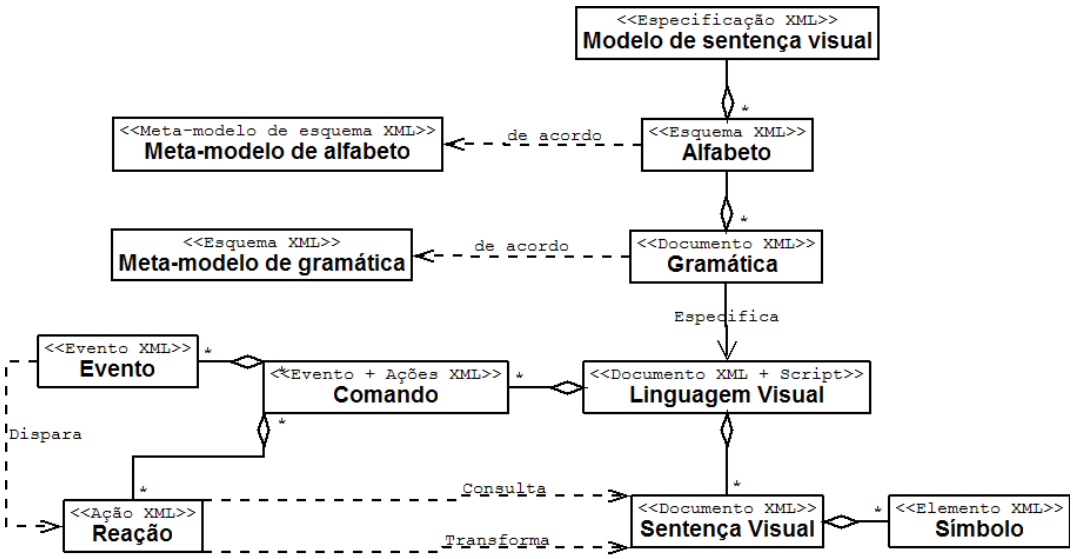


Figura 5.2: Modelo de linguagem visual do SV-XML e sua realização

Todos estes componentes são encapsulados em um ambiente de desenvolvimento e em um ambiente de execução. No ambiente de desenvolvimento um desenvolvedor define uma gramática com a ajuda de editores apropriados. A partir de uma gramática o ambiente de desenvolvimento gera um sistema visual interativo que consiste de um documento XML e artefatos que armazenam ações XML.

5.3 Realização das operações de edição do SV-XML

Até este ponto foram descritos detalhes sobre os componentes que realizam as estruturas de dados de um ADBG no SV-XML. Adicionalmente foi mostrado como estas estruturas são integradas neste framework. Nesta seção serão dados maiores detalhes sobre os componentes que realizam as operações que podem ser aplicadas sobre estas estruturas. Ou seja, as operações de edição que podem auxiliar o desenvolvedor a analisar, criar e alterar sentenças visuais, alfabetos e gramáticas.

Ao utilizar as estruturas descritas no modelo de linguagem visual do SV-XML pode-se livremente implementar as operações que irão manipular estas estruturas. Entretanto a grande vantagem de se utilizar uma estrutura de dados do espaço tecnológico XML é a compatibilidade com um padrão de fato em armazenagem de informações.

Tal compatibilidade permitirá o reuso de componentes XML que realizam operações e sistemas. Neste sentido o primeiro passo proposto por esta tese é o mapeamento dos modos de edição de ADBGs para os modos de edição XML que podem realizá-los. O passo seguinte é a utilização dos componentes e artefatos XML apropriados para cada modo de edição.

Os modos de edição de ADBGs previstos no framework SV-XML são os 6 modos de edição típicos de ADBGs mostrados na seção 2.3.2. Estes modos descrevem em alto nível características gerais sobre as funcionalidades de grupos de operações. Ao mapear estes modos de edição para os modos de edição XML são dadas informações adicionais sobre como os modos de edição do SV-XML são realizados em componentes XML.

Para exemplificar, ao informar que um ADBG tem um modo de edição controlado por alfabeto é informado que o ambiente disponibiliza as funcionalidades descritas na

seção 2.3.2.3. Ao informar que no SV-XML o modo de edição controlado por alfabeto é realizado como um modo de edição controlado por esquemas do XML são dadas informações sobre sua realização em um ambiente XML. Cada modo de edição XML tem um conjunto específico de componentes e artefatos que podem ser utilizados em sua implementação. Tais artefatos incluem ferramentas, sistemas, APIs, técnicas de programação, técnicas de modelagem, técnicas de desenvolvimento entre outras. Assim além de definir características gerais sobre como os modos de edição são realizados, o mapeamento proposto nesta tese informa um conjunto de artefatos e componentes que podem ser utilizados na realização e implementação de cada modo de edição de ADBGs construidos a partir do framework SV-XML.

As próximas seções estão divididas conforme os modos de edição XML apresentados na seção 4.5. Cada seção mostrará uma pequena descrição de um modo de edição XML e de alguns modos de edição de ADBGs. Após será descrito como cada modo de edição XML pode realizar seus respectivos modos de edição ADBG no SV-XML.

Após estas seções, é apresentado um resumo destes mapeamentos e como é possível a mesclagem dos modos de edição.

5.3.1 Edição de código fonte

5.3.1.1 No XML

O modo de edição de código fonte do XML permite a edição de documentos XML como um simples conjunto de caracteres. O usuário pode interagir livremente com estes caracteres inserindo e excluindo caracteres conforme as suas necessidades.

5.3.1.2 Nos ADBGs

O modo de edição de entidades básicas permite a edição de sentenças visuais, alfabetos e gramáticas como um simples conjunto de caracteres ou qualquer outro conjunto de entidades básicas. O usuário pode interagir livremente com estas entidades inserindo e excluindo entidades conforme as suas necessidades.

5.3.1.3 Realização no SV-XML

No SV-XML as operações da edição de entidades básicas que editam caracteres de sentenças visuais, alfabetos e gramáticas são realizados por operações do modo de edição de código fonte do XML.

Esta correspondência justifica-se porque edições em código fonte manipulam caracteres independentemente dos caracteres pertencerem a uma sentença visual, alfabeto, gramática, documento XML ou esquema XML.

Assim as ferramentas, editores e outros componentes do modo de edição de código fonte XML podem ser reutilizados para realizar operações do modo de edição de entidades básicas de ADBGs.

5.3.2 Edição estrutural livre

5.3.2.1 No XML

A edição estrutural livre do XML permite a edição de documentos XML guiada pelo modelo XML. Este modo de edição mostra ao usuário um documento XML destacando as entidades XML do documento e a sua estrutura.

5.3.2.2 Nos ADBGs

A edição controlada por modelo de sentença visual permite a edição de uma sentença visual guiada pelo modelo de sentença visual. Este modo de edição mostra uma sentença visual destacando as entidades definidas no modelo de sentença visual.

5.3.2.3 Realização no SV-XML

No SV-XML, as operações de edição controlada por modelo de sentença visual são realizadas por operações de edição estrutural livre do XML.

Esta correspondência justifica-se por que:

- o modelo de sentença visual do SV-XML é realizado por uma especificação do modelo XML;
- as sentenças visuais SV-XML são realizadas por documentos XML;
- as operações de edição estrutural livre do XML destacam e facilitam a edição de entidades do modelo de sentença visual do SV-XML, que é o próprio modelo XML.

Tanto as visões do modo de edição estrutural livre do XML quanto às visões do modo de edição controlada por modelo de sentença visual são visões sintáticas. Ou seja mostram estruturas de dados e não o que os dados representam.

As operações de edição estrutural livre do XML são guiadas pelo modelo XML e auxiliam a edição, visualização e análise da estrutura de dados de documentos XML independentemente da área de domínio que os dados do documento estão descrevendo. Por isso as operações que auxiliam a edição de documentos XML que representam sentenças visuais são as mesmas operações que auxiliam a edição de documentos XML que representam fórmulas matemáticas, cadastros de livros, mensagens eletrônicas, páginas Web, etc.

Assim as ferramentas, editores e outros componentes do modo de edição estrutural livre do XML podem ser reutilizados na realização das operações do modo de edição controlada por modelo de sentença visual de ADBGs.

5.3.3 Edição controlada por esquemas

5.3.3.1 No XML

A edição controlada por esquema do XML permite a edição de documentos XML guiada por um esquema XML. Este nível de edição é semelhante ao nível de edição estrutural livre. Porém as operações deste nível de edição possuem recursos adicionais que facilitam a edição específica de entidades que estão de acordo com um esquema XML. As visões da edição controlada por esquema facilitam a edição das entidades definidas no modelo XML e dos tipos de entidades definidos em um esquema XML.

5.3.3.2 Nos ADBGs

A edição controlada por alfabeto permite a edição de uma sentença visual guiada por um alfabeto. Visões sintáticas deste modo de edição mostram uma sentença visual destacando as entidades definidas no modelo de sentença visual e os tipos definidos no referido alfabeto.

5.3.3.3 Realização no SV-XML

No SV-XML, as operações em visões sintáticas da edição controlada por alfabeto são realizadas por operações de edição controlada por esquema do XML.

Esta correspondência justifica-se por que:

- as sentenças visuais SV-XML são realizados por documentos XML que representam sentenças visuais;
- os alfabetos SV-XML são realizados por esquemas XML que descrevem um tipo de sentença visual.
- as operações de edição controlada por esquema do XML destacam e facilitam a edição entidades do modelo de sentença visual do SV-XML, que é o próprio modelo XML. Adicionalmente destacam e facilitam a edição de tipos definidos no alfabeto SV-XML, que é um esquema XML.

Tanto as visões sintáticas do modo de edição controlado por esquema quanto as visões do modo de edição controlada por esquema do XML são visões sintáticas. Ou seja mostram estruturas de dados e não o que os dados representam.

As operações de edição controlada por esquema do XML são guiadas por um esquema XML. Elas auxiliam a edição, visualização e análise da estrutura de dados de documentos deste esquema, independentemente da área de domínio do esquema. Portanto não importa se o esquema descreve um tipo de sentença visual ou qualquer outro tipo de informação. As mesmas operações podem ser aplicadas em documentos de esquemas de qualquer domínio.

Assim as ferramentas, editores e outros componentes do modo de edição controlada por esquema do XML podem ser reutilizados na realização das operações do modo de edição controlada por alfabeto de ADBGs.

5.3.4 Edição controlada por semântica

5.3.4.1 No XML

A edição controlada por semântica do XML permite a edição de um documento XML guiada por um esquema XML mais um conjunto de regras relacionadas com a semântica das entidades deste esquema. Entidades que representam gráficos são editadas como gráficos, entidades que representam fórmulas matemáticas são editadas como fórmulas matemáticas. Logo este modo de edição é dependente da área de domínio do esquema que controla a edição.

5.3.4.2 Nos ADBGs

A visão semântica da edição controlada por alfabeto permite a edição de uma sentença visual guiada por um alfabeto e um conjunto de regras relacionadas à

semântica das sentenças visuais. Tais visões tratam uma sentença visual como uma tela com um conjunto de símbolos gráficos.

A edição controlada por gramática permite a edição de uma sentença visual guiada por uma gramática. Este modo de edição possui visões semânticas e sintáticas.

A edição controlada por meta-modelo de gramática permite a edição de uma gramática guiada por um meta-modelo de gramática. Este modo de edição possui visões semânticas e sintáticas.

5.3.4.3 Realização no SV-XML

No SV-XML a edição controlada por gramática, a edição controlada por meta-modelo de gramática e a visão semântica da edição controlada por alfabeto são realizadas por edições controladas por semântica do XML.

Esta correspondência é justificada da seguinte forma.

As visões semânticas dos modos de edição controlada por alfabeto, controlada por gramáticas e controlada por meta-modelos de gramáticas do SV-XML são realizadas por edições controladas por semântica XML, pois qualquer visão semântica exige um conjunto extra de regras que irão definir a edição de entidades destes modelos de acordo com a semântica destas entidades.

As visões sintáticas dos modos de edição controladas por gramáticas e meta-modelos de gramáticas também são realizados por edições controladas por semântica XML pois gramáticas e meta-modelos de gramáticas são estruturas mais complexas que esquemas XML. Cada entidade de uma gramática pode ser representada por um conjunto de entidades XML. Por isso regras extras devem informar como visualizar e editar estas entidades segundo estruturas de dados apropriadas para destacar estruturas de gramáticas. Assim, estruturas de dados de gramáticas são mostradas como estruturas de dados de gramáticas. É importante lembrar que as edições estrutural livre e controlada por esquema do XML são apropriadas para destacar estruturas do modelo XML e não de gramáticas.

Assim as ferramentas, editores e outros componentes do modo de edição controlada por semântica do XML podem ser reutilizados na realização das operações dos modos de edição controlada por gramática, controlada por meta-modelo de gramática e da visão semântica da edição controlada por alfabeto.

É importante notar que no caso da edição controlada por semântica as possibilidades de reaproveitamento são menores. Nos modos anteriores sistemas inteiros podem ser reaproveitados em um ADBG baseado no SV-XML sem a necessidade de adaptações. Isto ocorre porque estas edições são independentes da semântica dos dados que estão sendo editados. Todas as operações destes modos de edição podem estar encapsuladas em editores e visualizadores padronizados do XML. Tais sistemas podem ser usados para editar e visualizar documentos XML de qualquer área de domínio, inclusive documentos XML que representam sentenças visuais e alfabetos.

No caso de edições controladas por semântica, as operações mais básicas das ferramentas e as operações relacionadas com as regras do esquema XML podem ser reaproveitadas. Porém as operações relacionadas com as regras extras devem ser definidas ou adaptadas caso a caso pois são dependentes da semântica dos dados.

5.3.5 Edição de esquema

5.3.5.1 No XML

A edição de esquema do XML permite a edição de esquemas XML guiada por um meta-modelo de esquema XML (como por exemplo, o modelo de DTDs ou o modelo de XML schemas).

5.3.5.2 Nos ADBGs

A edição controlada por meta-modelo de alfabeto permite a edição de um alfabeto guiada por um meta-modelo de alfabetos. Visões sintáticas deste modo de edição mostram um alfabeto destacando as entidades definidas no meta-modelo de alfabeto.

5.3.5.3 Realização no SV-XML

No SV-XML as operações em visões sintáticas da edição controlada por meta-modelo de alfabeto são realizadas por operações de edição de esquema do XML.

Justifica-se esta correspondência por que:

- Os alfabetos SV-XML são realizados por esquemas XML que especificam tipos de sentenças visuais.
- Os meta-modelos de alfabetos SV-XML são realizados por meta-modelos de esquema XML.

Tanto as visões do modo de edição controlada por meta-modelo de alfabeto quanto as visões do modo de edição de esquema do XML são visões sintáticas.

As operações de edição de esquemas XML auxiliam a edição, visualização e análise da estrutura de dados de esquemas XML independentemente da área de domínio do esquema. Por isso as operações que auxiliam a edição de esquemas XML que especificam tipos de sentenças visuais são as mesmas operações que auxiliam a edição de esquemas XML que especificam qualquer outro tipo de dado.

Assim as ferramentas, editores e outros componentes do modo de edição de esquemas XML podem ser reutilizados na realização das operações do modo de edição controlada por meta-modelo de alfabeto de ADBGs.

5.3.6 Modo de edição de esquema controlada por semântica

5.3.6.1 No XML

A edição de esquema controlada por semântica permite a edição de esquemas XML guiada por um meta-modelo de esquema XML e um conjunto de regras relacionadas com a semântica dos tipos de dados que estão sendo definidos.

5.3.6.2 Nos ADBGs

Visões semânticas da edição controlada por meta-modelo de alfabeto permitem a edição de alfabetos guiada por um meta-modelo de alfabetos e um conjunto de regras relacionadas à semântica dos alfabetos. Tais visões tratam um alfabeto como um conjunto de tipos de símbolos gráficos.

5.3.6.3 Realização no SV-XML

No SV-XML, as operações em visões semânticas da edição controlada por meta-modelo de alfabeto são realizados por operações de edição de esquema controlada por semântica do XML.

Justifica-se esta correspondência por que:

- os alfabetos SV-XML são realizados por esquemas XML;
- os meta-modelos de alfabetos SV-XML são realizados por meta-modelos de esquema;

As operações XML de edição de esquema controlada por semântica são capazes de editar um esquema XML de acordo com a semântica dos tipos de dados definidos no esquema. No SV-XML esta semântica denota um conjunto de símbolos gráficos.

Assim as ferramentas, editores e outros componentes do modo de edição de esquema XML controlada por semântica podem ser reutilizados na realização das operações em visões semânticas do modo de edição controlada por meta-modelo de alfabeto de ADBGs.

É importante notar que as possibilidades de reaproveitamento da edição de esquema XML controlada por semântica são menores que as possibilidades da edição de esquema XML. Assim como no caso da edição controlada por semântica (seção 5.3.4), as regras extras da edição de esquemas devem ser definidas ou adaptadas caso a caso pois são dependentes da semântica dos dados.

5.3.7 Síntese dos modos de edição do SV-XML

Cada modo de edição típica de ADBGs é realizado no SV-XML por um conjunto de operações que pertencem a um ou mais modos de operação XML. Na primeira coluna da tabela 5.2 são mostrados todos os modos de edição típicos de ADBGs. No SV-XML as operações destes modos de edição são realizadas por operações do modo de edição XML mostrados na segunda coluna da tabela 5.2.

Tabela 5.2: Modos de edição de ADBGs e correspondentes modos de edição XML que os realizam no SV-XML

Modos de edição de ADBGs		**Correspondentes modos de edição XML no SV-XML**
Entidades básicas		Código fonte
Controlada por modelo de sentença visual		Estrutural livre
Controlada por alfabeto	Visão Sintática	Controlada por esquema
	Visão Semântica	Controlada por semântica
Controlada por meta-modelo de alfabeto	Visão Sintática	Edição de esquema
	Visão Semântica	Edição de esquema controlada por semântica
Controlada por gramática	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica
Controlada por meta-modelo de gramática	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica

5.3.8 Mesclando os modos de edição

Uma sentença visual, alfabeto ou gramática do SV-XML pode ter várias visões de cada um dos seus seis modos de edição. Cada visão pode mostrar diferentes aspectos ou partes de um documento e todas as visões devem estar sincronizadas. Devido ao recurso de múltiplos espaços de nomes, em um mesmo documento podem ser aplicadas operações que pertencem a diferentes modos de edição. Ou seja, em algumas partes do documento podem ser aplicadas operações do modo de edição controlado por alfabeto e em outras são aplicadas apenas operações do modo de edição controlado por modelo de sentença visual.

Pode-se utilizar alternadamente as operações de diferentes modos de edição. Por exemplo, pode-se editar os caracteres de uma sentença visual no modo de edição de entidades básicas e após utilizar o verificador sintático do modo de edição controlado por modelo de sentença visual para verificar se as mudanças efetuadas estão de acordo com o modelo de sentença visual.

5.4 Realização dos subsistemas do SV-XML

Até este ponto foram dados detalhes sobre os componentes XML que realizam a estrutura de dados de um ambiente baseado no SV-XML e sobre os componentes XML que podem realizar as operações que manipulam estas estruturas. Nesta seção serão dados detalhes sobre os componentes XML que realizam os subsistemas que encapsulam estas operações. O conjunto destes subsistemas forma um ambiente de desenvolvimento baseado no framework SV-XML.

Ao mapear os modos de operações do SV-XML para os modos de operações XML tem-se a disposição uma grande quantidade de componentes e artefatos que podem ser utilizados na realização e implementação do ambiente SV-XML. Estes componentes e artefatos vão desde técnicas de desenvolvimento de softwares até APIs, ferramentas e

sistemas XML. Nesta seção serão abordados alguns sistemas XML que podem realizar subsistemas de um ambiente SV-XML. Estes sistemas são editores típicos do espaço tecnológico XML que já encapsulam operações e estruturas de dados que podem ser utilizadas em ADBGs. Para facilitar o uso de editores do XML em ambientes SV-XML esta tese propõe um mapeamento entre os editores típicos de ADBGs e editores do XML que podem realizar estes editores no SV-XML.

As informações deste mapeamento podem ser utilizadas de diversas maneiras: pode-se utilizar o editor XML do mapeamento para realizar o correspondente editor de ADBGs sem nenhuma alteração. Alternativamente pode-se estender o referido editor XML para atender a necessidades específicas de cada realização. Ou então pode-se simplesmente utilizar o mapeamento para compreender o funcionamento de um editor SV-XML.

Os editores que fazem parte de um ADBG são: editor de sentença visual, editor de alfabeto e o editor de gramática. A seção 2.3 mostrou as características e funcionalidades gerais destes editores. Ao apresentar o referido mapeamento esta seção informará características adicionais sobre as funcionalidades e características da realização e implementação destes editores em ambientes XML.

Na próxima seção será apresentado como editores do espaço tecnológico XML podem ser utilizados em ADBGs baseados no modelo SV-XML.

5.4.1 Editores de um ambiente baseado no SV-XML

A tabela 2.1 mostrou o mapeamento entre os editores ADBG, seus modos de edição e os documentos editados em cada editor. A tabela 5.2 mostrou um mapeamento entre os modos de edição típicos de ADBGs e os modos de edição XML que os realizam no SV-XML. Por fim, a tabela 4.1 mostrou o mapeamento entre os editores XML, seus modos de edição e os documentos editados em cada editor. Com as informações destes mapeamentos pode-se construir a tabela 5.3 e deduzir a partir dela que editores XML podem realizar cada editor típico de ADBGs.

A primeira coluna da tabela 5.3 mostra os editores típicos de ADBGs. A segunda coluna mostra os modos de edição que são utilizados por estes editores. Como um conjunto de operações está associado a cada modo de edição, através da segunda coluna pode-se deduzir as operações que fazem parte de cada editor. A terceira coluna mostra o documento que é editado em cada modo de edição.

A quarta coluna mostra o componente XML que realiza o documento ADBG editado. A quinta coluna mostra o modo de edição XML que realiza o correspondente modo de edição de ADBGs no SV-XML. Já a sexta coluna mostra o editor XML que é utilizado para encapsular as operações de cada modo de edição de cada editor do SV-XML.

É importante notar que nas 3 primeiras colunas da tabela podem ser obtidas informações sobre as entidades do ambiente de desenvolvimento. Já nas 3 últimas colunas podem ser obtidas informações sobre como as correspondentes entidades podem ser realizadas em um ambiente baseado no SV-XML, bem como informações sobre os componentes e artefatos que estão disponíveis para implementar estas entidades.

Tabela 5.3: Entidades de ADBGs e componentes XML que realizam estas entidades no SV-XML

Editores de ADBGs	**Modos de edição de ADBGs**		**Documento ADBG editado**	**Componente XML editado**	**Modos de edição do XML**	**Editores XML**
Editor de sentença visual	Entidade básica		Sentença visual, alfabeto, gramática	Doc. XML, esquema XML	Código fonte	Editor de código fonte
	Controlada por modelo de sentença visual		Sentença visual	Doc. XML	Estrutural livre	Editor XML
Editor de alfabeto	Controlada por alfabeto	Visão Sintática	Sentença visual	Doc. XML	Controlada por esquema	Editor XML controlado por esquema
		Visão Semântica	Sentença visual	Doc. XML	Controlada por semântica	Extensão de editor XML controlado por esquema
	Controlada por meta-modelo de alfabeto	Visão Sintática	Alfabeto	Esquema XML	Edição de esquema	Editor de esquema
		Visão Semântica	Alfabeto	Esquema XML	Edição de esquema controlada por semântica	Extensão de editor de esquema XML
Editor de gramática	Controlada por gramática	Visão Sintática	Sentença visual	Doc. XML	Controlada por semântica	Extensão de editor XML controlado por esquema
		Visão Semântica	Sentença visual	Doc. XML	Controlada por semântica	Extensão de editor XML controlado por esquema
	Controlada por meta-modelo de gramática	Visão Sintática	Gramática	Doc. XML	Controlada por semântica	Extensão de editor XML controlado por esquema
		Visão Semântica	Gramática	Doc. XML	Controlada por semântica	Extensão de editor XML controlado por esquema

Para exemplificar a leitura da tabela 5.3, a segunda linha de dados desta tabela pode ser interpretada da seguinte maneira: a edição de uma sentença visual através do modo de edição controlada por modelo de sentença visual de um editor de sentença visual é realizada no SV-XML pela edição de um documento XML através do modo de edição estrutural livre de um editor XML.

Comparando-se as colunas 1 e 6 pode-se saber quais editores XML podem realizar cada editor de um ADBG. Assim, analisando a tabela 5.3 pode-se descrever os editores de um ambiente baseado no SV-XML da seguinte forma.

O editor de sentença visual do SV-XML contém operações relacionadas com o modo de edição de entidades básicas e com o modo de edição controlado por modelo de sentença visual. As operações de entidades básicas são realizadas em um editor de código fonte. Já as operações controladas por modelo de sentença visual são realizadas em um editor XML. Portanto um editor de sentença visual do SV-XML é realizado por um editor de código fonte e um editor XML. Usualmente o editor de código fonte está contido no editor XML.

O editor de alfabeto do SV-XML contém operações relacionadas com o modo de edição controlada por alfabeto e com o modo de edição controlada por meta-modelo de alfabeto. As operações em visões sintáticas do modo de edição controlada por alfabeto são realizadas em um editor XML controlado por esquema. As operações em visões semânticas do modo de edição controlada por alfabeto são realizadas em uma extensão de editor XML controlado por esquema. Já as operações em visões sintáticas do modo de edição controlada por meta-modelo de alfabeto são realizadas em um editor de esquemas XML. As operações em visões semânticas do modo de edição controlada por meta-modelo de alfabeto são realizadas em uma extensão de editor de esquemas XML.

Portanto um editor de alfabeto do SV-XML é realizado por um editor XML controlado por esquema, uma extensão de editor XML controlado por esquema, um editor de esquemas XML e uma extensão de editor de esquemas XML. Usualmente o editor XML controlado por esquema está contido na extensão de editor XML controlado por esquema. Já o editor de esquemas XML está contido na extensão de editor de esquemas XML.

O editor de gramática do SV-XML contém operações relacionadas com o modo de edição controlada por gramática e com o modo de edição controlada por meta-modelo de gramática. As visões sintáticas e semânticas dos dois modos de edição são realizadas em extensões de editores XML controlados por esquema.

Os ambientes do SV-XML podem ter as mesmas variações dos demais ADBGs. Ou seja, podem existir ambientes sem alguns dos editores indicados. Ambientes que não tem suporte as operações relacionadas ao modo de edição controlado por meta-modelo de alfabeto, por exemplo, não contém editores de esquemas nem extensões de editores de esquemas. Ambientes sem suporte as operações relacionadas ao modo de edição de código fonte não contém o editor de código fonte e assim por diante.

Os editores podem ser combinados e estendidos de diversas maneiras. Podem existir desde arquiteturas onde todos os editores existem separadamente até arquiteturas onde todos editores estão contidos em um único editor que disponibiliza, mescla e sincroniza várias visões de todos os modos de edição.

Enfim, o uso do framework é modular e flexível. Por definição, o modelo de sentença visual de um ADBG que segue as recomendações do framework SV-XML é

sempre o modelo XML e a sua realização é uma especificação XML. Todas as demais estruturas de dados, operações e sistemas do framework SV-XML são opcionais. O uso do framework SV-XML pode ser parcial e combinado com outros componentes XML e até mesmo com componentes fora do espaço tecnológico XML. Contudo quando alguma estrutura de dados, operação ou sistema do framework SV-XML não é utilizada pode-se perder algumas de suas vantagens e características.

6 IMPLEMENTAÇÃO – VITRANSF

Como prova de conceito, as estruturas, operações e sistemas propostos nesta tese foram implementados em um protótipo. O objetivo desta implementação foi demonstrar concretamente a viabilidade e aplicabilidade de um ambiente SV-XML. O ambiente SV-XML implementado chama-se Vitransf (Visual Interface Transformation) (TELECKEN, 2008). Ele é formado por um ambiente de desenvolvimento e um ambiente de execução.

O ambiente de desenvolvimento reutilizou o editor Amaya como base de sua implementação. O Amaya é um editor XML oficial da W3C (QUINT, 2007; QUINT, 2004) que integra um conjunto de editores XML padrão (editores XML, XML controlados por esquema etc). No Vitransf alguns destes editores foram reutilizados sem alterações, outros foram estendidos para atender as necessidades específicas de um ADBG.

Com a ajuda destes editores e suas visões o desenvolvedor especifica uma gramática VCARW. Os dados desta gramática são armazenados em um documento XML. Um conjunto de rotinas do Vitransf pode interpretar este documento XML e gerar o sistema visual interativo especificado na gramática.

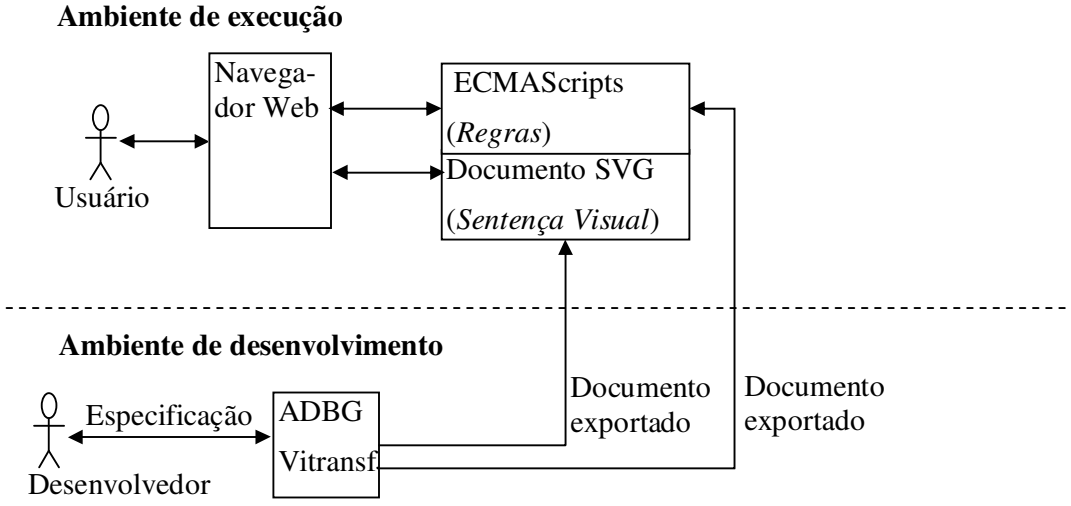


Figura 6.1: Arquitetura da implementação

O sistema visual interativo gerado é um documento SVG (um documento XML)(FERRAIOLO, 2003) e um conjunto de ECMAScripts. O documento SVG descreve gráficos 2D, os scripts descrevem alterações que podem ocorrer nestes gráficos. As

alterações são disparadas por eventos XML que podem ocorrer na interação entre o usuário e o documento SVG. Navegadores Web padrão com suporte a SVG e ECMAScripts podem interpretar os documentos e scripts gerados pelo ambiente de desenvolvimento. Portanto, navegadores Web padrão, como o Internet Explorer e o Mozilla, são os ambientes de execução do Vitransf. A figura 6.1 mostra a arquitetura do Vitransf.

As próximas seções irão detalhar o ambiente de desenvolvimento do Vitransf. Em seguida serão mostrados dois estudos de caso onde sistemas visuais interativos são projetados no ADBG Vitransf e utilizados em navegadores Web. Por fim esta implementação será analisada.

6.1 Ambiente de desenvolvimento – Estrutura de dados

6.1.1 O modelo de sentença visual

O **modelo de sentença visual** do Vitransf é uma especificação XML implementada no Amaya. Esta especificação é uma realização do modelo SV-XML, ou seja, do modelo XML interpretado como modelo de sentença visual.

As **sentenças visuais** do Vitransf são documentos XML bem formados.

6.1.2 O alfabeto

O alfabeto do Vitransf é fixo. Ou seja, pode-se especificar várias gramáticas que poderão gerar vários sistemas visuais interativos, porém todas gramáticas devem utilizar o mesmo alfabeto. Este alfabeto é um subconjunto da linguagem SVG (FERRAIOLO, 2003).

A linguagem ***Scalable Vector Graphics*** **(SVG**) (FERRAIOLO, 2003) é uma linguagem XML que descreve gráficos 2D. Esta especificação tornou-se em 4 de setembro de 2001 uma recomendação oficial do World Wide Web Consortium (W3C).

Diferentemente de formatos como JPEG e PNG que representam um gráfico descrevendo-o *pixel* a *pixel,* o SVG descreve um gráfico através de vetores. Vetores descrevem figuras geométricas através de elementos e atributos.

A figura 6.2 mostra um exemplo de código SVG e ao lado o gráfico descrito neste código.

Junto com suas figuras geométricas, a representação vetorial do SVG permite que figuras (JPEG, por exemplo) e textos sejam adicionados a um gráfico. Os textos adicionados podem ser copiados e manipulados como figuras ou editados como textos comuns. Estes textos, assim como todo o código SVG podem ser interpretados por um navegador web e manipulados por ECMAScripts (e.g. Java Script, Jscript) em páginas Web.

A estrutura de documentos SVG está definida no esquema XML mostrado em (FERRAIOLO, 2003). Editores e visualizadores que tem um suporte total ao SVG podem editar e apresentar uma visão de documentos SVG que estão de acordo com este esquema XML. Entretanto muitos editores e visualizadores SVG tem um suporte parcial ao SVG. Assim somente algumas estruturas do SVG podem ser editadas ou apresentadas nestes editores. Estas estruturas, que são subestruturas do SVG completo, também estão especificadas em esquemas XML alternativos. Este é o caso do Amaya. O

Amaya define uma estrutura de documentos SVG básicos. Documentos que estão de acordo com o esquema SVG básico do Amaya podem ser editados e visualizados nas visões semânticas do Amaya.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-
  20010904/DTD/svg10.dtd">
<svg width="816" height="653">
     <rect x="73" y="65" width="102" height="67"
       fill="rgb(0,0,255)"  stroke="rgb(0,0,0)" stroke-width="1"/>
     <circle cx="274" cy="104.5" r="30" fill="rgb(0,0,255)"
     stroke="rgb(0,0,0)" stroke-   width="1"/>
     <ellipse cx="457" cy="136" rx="16" ry="52" fill="rgb(0,0,255)"
     stroke="rgb(0,0,0)" stroke-width="1"/>
     <line x1="175" y1="330" x2="341" y2="297"stroke="rgb(0,0,0)"  stroke-width="1"/>
     <path d="M414.667 437.167 L310.667 475.833 L356 391.833 L394.667"
       fill="rgb(0,0,255)" stroke="rgb(0,0,0)" stroke-width="1"/>
</svg>
```

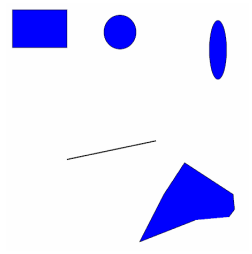


Figura 6.2: Exemplo de código SVG

O **alfabeto Vitransf** é o esquema SVG básico do Amaya (QUINT, 2007). Cada tipo de elemento definido neste esquema deve ser interpretado como um tipo de símbolo que pode ser colocado em uma sentença visual. Cada tipo de atributo definido neste esquema deve ser interpretado como um tipo de atributo de um símbolo e assim por diante.

A interpretação/semântica dos símbolos do Vitransf está de acordo com o padrão SVG (FERRAIOLO, 2003). A figura 6.2 mostra, à esquerda, um código fonte XML que esta de acordo com o alfabeto Vitransf. Esta estrutura de dados descreve 6 símbolos. A direita da figura 6.3 é mostrada a interpretação/semântica destes 6 símbolos. Dos 6 símbolos apresentados somente 5 tem uma representação gráfica. O símbolo <svg> não possui representação gráfica. Ele representa a sentença visual e contém todos os demais símbolos desta sentença visual.

Os elementos SVG têm atributos relacionados a eventos tais como *onmouseover*, *onload* ou *onclick* (atributo relacionado com o evento de clicar em um elemento SVG). O padrão SVG especifica que quando os eventos relacionados a estes atributos ocorrem a ação que está especificada no valor destes atributos é executada (associação *in-line*).

Assim os eventos que podem ocorrer em um tipo de símbolo do alfabeto Vitransf estão definidos na lista de possíveis atributos deste símbolo. O tipo *circle* , por exemplo, pode ter o atributo *onclick*. Isto significa que o evento de clicar o círculo pode ser monitorado pelo ambiente de execução. Caso este evento ocorra em tempo de execução, uma ação pode ser disparada. No caso do Vitransf a ação executará um script que realiza as transformações definidas em uma regra da gramática. A sintaxe e semântica destes atributos estão definidas na especificação SVG.

Como foi visto, o reuso do esquema SVG e de sua semântica foi muito conveniente, pois o esquema SVG já possui estruturas que descrevem duas características essenciais

dos símbolos de uma sentença visual: a sua aparência e os seus comandos (eventos+ações).

O **meta-modelo de alfabeto do Vitransf** é o modelo da linguagem S (*Structure language*). S assim como a DTD e a XML schema é uma linguagem para especificar estruturas de documentos XML. Ele tem os mesmos objetivos que uma DTD porém possui uma sintaxe um pouco diferente. Pode-se facilmente traduzir documentos S para uma DTD equivalente e vice-versa. O S foi desenvolvido no projeto THOT (QUINT, 2000) e adotado pela equipe de desenvolvimento do software Amaya. Como o meta-modelo de alfabeto do Vitransf é o modelo da linguagem S, o alfabeto Vitransf é um esquema XML da linguagem S.

6.1.3 A gramática

As gramáticas do Vitransf são gramáticas VCARW. O anexo A descreve em detalhes o modelo formal destas gramáticas. Através de suas visões o Vitransf provê uma interface amigável para a especificação destas gramáticas. Nesta tese as gramáticas VCARW implementadas no Vitransf serão chamadas de gramáticas Vitransf. Em resumo uma gramática Vitransf contém uma sentença inicial e um conjunto de regras chamadas de regras Vitransf.

A sentença visual inicial é a tela inicial do sistema que está sendo projetado. Cada transformação nesta tela é disparada por um evento descrito na sentença visual. Cada evento chama uma regra. Uma regra é usada para transformar uma sentença visual em outra através da criação, exclusão ou alteração de símbolos. As sentenças visuais resultantes também descrevem os eventos que poderão transformar novamente estas sentenças visuais.

As **gramáticas Vitransf** são armazenadas em documentos XML. A sentença visual inicial fica armazenada em um documento XML que está de acordo com o alfabeto Vitransf. As regras são armazenadas em documentos XML que estão de acordo com um esquema XML chamado **esquema das regras Vitransf**. O esquema das regras Vitransf foi escrito na linguagem S.

As regras Vitransf podem citar tipos de elementos definidos no alfabeto Vitransf, por isso o esquema das regras Vitransf possui a especificação de todos os tipos de símbolos definidos no alfabeto Vitransf (o esquema SVG básico) mais um conjunto de tipos específicos das regras Vitransf.

O **meta-modelo de gramática do Vitransf** está definido em dois esquemas XML, o alfabeto Vitransf que especifica a estrutura das sentenças visuais iniciais e o esquema das regras Vitransf que especifica a estrutura das regras da gramática. Os dois esquemas foram escritos e implementados no Vitransf através da linguagem S.

6.1.3.1 Especificação de gramáticas através de alfabeto fixo com suporte a múltiplos espaços de nomes

Como foi visto na seção anterior, a sentença visual inicial e as regras Vitransf são documentos XML que só podem conter os tipos de elementos XML definidos no alfabeto Vitransf. Portanto, o alfabeto do Vitransf é fixo. Ou seja, pode-se especificar várias gramáticas que poderão gerar vários sistemas visuais interativos, porém todas gramáticas só podem utilizar os tipos de elementos definidos no mesmo alfabeto.

O alfabeto fixo pode ter dois inconvenientes. Primeiramente, uma determinada aplicação pode precisar de um determinado tipo de símbolo que não faz parte do alfabeto. Neste caso tal aplicação não poderia ser especificada através do referido alfabeto fixo. Outra inconveniência é o fato de uma aplicação precisar apenas de um determinado subconjunto de tipos definidos no alfabeto. Os símbolos que não são úteis apenas atrapalhariam a edição da gramática pois constariam como possíveis símbolos porém não poderiam ser utilizados.

Um trabalho futuro interessante seria tornar o alfabeto configurável. Neste caso a estrutura dos tipos de símbolos de um alfabeto, a sua representação gráfica e os seus comandos seriam especificados através de uma visão apropriada (uma visão controlada por meta-modelo de alfabeto). Após esta especificação, o editor de gramática interpretaria o alfabeto e geraria visões que permitiriam somente a edição de tipos especificados no alfabeto projetado.

Porém o Vitransf utiliza outra abordagem para flexibilizar o seu alfabeto fixo. O Vitransf explora o seu suporte a múltiplos espaços de nomes.

Este suporte permite a adição de elementos e atributos de outras linguagens XML em sentenças visuais iniciais e regras. O desenvolvedor pode especificar atributos e elementos não definidos no vocabulário SVG e adicioná-los nas sentenças visuais inicias e nas regras Vitransf.

A adição deve estar de acordo com o padrão namespace da W3C (BRAY, 2006b). Ou seja, para cada nova entidade que é adicionada nas sentenças visuais iniciais ou regras, deve-se informar o seu contexto. Se o contexto é conhecido esta nova entidade é interpretada de acordo com o esquema XML do seu contexto. Se o contexto é desconhecido a nova entidade é interpretada como uma entidade bem-formada.

Os alfabetos fixos com suporte a múltiplos espaços de nomes são tão expressivos quanto os alfabetos configuráveis. Tal fato resolve a primeira inconveniência dos alfabetos fixos. No caso dos alfabetos fixos com suporte a múltiplos espaços de nomes, qualquer tipo de entidade pode ser diretamente adicionado em uma regra ou sentença. Nesta situação a entidade seria adicionada como uma entidade bem formada, i.e. uma entidade que está de acordo com o modelo de sentença visual.

No caso de alfabetos configuráveis qualquer tipo de entidade pode ser adicionada a um alfabeto. Depois, esta entidade pode ser adicionada a uma regra ou sentença. Nesta situação a entidade seria adicionada como uma entidade do alfabeto projetado.

Porém, os alfabetos fixos com suporte a múltiplos espaços de nomes têm uma desvantagem em relação aos alfabetos configuráveis. Quando uma entidade bem formada é adicionada diretamente em uma regra ou gramática perde-se as facilidades de se utilizar um alfabeto. Por exemplo, quando um símbolo de um alfabeto é adicionado a uma sentença ele é adicionado com todos seus atributos já previamente definidos. Adicionalmente os atributos já vêm preenchidos com valores obrigatórios ou padronizados.

Quando os símbolos bem formados são adicionados diretamente nas sentenças o desenvolvedor é responsável por informar todos os atributos e valores. Em outras palavras ele não pode utilizar as facilidades do modo de edição controlada por alfabeto quando adiciona uma entidade bem formada diretamente na regra ou na sentença visual inicial. Ao invés disto, para adicionar elementos e atributos diferentes dos elementos e

atributos do SVG, ele apenas pode utilizar as facilidades do modo de edição controlado por modelo de sentença visual.

6.1.4 Verificação sintática

Para verificar se uma sentença visual está de acordo como o modelo de sentença visual do Vitransf basta verificar se a sentença visual é um documento XML bem formado.

Para verificar se uma sentença visual está de acordo com o alfabeto do Vitransf basta verificar se esta sentença está de acordo com o esquema SVG básico implementado no Amaya. O Vitransf suporta múltiplos espaços de nomes, portanto diferentes partes do documento podem ser verificadas por diferentes esquemas. Algumas podem ser verificadas pelo esquema *default* que é o alfabeto Vitransf, outras partes podem ser validadas de acordo com outros esquemas XML ou podem ser apenas bem formadas. A especificação de qual parte do documento deverá ser validada por qual esquema deve estar de acordo com a especificação namespace 1.1 da W3C (BRAY, 2006b).

No Vitransf não foram implementados mecanismos que verificam se uma sentença visual pertence a uma dada gramática.

Para verificar se um alfabeto está de acordo com o meta-modelo de alfabeto basta verificar se o alfabeto é um esquema S válido. Como o alfabeto Vitransf é fixo, esta operação não é utilizada.

Para verificar se uma gramática está de acordo com o meta-modelo de gramática basta verificar se a sentença visual inicial da gramática existe e está de acordo com o alfabeto Vitransf. Adicionalmente deve-se verificar se as regras da gramática estão de acordo com o esquema de regras Vitransf.

6.1.5 Síntese das estruturas de dados do Vitransf

A tabela 6.1 mostra na primeira coluna as estruturas de dados típicas de ADBGs. Na segunda coluna é mostrado o componente XML que realiza estas estruturas no ADBG Vitransf. Entre parênteses é dada uma definição mais específica de cada estrutura.

Todas as estruturas de dados do Vitransf (modelos, meta-modelos, sentenças visuais, alfabetos e gramáticas) são armazenadas e definidas através de recursos do espaço tecnológico XML (modelo XML, documentos XML, esquemas XML e meta-modelos de esquemas XML). Adicionalmente estes recursos estão em conformidade com as especificações da seção 5.2 que caracteriza ADBGs baseados no modelo SV-XML.

Tabela 6.1: Estruturas de ADBGs e correspondentes estruturas no Vitransf

Estrutura de dados de ADBGs	**Correspondente componente no Vitransf**
Modelo de sentença visual	**Especificação XML**
Sentença visual	**Documento XML** (bem formado)
Meta-modelo alfabeto	**Meta-modelo de esquema XML** (modelo da linguagem S)
Alfabeto	**Esquema XML** (Alfabeto Vitransf)
Meta-modelo de gramática	**Esquema XML** (Esquema das regras Vitransf para as regras. Alfabeto Vitransf para as sentenças visuais iniciais)
Gramática	**Documento XML** (As regras Vitransf devem estar de acordo com o esquema de regras Vitransf. As sentenças visuais iniciais devem estar de acordo com o alfabeto Vitransf. Os dois tipos de documentos admitem a inclusão de entidades bem formadas e de outros alfabetos - suporte a múltiplos espaços de nomes)

6.2 Ambiente de desenvolvimento - Operações e sistemas

A base de implementação do Vitransf foi o software Amaya (QUINT, 2007, QUINT 2004). O Amaya é um ambiente de edição que integra um grande conjunto de editores XML (com visões sintáticas e semânticas). Este software está sendo desenvolvido pela W3C desde 1996, foi escrito em C e C++ e utiliza/implementa vários recursos XML. As próximas seções mostram como os diferentes modos de edição do Vitransf reutilizaram e/ou estenderam os recursos e editores XML implementados no Amaya.

6.2.1 Modo de edição de entidades básicas

As sentenças visuais e as gramáticas do Vitransf são armazenadas em documentos XML. Estes documentos podem ser editados através de uma **visão do modo de edição de entidades básicas** do Vitransf. Esta visão permite a edição de documentos XML como se fossem textos. Nesta visão o usuário pode inserir e excluir caracteres livremente, ou seja, sem nenhuma verificação sintática. Portanto esta é uma visão do modo de edição de entidades básicas. Tal visão é mostrada na figura 6.3.

A visão do modo de edição de entidades básicas do Vitransf é o editor de texto do Amaya. Nenhuma alteração foi feita neste editor para que ele fosse adaptado ao Vitransf.

Também pode-se utilizar outros editores de textos fora do Vitransf. Após a edição nestes editores externos, os arquivos XML editados podem ser interpretados e utilizados no Vitransf como se tivessem sido editados no próprio Vitransf.

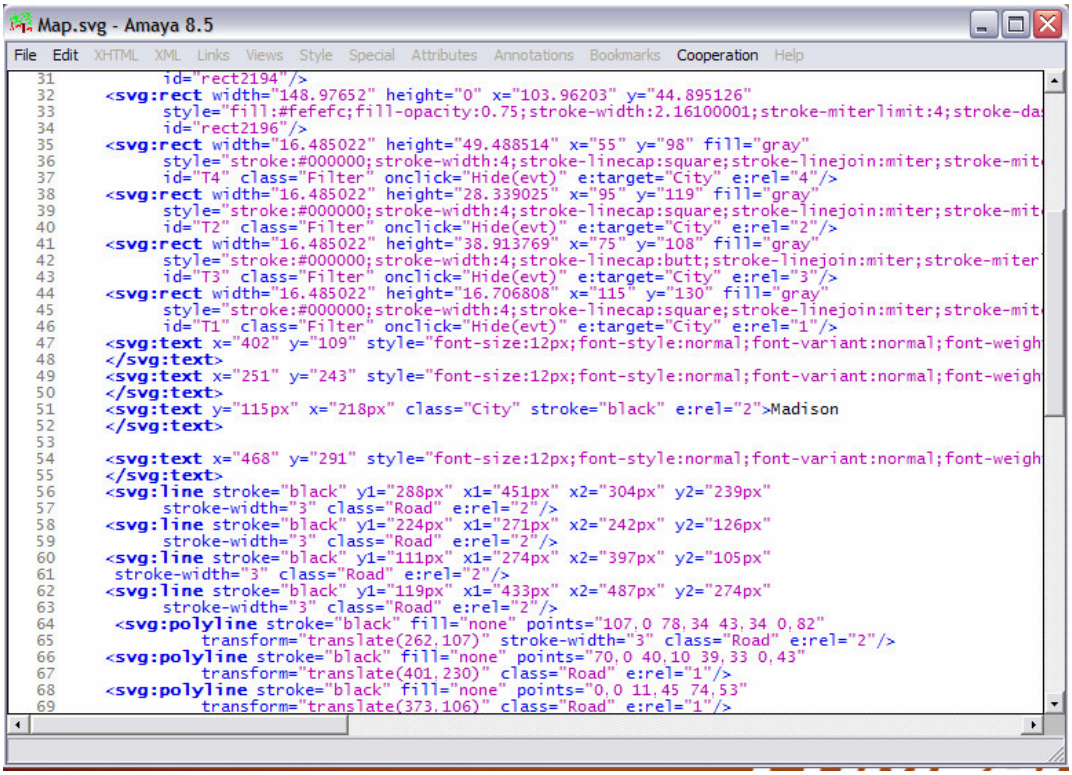


Figura 6.3: Visão do modo de edição de entidades básicas do Vitransf

6.2.2 Modo de edição controlada por modelo de sentença visual

As sentenças visuais do Vitransf são armazenadas em documentos XML. O Vitransf permite a edição destes documentos através de uma visão estruturada. Nesta visão a estrutura hierárquica do XML é destacada. Abaixo de cada elemento XML é colocada uma barra azul vertical, ao lado desta barra são colocados os filhos deste elemento. Os filhos também possuem suas próprias barras e filhos. Assim as relações hierárquicas entre pais e filhos são destacadas através de barras e endentações. A estrutura do modelo XML também é destacada através de cores e estilos de letras. Os nomes de elementos estão escritos em azul escuro, os atributos estão escritos em azul claro, os valores de atributos estão escritos em rosa e os textos (elementos básicos) estão escritos em preto.

Adicionalmente esta visão disponibiliza operações de edição que auxiliam o desenvolvedor a manter a coerência sintática do documento levando-se em conta o modelo de sentença visual. Para exemplificar, o desenvolvedor não consegue inserir caracteres livremente em qualquer ponto do documento. Ele só consegue inserir/excluir caracteres nos valores de atributos e nos textos (elementos básicos). Se o desenvolvedor tentar excluir um caractere do nome de um atributo todo atributo e o seu valor é selecionado. Depois de selecionado, o desenvolvedor pode escolher se quer deletar todo atributo ou deixá-lo sem alterações. Se o desenvolve tentar excluir um caractere do nome de um elemento, todo elemento e seus filhos são selecionados. Depois de selecionar todo elemento o desenvolvedor pode escolher se quer deletá-lo ou deixá-lo sem alterações. Para inserir elementos e atributos o desenvolvedor deve chamar, através de menus ou do teclado, caixas de dialogo apropriadas para estas finalidades. Não

importa a alteração que seja feita nos dados através desta visão, o resultado final sempre será um documento que está de acordo com o modelo XML.

A figura 6.4 mostra a visão estruturada do Vitransf, esta é uma **visão controlada por modelo de sentença visual** (no caso do Vitransf o modelo de sentença visual é o modelo XML)

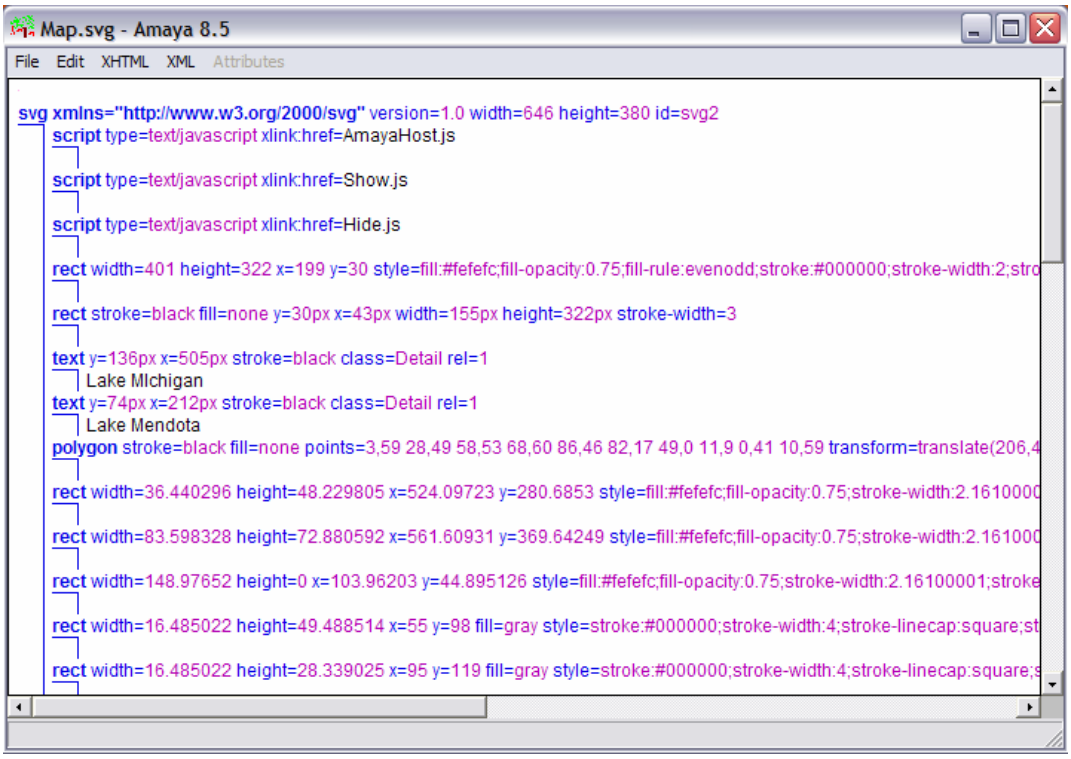


Figura 6.4: Visão controlada por modelo de sentença visual do Vitransf

A visão do modo de edição controlado por modelo de sentença visual do Vitransf é o editor XML do Amaya. Nenhuma alteração foi feita neste editor para que ele fosse adaptado ao Vitransf.

Também pode-se utilizar editores XML externos. Estes editores oferecem as mesmas vantagens do editor XML do Amaya, ou seja ajudam a escrever documentos bem formados. Depois da edição, os arquivos XML editados em editores externos podem ser novamente interpretados pelo Vitransf.

Se uma sentença visual editada em editor externo não estiver coerente com o modelo de sentença visual (o modelo XML), esta visão mostra os erros e auxilia na sua localização e correção.

6.2.3 Modo de edição controlada por alfabeto

O Vitransf possibilita a edição de sentenças visuais através de uma **visão sintática controlada por alfabeto**. Esta visão é semelhante à visão controlada por modelo de sentença visual, porém ela leva em conta as regras do modelo XML e do alfabeto Vitransf. Por isso esta visão contém recursos que facilitam a edição de sentenças visuais de forma a deixá-las de acordo com o alfabeto Vitransf.

Para exemplificar um destes recursos, a figura 6.5 mostra a visão estruturada controlada por alfabeto. Nesta visão, quando o usuário seleciona um elemento ele pode inserir um atributo. Para inserir um atributo ele deve clicar no menu "*attribute*" da visão. Este menu vai mostrar ao usuário os atributos que podem ser inseridos neste elemento. Ao escolher que atributo deve ser adicionado, a visão mostra ao desenvolvedor uma caixa de dialogo apropriada para se preencher o valor deste atributo. Ao preencher o valor do atributo o atributo é inserido no elemento previamente selecionado. No exemplo da figura 6.5 foi selecionado o elemento "*text*", em seguida foram mostrados os atributos que podem ser inseridos neste tipo de elemento. Dentre os atributos que podiam ser inseridos foi escolhido o atributo "*visibility*". Após esta escolha foi mostrado ao desenvolvedor a caixa de dialogo apropriada. Na estrutura do alfabeto Vitransf foi especificado que o valor do atributo *visibility* deveria ser um dos seguintes valores: *visible, hidden* ou *inherit*. Por isso a caixa de dialogo mostrada só permite que o desenvolvedor escolha um destes três valores. Assim que o desenvolvedor escolher um dos três valores, o atributo e o seu valor serão inseridos no elemento *text* selecionado.

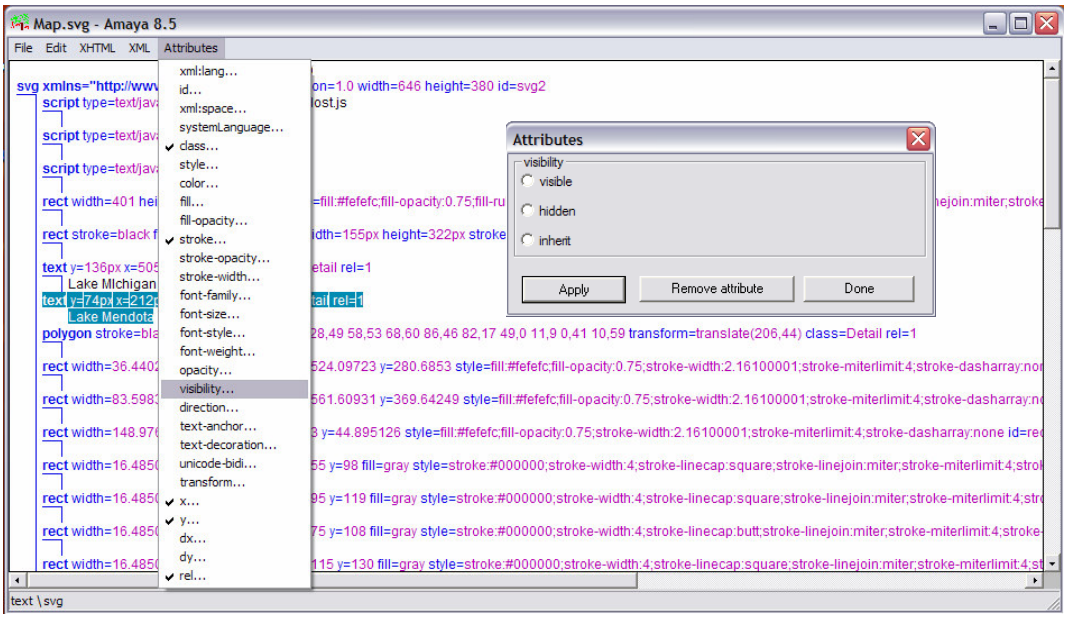


Figura 6.5: Visão sintática do modo de edição controlado por alfabeto do Vitransf

Um ponto interessante desta visão é que ao chamar a visão estruturada, o Vitransf escolhe automaticamente se a visão será controlada pelo modelo de sentença visual ou pelo alfabeto. Se o contexto do documento indicar um esquema XML correto e que possa ser interpretado pelo Vitransf, a visão será controlada pelo correspondente alfabeto. Se o contexto não indicar um esquema ou indicar um esquema desconhecido, a visão será controlada pelo modelo de sentença visual (o modelo XML).

Como o Vitransf suporta múltiplos espaços de nomes, uma parte do documento pode ser controlada por um esquema, outra parte pode ser controlada por outro esquema e uma terceira parte pode ser controlada pelo modelo XML. O contexto de cada parte do documento é que definirá como a edição de cada parte será controlada.

Se uma parte da sentença visual não estiver coerente com o modelo indicado em seu contexto, esta visão mostra os erros e auxilia na sua localização e correção.

A visão sintática do modo de edição controlado por alfabeto do Vitransf é o editor XML controlado por esquema do Amaya. Este editor interpreta um esquema XML S e gera uma visão que irá auxiliar a edição de documentos de acordo com este esquema. Um esquema S descrevendo o alfabeto do Vitransf foi disponibilizado ao Amaya, que o interpreta e gera uma visão adequada cada vez que o contexto de um documento informa que precisa estar de acordo com este esquema. Nenhuma alteração foi feita no editor XML controlado por esquema do Amaya para que ele fosse adaptado ao Vitransf.

Editores XML externos controlados por esquema também podem ser utilizados para esta mesma tarefa oferecendo as mesmas vantagens. Para isso basta fornecer a estes editores o esquema XML que descreve o alfabeto Vitransf. Como a linguagem S não é muito difundida, a melhor alternativa é traduzir o alfabeto Vitransf para a linguagem DTD e então fornecer esta DTD aos editores externos. Feito isto qualquer arquivo XML editado nestes editores externos pode ser integrado e interpretado no Vitransf.

O Vitranf também possui uma **visão semântica controlada por alfabeto**. Nesta visão pode-se editar os símbolos de uma sentença visual conforme a sua semântica, ou seja conforme a aparência e comportamento destes símbolos em ambientes de execução.

A figura 6.6 mostra esta visão, nela o desenvolvedor pode desenhar a sentença visual através de recursos gráficos e ao final deste processo o documento é salvo em seu formato XML. Para incluir símbolos na sentença o desenvolvedor utiliza uma paleta de símbolos, somente os símbolos que estão na paleta é que podem ser inseridos na sentença através da visão semântica.

Para adicionar símbolos e atributos de outros alfabetos deve-se utilizar as visões sintáticas. Como o Vitransf não tem a implementação da semântica dos símbolos de outros alfabetos, eles não têm uma representação na visão semântica. Atributos e símbolos de outros alfabetos são ignorados na visão semântica do Vitransf.

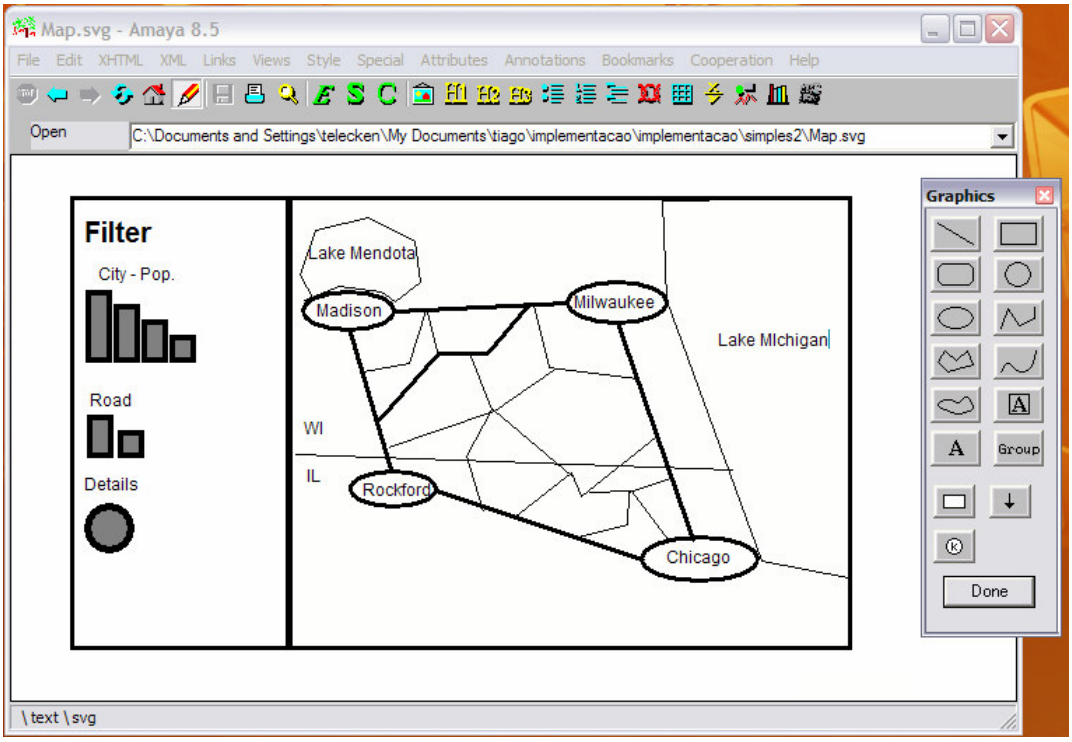


Figura 6.6: Visão semântica controlada por alfabeto do Vitransf

A visão semântica do modo de edição controlada por alfabeto do Vitransf é uma extensão do editor XML controlado por esquema do Amaya. Esta extensão provê uma visão semântica de documentos SVG, de acordo com as especificações SVG. Como o Vitransf também adotou a semântica do SVG (não apenas a sua estrutura), a visão semântica de documentos SVG do Amaya pode ser reutilizada sem nenhuma alteração.

É raro encontrar dois editores SVG semânticos com o mesmo suporte SVG. Por isso para este tipo de visão não é aconselhável a utilização de editores externos. A visão de um documento SVG em um editor SVG semântico externo pode ser diferente da visão deste mesmo documento no editor SVG semântico do Amaya. Isto ocorre porque alguns atributos e elementos que tem suporte semântico em outros editores podem não ter no Amaya. Diferentemente do esquema SVG que pode ser exportado e utilizado por outros editores, o suporte semântico do SVG não pode ser exportado.

6.2.4 Modo de edição controlada por meta-modelo de alfabeto

O Vitransf não possui um modo de edição controlada por meta-modelo de alfabeto. Este modo de edição permite a criação e edição de alfabetos. Como o alfabeto do Vitransf é fixo, não há a necessidade deste modo de edição

6.2.5 Modo de edição controlada por gramática

Um sistema visual interativo gerado no Vitransf é um documento SVG (uma cópia da sentença visual inicial) e um documento ECMAScript que contém scripts que podem transformar a sentença visual inicial conforme as transformações previstas na gramática. O documento SVG e o script podem ser interpretados por navegadores web (IE, Mozilla,...) com suporte a ECMAScripts e SVG. Portanto estes navegadores web são os ambientes de execução do Vitransf. Pode-se utilizar estes ambientes sem a necessidade de nenhuma adaptação.

O ambiente de execução do Vitransf também pode ser utilizado como uma **visão semântica controlada por gramática**. A seção 6.3 (estudos de casos) mostra telas do ambiente de execução que podem ser consideradas como visões semânticas controladas por gramática.

O Vitransf não tem a implementação de uma **visão sintática controlada por gramática**. Entretanto uma operação típica desta visão está implementada. A operação que gera um sistema visual interativo a partir de uma gramática Vitransf é uma operação controlada por gramática. Esta operação foi implementada no Vitransf e é acessada a partir de um menu do Vitransf. Ao acionar o referido item do menu, esta operação acessa a sentença visual inicial e as regras de uma gramática. A partir destes documentos o sistema visual interativo é gerado. Se houver inconsistências na gramática os erros são informados ao usuário. A sentença visual inicial e suas regras devem estar em um diretório específico para serem interpretadas por esta operação.

6.2.6 Modo de edição controlada por meta-modelo de gramática

Uma gramática possui uma sentença visual inicial e um conjunto de regras. Cada uma destas partes da gramática tem suas próprias visões e é editada a parte.

A sentença visual inicial é uma sentença visual, por isso pode ser editada através dos modos de edição de entidades básicas, controlado por modelo de sentença visual ou controlado por alfabeto.

As regras não possuem uma visão sintática no Vitransf. Porém foi proposta e implementada uma visão semântica para estas regras. Esta visão é descrita na seção 6.2.6.1.

Esta foi a única visão que exigiu extensões nos editores originais do Amaya. Todo o comportamento da visão semântica das regras Vitransf foi implementado. Isto inclui rotinas de visualização e edição. Ao final de cada edição a regra é armazenada em um documento XML que esta de acordo com a estrutura definida no esquema de regras Vitransf.

Nesta adaptação foram reutilizadas as rotinas de visualização de elementos SVG, a API XML e a API SVG já implementadas no Amaya. Portanto uma adaptação foi feita nesta visão porém muitos recursos XML básicos foram reaproveitados. Além disto todas as operações implementadas nesta visão podem ser consideradas como operações XML do modo de edição controlada por semântica.

6.2.6.1 Visão semântica das regras Vitransf

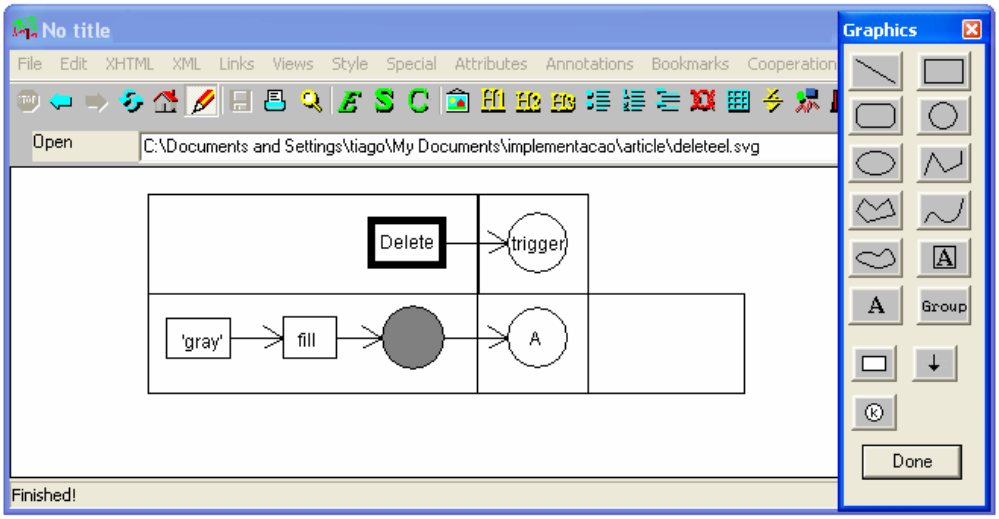


Figura 6.7: Visão semântica das regras Vitransf com a área de trabalho (à esquerda) e uma paleta Vitransf (à direita). Os últimos três botões da paleta são usados para adicionar os símbolos Vitransf: nodos, setas e token. Os demais são utilizados para adicionar elementos SVG (os ícones).

Na abordagem deste trabalho, uma regra é usada para transformar uma sentença visual *SV1* em outra chamada *SV2* através da criação, exclusão ou alteração de símbolos.

As regras são especificadas em uma tabela de regras com três colunas. A primeira, chamada coluna de **pré-condições,** especifica as pré-condições necessárias para a aplicação da regra. A segunda, chamada **coluna de tokens**, mostra as relações entre os símbolos da pré e pós-condições. A última coluna, chamada **pós-condições**, especifica as pós-condições da regra. Esta coluna descreve o estado da sentença visual após a aplicação da regra.

As colunas pré-condição e pós-condição são compostas de nodos e ícones. Os **ícones** representam **símbolos** encontrados na sentença visual ou **recursos externos**, por isso devem ser graficamente similares aos símbolos encontrados nas sentenças visuais ou representar/lembrar o recurso externo que representa.

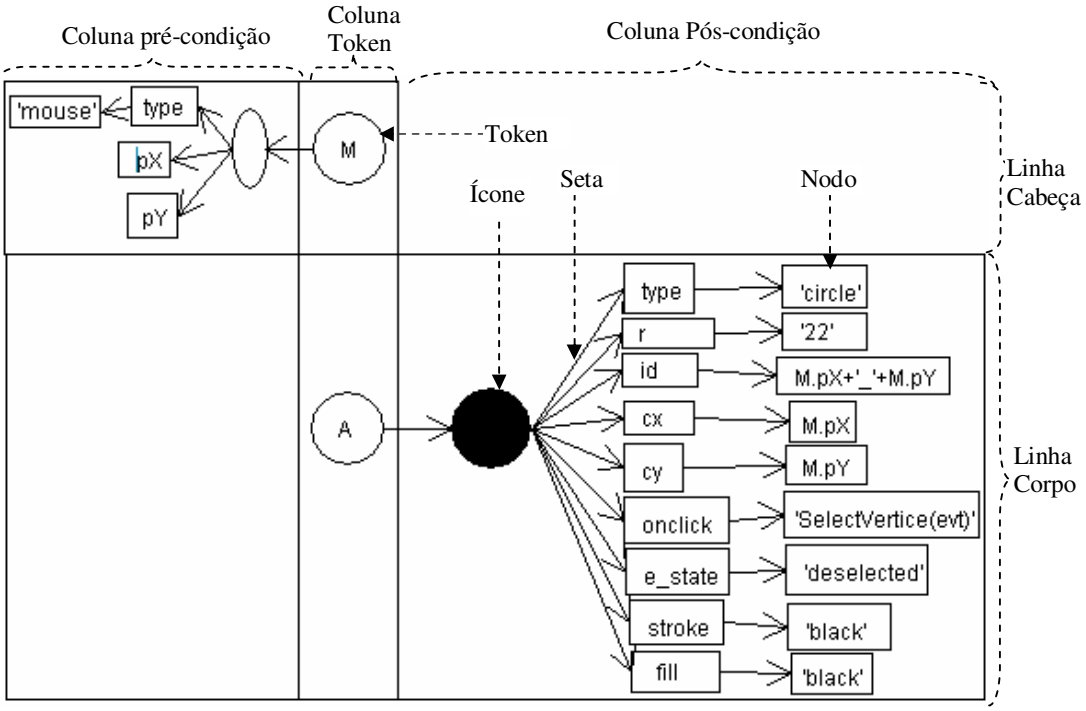
Um recurso externo é uma variável ou objeto que não pertence à sentença visual. Estes recursos fazem parte de um contexto associado a um evento do documento. Apesar de não pertencer à sentença visual, os recursos externos podem ser consultados e usados durante a aplicação de uma regra Vitransf.

A posição atual do mouse e o atual horário do sistema operacional são dois exemplos de recursos externos. Uma regra Vitransf pode especificar, por exemplo, que um círculo deve ser criado na sentença visual, sendo que a coordenada do centro do círculo é a atual coordenada do mouse. Neste caso, o círculo criado é um símbolo, o ato de clicar o mouse é o evento e a coordenada do mouse é um recurso externo que foi consultado. No Vitransf os eventos são os eventos padrão do SVG e os recursos externos são os objetos da *DOM Interface Event* (PIXLEY, 2000) associada a cada um destes eventos.

Os ícones não têm função semântica, portanto são comentários gráficos que precisam ser colocados em pontos específicos da regra para orientar ou esclarecer o significado destas regras. Os **nodos** são estruturas que representam abstratamente atributos e valores de atributos.

A coluna token é composta de estruturas visuais chamadas tokens. **Tokens** são estruturas que representam operações de mapeamento. Adicionalmente, a seta é outro tipo de estrutura que pode ser encontrado em todas as colunas. A **seta** mostra as conexões entre as estruturas da regra e tem uma direção (cada seta começa em uma estrutura visual e termina em outra). A figura 6.8 mostra dois exemplos de regras. A regra *CreateVertice* especifica que um círculo preto deverá ser criado na posição atual do mouse. Os nodos conectados ao círculo definem os atributos do novo círculo e seus valores. A regra *SelectVertice* específica que um círculo preto será transformado em um círculo cinza. Os nodos da pré-condição definem atributos que precisam estar presentes num círculo preto para que ele seja transformado, os atributos que não são mostrados são opcionais. Os nodos da pós-condição especificam quais atributos serão transformados ou criados, os atributos não mostrados permanecem sem alterações.

Regra: CreateVertice



Regra: SelectVertice

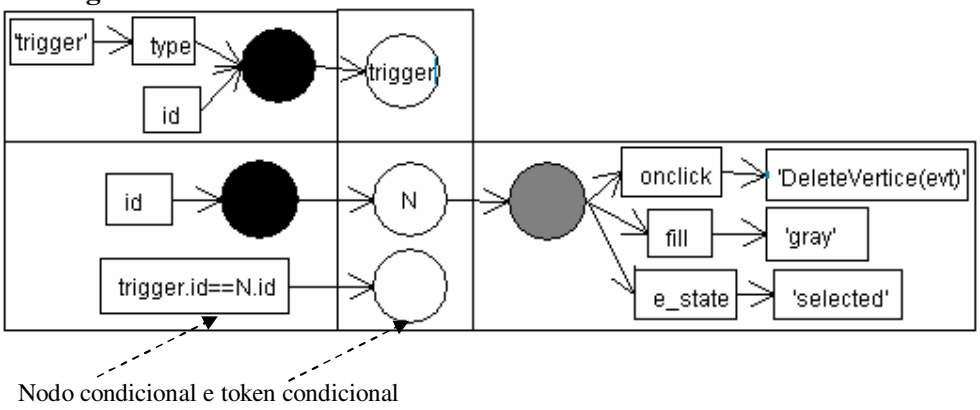


Figura 6.8: Regras Vitransf. *CreateVertice* mostra a criação de um símbolo e *SelectVertice* mostra uma alteração.

A tabela de regras é dividida horizontalmente em duas partes: cabeça e corpo.

A **cabeça** contém apenas a primeira linha. A **coluna token da cabeça** tem um token para cada recurso externo que precisa estar disponível como uma pré-condição para a aplicação da regra. Na **coluna pré-condição da cabeça**, um ícone é conectado a cada token. Cada ícone deve ser um comentário que representa um recurso externo. Vários nodos podem ser conectados a cada ícone. Cada nodo conectado a um ícone representa um atributo que precisa estar presente no recurso externo que está sendo representado. Cada nodo que representa um atributo pode estar conectado a outro nodo que

representará o valor que o referido atributo precisará ter para que a regra possa ser aplicada. A **coluna pós-condição da cabeça** é vazia e sem bordas.

A cabeça define como o ambiente de execução vai gerenciar as interações entre a regra e os recursos externos. A cabeça especifica todos os recursos externos que precisam estar disponíveis para a aplicação da regra (qualquer recurso não especificado é opcional). Após o ambiente de execução verificar que todas pré-condições da cabeça da regra são verdadeiras ele verificará as pré-condições do corpo.

Adicionalmente a máquina Vitransf cria uma **variável associada** para cada atributo *At* de cada recurso externo *R* representado na cabeça. O nome da **variável associada** é da forma: *resource.attr*, onde *resource* é o texto que está dentro do token que representa *R*. Já *attr* é o texto que esta dentro do nodo que representa *At*. O valor de cada variável é o atual valor de seu correspondente atributo. Uma vez atribuídos os valores para as variáveis, estes valores permanecem os mesmos até o fim da aplicação da regra. Durante a aplicação da regra *CreateVertice* da fig. 6.8, por exemplo, será criada a variável associada *M.py,* o seu valor será a coordenada *y* do mouse. Esta variável é referenciada em outras partes da regra, por exemplo, ela será utilizada para definir a coordenada *y* do centro do círculo que será criado. Também serão criadas as variáveis associadas *M.px* e *M.type*.

O **corpo** da regra é composto de uma ou mais linhas onde cada linha especifica uma transformação que pode ser aplicada na sentença visual. Durante a execução da regra, a primeira linha do corpo é avaliada. Se os símbolos da sentença visual têm todas pré-condições especificadas na primeira linha da regra, então, as transformações especificadas nas pós-condições da primeira linha da regra são aplicadas. Quando um símbolo da sentença visual atende a um conjunto de condições diz-se que o símbolo é unificado a estas condições. Este processo é chamado de processo de unificação (*matching process*). Os símbolos da sentença visual com as pós-condições são marcados como "consumidos". A sentença visual é avaliada novamente e se os símbolos não "consumidos" da sentença visual estão unificados a pré-condição da primeira linha as transformações são executadas novamente. Este processo é repetido até que não haja nenhuma unificação. No caso da regra *SelectVertice* da fig. 6.8 a regra irá transformar todos os círculos pretos da sentença visual que atenderem a suas pré-condições. Cada círculo que for transformado é marcado como "consumido". As marcas de consumido são excluídas no final da aplicação da primeira linha da regra.

Após estes passos a próxima linha é avaliada da mesma forma que a primeira linha. Este processo é repetido para todas as linhas do corpo. É importante notar que quando uma linha transforma a sentença visual, a próxima linha irá avaliar a sentença visual já transformada.

Na **coluna pré-condição do corpo**, um ícone é um comentário gráfico que deve representar um símbolo que precisa estar presente na sentença visual para a aplicação da regra. Nodos diretamente conectados a ícones especificam atributos obrigatórios. Nodos conectados aos nodos atributos especificam valores obrigatórios para os atributos aos quais estão conectados.

Quando os recursos da sentença visual são unificados com as pré-condições, a máquina associa o atual valor de cada atributo *At* de cada símbolo unificado *R* a uma variável cujo nome é da forma: *resource.attr*, onde *resource* é o texto que está dentro do token conectado ao ícone que representa *R* e *attr* é o texto que esta dentro do nodo que representa *At*. As **variáveis associadas** podem ser usadas apenas em atribuições e

predicados da mesma linha. As variáveis associadas da cabeça e do corpo são criadas e associadas a valores da mesma maneira.

A coluna pré-condição do corpo da regra também pode ter um tipo especial de nodo, chamado de nodo condicional. Este nodo contém um texto que especifica condições adicionais para a aplicação da regra. Estas especificações são predicados que podem usar valores literais, operadores lógicos (AND, OR, etc.) e variáveis associadas. Este nodo também pode especificar pré-condições hierárquicas, ou seja, ele pode definir que um símbolo precisa ser filho de outro símbolo, por exemplo. O nodo condicional sempre está conectado a um token chamado token condicional. O token condicional é opcional. Se ele está presente ele é o último token da linha e não contém um texto.

A **coluna token do corpo** define como as pré-condições estão conectadas com as pós-condições. As setas que chegam ou saem do token especificam operações de mapeamento de acordo com as seguintes definições:

- Uma **exclusão** é especificada se uma seta chega ao token, mas nenhuma sai do token. O símbolo da sentença visual unificado com a pré-condição conectada ao token é excluído na transformação. Os filhos deste símbolo também são excluídos neste processo.
- Uma **criação** é especificada quando nenhuma seta chega ao token e uma sai do token. Um símbolo é criado na sentença visual durante a transformação. As características deste símbolo são especificadas nas pós-condições conectadas ao token.
- Uma **alteração** é especificada quando uma seta chega ao token e outra sai. O símbolo da sentença visual que está unificado com as pré-condições conectadas ao token será transformado de acordo com as pós-condições conectadas ao mesmo token.

A **coluna de pós-condições do corpo** pode ter as seguintes configurações. Para **exclusões** não há nenhuma pós-condição. Para **alterações**, um ícone conectado ao token representa o estado do símbolo após a transformação. Nodos conectados a este ícone especificam os atributos do símbolo que serão alterados. Nodos conectados aos nodos atributos especificam os novos valores dos referidos atributos. Estas especificações são atribuições que podem usar valores literais, operadores aritméticos e variáveis associadas.

Em **criações**, o token estará conectado a um ícone que representa o símbolo que será criado na sentença visual. Um nodo com um valor especial (*type*) estará conectado a este ícone. Este nodo especial chama-se **nodo tipo**. Um nodo conectado ao nodo tipo definirá o tipo de símbolo que será criado e o pai deste símbolo (informando assim a posição hierárquica do símbolo que será criado). Quando o pai não é informado o nodo é adicionado como filho do elemento raiz. Outros nodos conectados ao referido ícone especificarão os atributos que serão adicionados ao símbolo criado. Nodos conectados aos nodos atributo especificarão os valores dos novos atributos. Estas especificações são atribuições com a mesma sintaxe usada no caso de alterações.

É importante notar que o nodo tipo também pode ser usado em alterações e exclusões. O nodo tipo é usado da mesma forma que os nodos atributo. Ele pode fazer parte de qualquer pré ou pós-condição e quando não é citado significa que o tipo do símbolo é opcional ou não é alterado durante a transformação.

6.2.6.2 Detalhes sintáticos da visão semântica das regras Vitransf

Os textos que podem ser colocados nos nodos das pré e pós condições têm as seguintes sintaxes:

- Inicialmente, somente nomes de elementos SVG podem ser colocados dentro de nodos tipo e somente nomes de atributos SVG podem ser colocados dentro de nodos atributos. Porém esta limitação foi flexibilizada pela implementação de um suporte a múltiplos espaços de nomes que permite a adição de entidades de outras linguagens e até mesmo de entidades que são apenas bem formadas. O desenvolvedor pode especificar elementos e atributos não definidas no vocabulário SVG e adicionar estas entidades nas regras Vitransf usando o formato *prefix_ent*, onde *prefix* é um prefixo para o espaço de nomes alternativo e *ent* é o nome da entidade neste espaço de nomes (e.g. atributo *e_state* na fig. 6.8). O processo de exportação transforma este formato em um formato XML que está de acordo com o padrão namespace da W3C e que pode ser manipulado por navegadores web ou scripts.
- A sintaxe dos predicados dos nodos condicionais é a mesma da clausula "if" da linguagem JScript.
- A sintaxe das atribuições dos nodos das pós-condições é a mesma das atribuições JScript (após o sinal ´=´).
- Os predicados e atribuições podem usar variáveis associadas (as variáveis do tipo *resource.atrr* mostradas nesta seção) da mesma forma que as demais variáveis utilizadas na linguagem JScript.

Os recursos externos da atual implementação são objetos Jscript que são criados pela máquina Jscript quando o documento SVG é carregado ou quando ocorre um evento. Para que estes recursos possam ser utilizados nas regras Vitransf é preciso criar, no editor Vitransf, rotinas que traduzam as referências a recursos externos em comandos Jscript. Assim, até o momento foram definidos e implementados três tipos de objetos jscript que podem ser citados em regras Vitransf.

- Os dois primeiros são os objetos jscript *evt.clientX* e *evt.clientY*. Nas regras Vitransf estes objetos devem ser referenciados como os atributos *pX* e *pY* do recurso externo *mouse*, estes atributos contém respectivamente os valores das coordenadas x e y do mouse.
- O terceiro objeto jscript que pode ser referenciado nas regras Vitransf desta implementação é o *evt.trigger*. Este objeto referencia o elemento do documento SVG onde foi aplicado algum evento, por exemplo, o elemento que foi clicado. Nas regras Vitransf desta implementação este objeto deve ser referenciado como *trigger*. Os atributos do elemento SVG apontado pelo objeto *evt.trigger* também podem ser citados nas regras Vitransf como atributos do recurso externo *trigger*.

Todos os dados informados nesta visão são armazenados em um documento XML que esta de acordo com o esquema de regras Vitransf.

6.2.7 Síntese das operações e sistemas do Vitransf

No Vitransf um documento pode ter várias visões. Cada visão mostra diferentes aspectos ou partes do documento e todas as visões estão sincronizadas. Além disto, devido ao recurso de espaço de nomes, em um mesmo documento podem ser aplicadas operações que pertencem a diferentes modos de edição. Portanto o Vitransf é um ambiente que integra, gerencia e sincroniza todas estas visões e operações. Desta

maneira pode-se utilizar várias visões alternadamente e simultaneamente. A figura 6.9 mostra 3 visões sincronizadas que estão sendo utilizadas simultaneamente para editar o mesmo documento, uma sentença visual inicial.

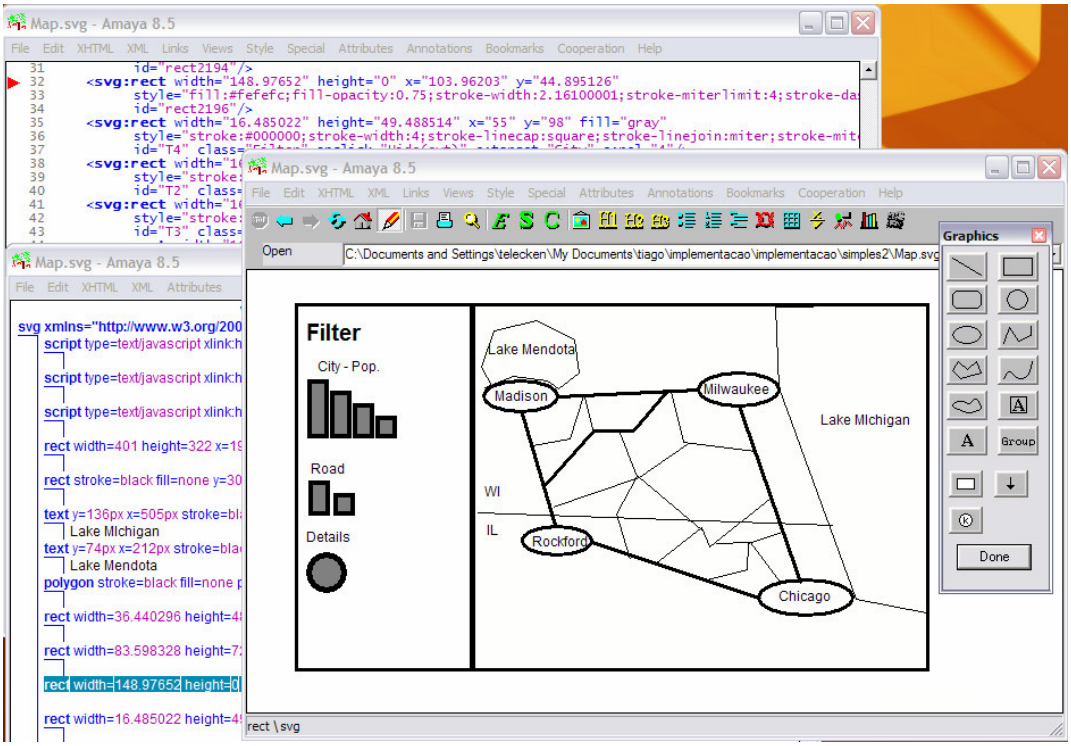


Figura 6.9: Mútiplas visões sincronizadas do Vitransf

A tabela 6.2 mostra os editores e modos de edição típicos de ADBGs nas duas primeiras colunas. A terceira coluna mostra o modo de edição XML que contém as operações que realizam as respectivas visões típicas de ADBGs. A quarta coluna mostra o editor utilizado para realizar as funcionalidades de suas respectivas visões. As visões não implementadas estão marcadas com asteriscos "******".

Dos 10 tipos de visões possíveis de se implementar em ADBGs, 6 foram implementadas no Vitransf. Dentre as 6 visões implementadas 5 visões reutilizaram ferramentas XML padrão, nenhuma implementação foi feita nestas ferramentas para que elas fossem reaproveitadas no Vitransf. Somente a visão semântica das regras Vitransf precisou ser implementada a parte, mesmo assim muitos recursos básicos do XML foram reaproveitados.

Dentre as 6 visões implementadas, 4 são compatíveis com editores/visualizadores externos. Pode-se editar/visualizar documentos em ferramentas externas e depois reutilizar estes documentos como se os mesmos tivessem sido editados no Vitransf. Quando estas ferramentas externas são utilizadas têm-se as mesmas vantagens encontradas nos editores Vitransf. Ou seja um editor XML externo controlado por esquema implementa todas as funcionalidades de uma visão sintática controlada por alfabeto. Um editor XML padrão implementa todas as funcionalidades de uma visão controlada por modelo de sentença visual, e assim por diante.

Tabela 6.2: Entidades de ADBGs e componentes XML que realizam estas entidades no Vitransf

Editores ADBGs	**Modos de edição de ADBGs**		**Modos do XML**	**Editores XML utilizados no Vitransf**
Editor de sentença visual	Entidade básica		Código fonte	Editor de código fonte do Amaya
	Controlada por modelo de sentença visual		Estrutural livre	Editor XML do Amaya
Editor de alfabeto	Controlada por alfabeto	Visão Sintática	Controlada por esquema	Editor XML controlado por esquema do Amaya
		Visão Semântica	Controlada por semântica	Extensão do editor XML controlado por esquema do Amaya (editor SVG semântico)
	Controlada por meta-modelo de alfabeto	Visão Sintática	*******	*******
		Visão Semântica	*******	*******
Editor de gramática	Controlada por gramática	Visão Sintática	*******	*******
		Visão Semântica	Controlada por semântica	Navegador Web
	Controlada por meta-modelo de gramática	Visão Sintática	*******	*******
		Visão Semântica	Controlada por semântica	Extensão do editor XML controlado por esquema

6.3 Estudo de casos

O Vitransf pode ser utilizado para se especificar simulações, animações e outros sistemas visuais interativos que rodam em navegadores Web. Nesta seção, serão apresentados dois estudos de caso onde sistemas visuais interativos são especificados através de uma gramática Vitransf e executados em navegadores Web.

6.3.1 Estudo de caso 1: Editor de grafos

O primeiro estudo de caso mostra como especificar a aparência e o comportamento de um editor de grafos através de uma gramática Vitransf.

O referido editor de grafos tem as seguintes características: é composto de um grande retângulo branco (a área de trabalho) no qual o usuário pode clicar com o mouse. No ponto clicado, o sistema cria um vértice (um círculo preto). Se o usuário clicar em um vértice, o vértice é selecionado e muda a sua cor (de preto para cinza). Na parte

inferior da área de trabalho há um botão chamado "*Create edge*". Se o usuário clicar este botão o sistema cria um arco entre dois nodos previamente selecionados (ver figuras 6.11b e 6.11c). Se o usuário clicar em um nodo selecionado, este nodo e todos os arcos ligados ao nodo serão excluídos (ver figuras 6.12b e 6.12c). As figuras 6.10, 6.11 e 6.12 mostram algumas telas do editor.

Para especificar este editor através de regras Vitransf são necessárias 4 regras e uma sentença visual inicial. A sentença visual inicial usada neste estudo de caso contém um retângulo branco que quando clicado chama a regra *CreateVertice* e um botão chamado "*Create edge"* (um retângulo com texto*)* que quando clicado o chama a regra *CreateEdge*. A figura 6.10 mostra a referida sentença visual inicial.

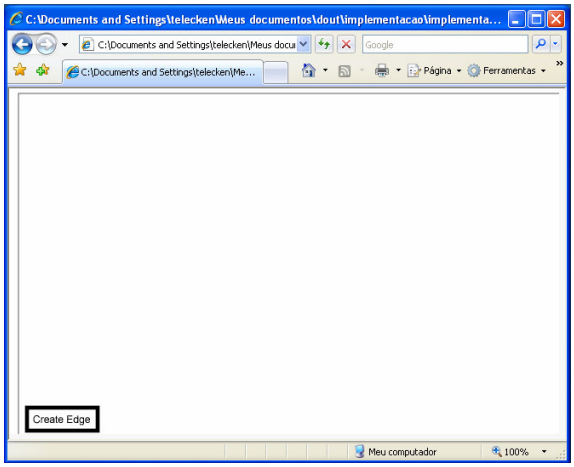


Figura 6.10: Sentença visual inicial do editor de grafos

As 2 primeiras regras (*CreateVertice* e *SelectVertice*) foram mostradas na figura 6.8 (ver seção 6.2.6.1). Estas regras especificam como um vértice é criado e selecionado. As últimas regras são apresentadas nas figuras 6.11 e 6.12. Para uma maior brevidade serão detalhadas apenas as duas últimas regras. Na fig. 6.11, a regra *CreateEdge* especifica a criação de um arco. A cabeça da regra é vazia por isso nenhum recurso externo é requisito para a execução da regra. A coluna pré-condição do corpo da regra especifica que dois vértices com um conjunto pré-definido de atributos precisam ser encontrados na sentença visual. O conjunto de atributos especifica um vértice selecionado. As pós-condições especificam que após as transformações os atributos *onclick, e_state* e *fill* do vértice encontrado terão os valores '*SelectVertice (evt)*', '*black*' e '*deselected*' respectivamente. Um arco será criado. O arco é uma linha (elemento *line* do SVG), a coordenada de seu ponto inicial (x1, y1) será a coordenada do ponto central (cx, cy) do primeiro vértice encontrado (N1). O ponto final da linha será a coordenada do ponto central do segundo vértice encontrado (N2). Dois atributos serão criados: *e_from* e *e_to*. Seus valores serão os *ids* do N1 e N2, respectivamente.

Nesta regra os atributos *e_state, e_from* e *e_to* não pertencem ao alfabeto Vitransf (não são atributos SVG). Estes atributos foram adicionados a regra como entidades bem formadas (utilizando o suporte a múltiplos espaços de nomes).

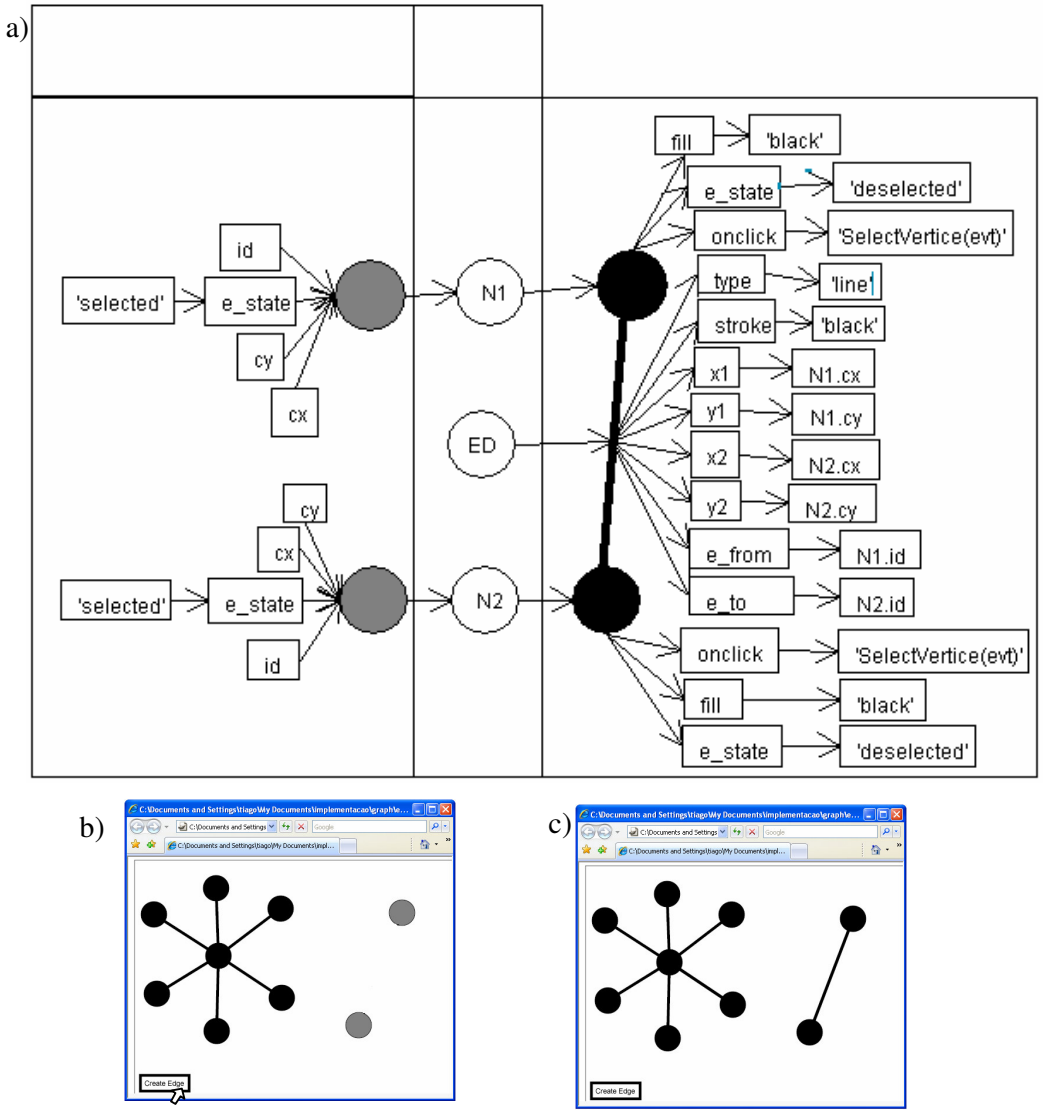


Figura 6.11: A parte (a) Mostra a regra *CreateEdge*. As partes (b) e (c) mostram a regra *CreateEdge* sendo aplicada em uma sentença visual em tempo de execução. Em (b) o usuário está clicando o botão '*Create Edge*'. A parte (c) mostra o arco criado.

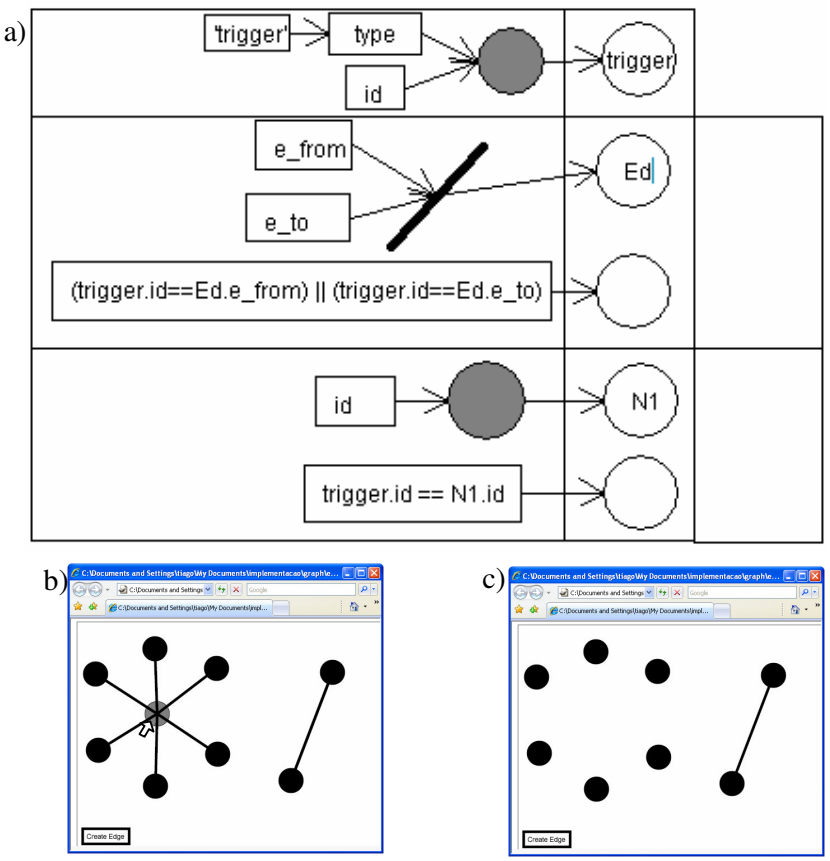


Figura 6.12: A parte (a) mostra a regra *DeleteVertice*. As partes (b) e (c) mostram a regra *DeleteVertice* sendo aplicada em uma sentença visual em tempo de execução. Em (b) o usuário está clicando um nodo selecionado. A parte (c) mostra a tela após a execução da regra *DeleteVertice*.

Na fig. 6.12 a regra **DeleteVertice** especifica como um vértice é excluído. A cabeça da regra especifica que o elemento que chamou a regra precisa ter o atributo *id*. A primeira linha do corpo da regra especifica que um elemento com os atributos *e_from* e *e_to* precisa ser encontrado. De acordo com o nodo condicional o atributo *e_to* ou o *e_from* precisa ser igual ao *id* do elemento que chamou a regra (neste caso, o vértice que o usuário clicou). Todos os arcos com estas características serão excluídos (ver fig. 3.7b e 3.7c). A segunda linha determina que um vértice com o *id* igual ao *id* do elemento que chamou a regra será excluído (neste caso, o vértice clicado pelo usuário).

6.3.2 Estudo de caso 2: Ferramenta filtro

O segundo estudo de caso mostra como especificar uma ferramenta filtro através da gramática Vitransf. Nas figuras 6.13b e 6.13c uma aplicação cartográfica utiliza a ferramenta de filtro. A aplicação cartográfica tem um mapa de um lado e alguns filtros de outro. O mapa tem diferentes classes de elementos tais como cidades, rodovias e detalhes. Cada filtro está associado a uma classe de elementos do mapa. Quando um filtro é clicado os elementos associados ao filtro são mostrados ou escondidos. Na figura 6.13b o usuário clica nos filtros associados às estradas secundárias e aos detalhes do mapa. A figura 6.13c mostra que os detalhes e estradas secundárias foram escondidos. Se o usuário clicar nos mesmos filtros novamente os correspondentes elementos serão mostrados.

Os filtros têm um atributo chamado *target* que especifica a classe de elementos que o filtro pode esconder ou mostrar. Os elementos no mapa têm um atributo chamado *class* que especifica a sua classe. O comportamento da aplicação é especificado em duas regras: *Hide* e *Show*. A regra *Hide* é mostrada na figura 6.13a. A cabeça da regra especifica que o elemento que chama a regra precisa ter o atributo *e_target*. A primeira linha do corpo especifica que um elemento da classe *filter* que tem o mesmo *target* que o elemento que chamou a regra irá mudar seus atributos *onclick* e *fill*. A segunda linha especifica que os elementos que tem o mesmo atributo *class* que o elemento que chamou a regra terão os atributos *stroke* alterados. A regra *Show* é similar a regra *Hide*, porém ela especifica as transformações inversas.

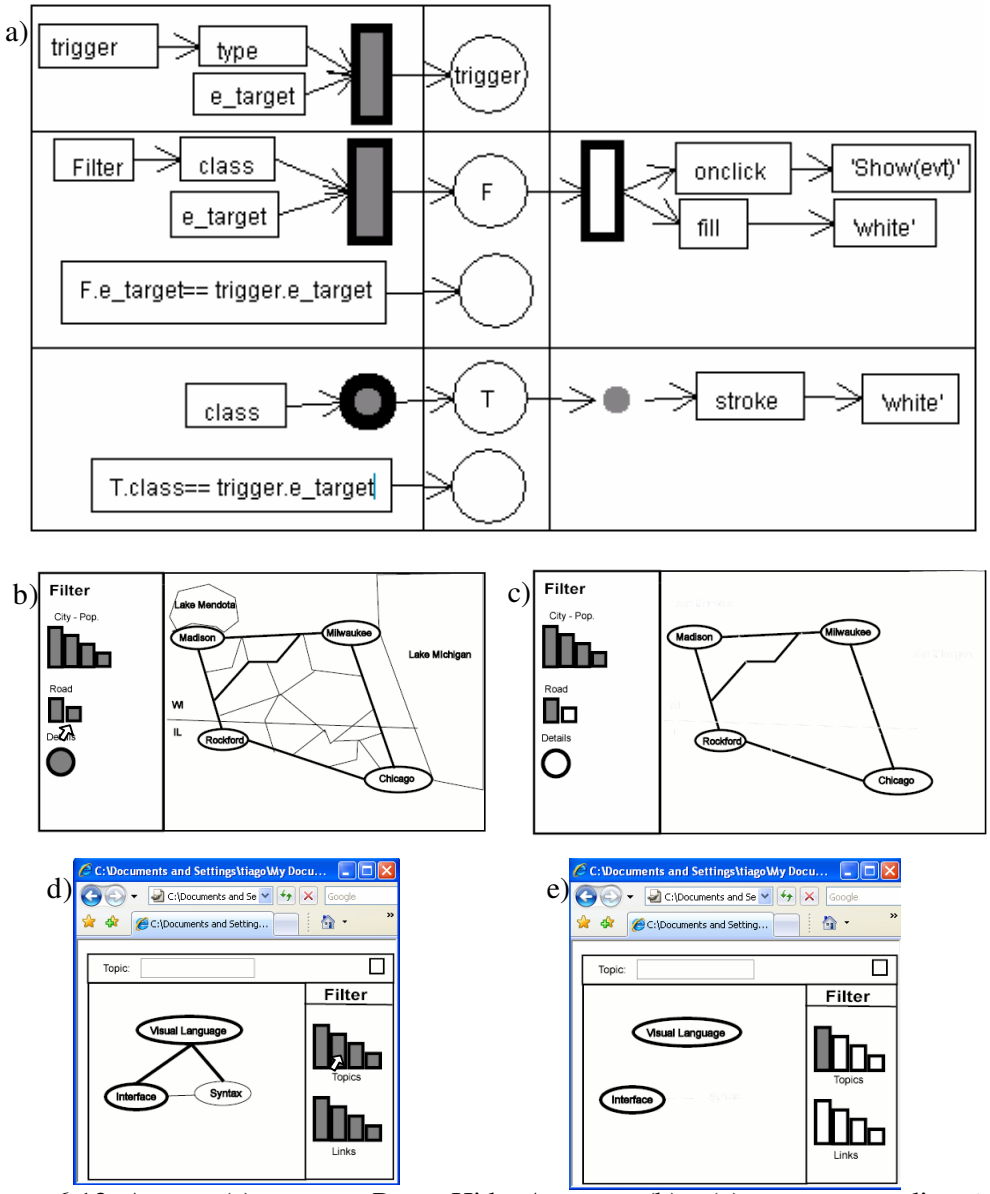


Figura 6.13: A parte (a) mostra a Regra Hide. As partes (b) e (c) mostram a aplicação cartográfica. As partes (d) e (e) mostram a aplicação mapa conceitual

A ferramenta filtro também mostra a flexibilidade das regras; o filtro pode ser um retângulo um circulo ou qualquer outro elemento SVG. O filtro apenas precisa ter um

atributo *e_target* e outro *class*. Os elementos no mapa também podem ser diferentes tipos de elementos SVG. De fato, as mesmas duas regras podem ser usadas por muitas aplicações. As figuras 6.13d e 6.13e mostram uma aplicação de mapas conceituais onde o usuário pode filtrar conceitos e relacionamentos. A aplicação de mapas conceituais usa exatamente as mesmas regras da aplicação cartográfica. O que diferencia as duas aplicações é a sentença visual inicial.

6.4 Análise

Como foi demonstrado o Vitransf usou e reusou extensivamente componentes do espaço tecnológico XML. Todas as estruturas do sistema (modelos, meta-modelos, sentenças visuais, alfabetos e gramáticas) são realizadas por recursos do espaço tecnológico XML (especificação XML, documentos XML, esquemas XML e meta-modelos de esquemas XML).

Todas estas estruturas são manipuladas por componentes XML que realizam operações típicas de ADBGs. Tais operações são agrupadas em ferramentas XML já padronizadas. O ambiente de execução é um navegador web padrão com suporte ao SVG e a ECMAScripts. Já o Ambiente de desenvolvimento é formado por um conjunto de editores já padronizados pelo espaço tecnológico XML.

Analisando mais detalhadamente o ambiente de desenvolvimento, pode-se constatar que dos 10 tipos de visões possíveis de se implementar em um ADBG, 6 foram implementadas no Vitransf. Dentre as 6 visões implementadas 5 visões reutilizaram ferramentas XML padronizadas. Nenhuma implementação foi feita nestas ferramentas para que elas fossem reaproveitadas no Vitransf. Somente a visão semântica das regras Vitransf precisou ser implementada a parte, mesmo assim muitos recursos básicos do XML foram reaproveitados. Além disto todas as operações implementadas nesta visão foram operações XML que podem ser classificadas como operações do modo de edição controlada por semântica.

Adicionalmente, dentre as 6 visões implementadas, 4 são compatíveis com editores/visualizadores externos já padronizados pelo espaço tecnológico XML. Existem portanto muitas alternativas para a edição/análise de gramáticas. Desenvolvedores podem utilizar editores XML externos com os quais estão mais acostumados, que tem mais recursos ou que tem diferentes interfaces. Depois estes documentos podem ser interpretados pelo Vitransf como se tivessem sido editados no Vitransf.

Assim, do ponto de vista do desenvolvedor de sistemas visuais interativos, a compatibilidade com o espaço tecnológico XML permite a utilização e integração de ferramentas e recursos externos. Estas ferramentas podem ser integradas aos processos de desenvolvimento e execução de sistemas visuais interativos de acordo com as necessidades do desenvolvedor e do sistema que está sendo editado.

Já do ponto de vista dos implementadores do ADBG, o reuso dos recursos XML facilitou a implementação dos ambientes de execução e de desenvolvimento do Vitransf pois grande parte dos ambientes reutilizou recursos XML já prontos, amplamente testados e difundidos. Foram reutilizados recursos XML para definição, edição, visualização e armazenamento de dados semi-estruturados. Restou para ser codificado em C (sem a utilização de nenhuma linguagem de alto nível do XML) apenas a implementação da lógica da gramática que está concentrada em dois conjuntos de

rotinas. Um destes conjuntos de rotinas interpreta documentos XML (a gramática Vitransf) e a partir deles gera um sistema visual interativo. O outro conjunto de rotinas implementadas especifica a interface gráfica da visão semântica das regras Vitransf.

7 TRABALHOS RELACIONADOS - A APLICAÇÃO DE COMPONENTES XML EM ADBG

Muitos ambientes de desenvolvimento baseados em gramáticas utilizam componentes XML. A maneira como estes componentes XML são utilizados é descrita nas próximas seções.

7.1 Estruturas de dados

Em ambientes de desenvolvimento que utilizam componentes XML as sentenças visuais, alfabetos e gramáticas são realizadas por documentos XML. Já a definição das estruturas destes documentos é realizada por esquemas XML. Portanto o modelo de sentença visual, o meta-modelo de alfabeto e o meta-modelo de gramática são realizados por esquemas XML. A tabela 7.1 mostra as estruturas de ADBGs na primeira coluna e os componentes XML que realizam estas estruturas na segunda coluna.

Tabela 7.1: Estruturas de dados de ADBGs e componentes XML que os realizam

Estruturas de ambientes de desenvolvimento	**Componentes XML utilizados nos atuais ADBGs**
Modelo de sentença visual	Esquema XML
Sentença visual	Documento XML
Meta-modelo alfabeto	Esquema XML
Alfabeto	Documento XML
Meta-modelo de gramática	Esquema XML
Gramática	Documento XML

Esta é a maneira como as atuais abordagens utilizam componentes XML para realizar seus dados e estruturas de dados. Eventualmente, alguns ambientes de desenvolvimento realizam a gramática e o alfabeto em um mesmo documento XML e a estrutura deste documento é definida em um único esquema XML. Em outros casos vários esquemas são utilizados para especificar diferentes partes da gramática que serão realizadas em diferentes documentos. Entretanto o uso de documentos XML para

realizar dados e o uso de esquemas XML para realizar a definição da estrutura dos dados é mantida em todos os casos.

Outra variação do uso de componentes XML que realizam dados é que nem todos ambientes utilizam todos componentes aqui citados. Alguns sistemas realizam em documentos XML apenas as sentenças visuais, outros realizam apenas as gramáticas e assim por diante. Entretanto quando os componentes XML são utilizados, os dados são sempre realizados em documentos XML e a definição das estruturas destes dados é realizada em esquemas XML.

Os componentes XML que realizam as classes do modelo de linguagem visual dos atuais ADBGs são mostrados no correspondente estereótipo de cada classe da figura 7.1. Em adição as estruturas já citadas na tabela 7.1, a figura 7.1 mostra que em tempo de execução uma linguagem visual é realizada por um documento XML e um script, onde script é qualquer componente capaz de alterar um documento XML.

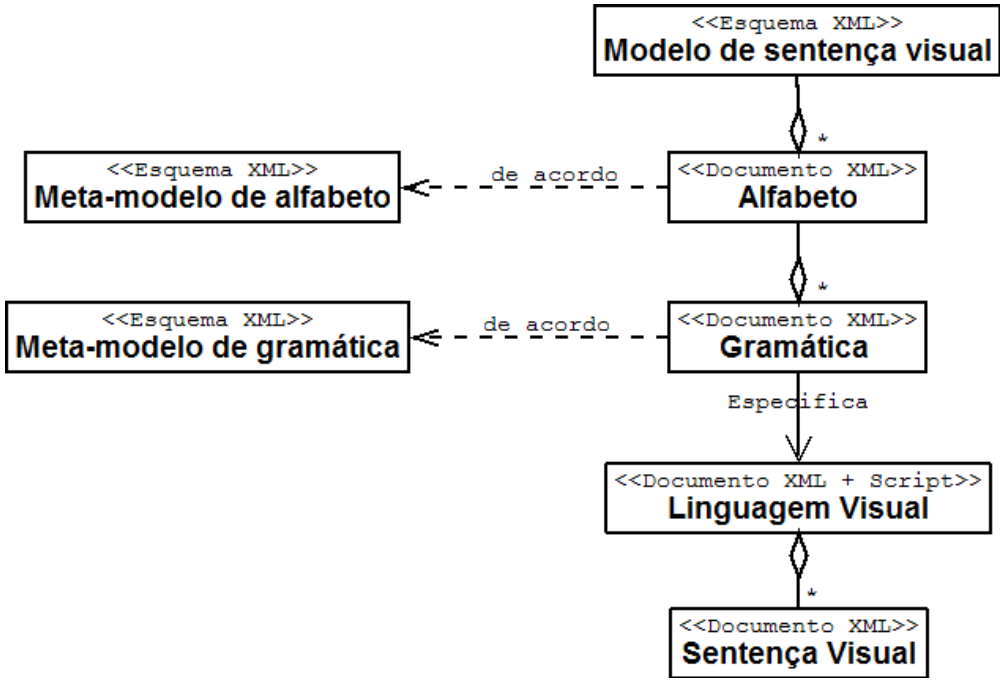


Figura 7.1: Entidades e correspondentes componentes XML das atuais abordagens

7.2 Operações

As operações típicas do modo de edição de código fonte XML destacam e manipulam caracteres. Já as operações típicas dos modos de edição estrutural livre e controlado por esquema do XML destacam e manipulam entidades do modelo XML. Estas operações não destacam ou manipulam as entidades de sentenças visuais, alfabetos e gramáticas. Elas tratam estas estruturas de dados em um nível de abstração mais baixo. Ou seja tratam estas estruturas como um conjunto de entidades básicas que neste caso são caracteres e entidades do modelo XML. Portanto as operações do modo de edição de entidades básicas dos atuais ADBGs são realizadas por operações do modo de edição de código fonte, estrutural livre e/ou controlada por esquema do XML.

Para editar sentenças visuais, alfabetos e gramáticas destacando estruturas do modelo de sentença visual, do meta-modelo de alfabeto e do meta-modelo de gramática

são utilizadas um conjunto de operações relacionadas ao domínio dos dados. Há a necessidade de operações dependentes do domínio pois as operações XML independentes de domínio não são suficientes para tratar sentenças visuais como sentenças visuais, alfabetos como alfabetos etc.

No XML as operações dependentes de domínio caracterizam o modo de edição controlada por semântica. Portanto, nos atuais ADBGs as operações dos modos de edição controladas por modelo de sentença visual, alfabeto, gramática, meta-modelo de alfabeto e meta-modelo de gramática são realizadas por operações do modo de edição controlada por semântica do XML.

Na primeira coluna da tabela 7.2 são mostrados todos os modos de edição típicos de ADBGs. Nos atuais ADBGs que utilizam componentes XML as operações destes modos de edição são realizadas por operações do modo de edição XML mostrado na segunda coluna da tabela 7.2.

Tabela 7.2: Modos de edição de ADBG e correspondentes componentes XML que os realizam

Modo de edição de ADBG		**Correspondente modo de edição XML utilizado em ADBGs**
Entidade básica		Código fonte, estrutural livre e controlada por esquema
Controlada por modelo de sentença visual		Controlada por semântica
Controlada por alfabeto	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica
Controlada por meta-modelo de alfabeto	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica
Controlada por gramática	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica
Controlada por meta-modelo de gramática	Visão Sintática	Controlada por semântica
	Visão Semântica	Controlada por semântica

Nem todos ADBGs que utilizam componentes XML realizam todos seus modos e visões através de componentes XML. O uso de componentes XML que realizam operações pode ocorrer em alguns modos de edição e não em outros.

Por exemplo, o modo de edição controlada por alfabeto de um ADBG pode ser realizado por operações XML do modo de edição controlada por semântica. Porém as operações do modo de edição controlado por gramática são realizadas através de componentes que não pertencem ao espaço tecnológico XML. Em outras palavras o uso de componentes XML pode ser apenas parcial e pode ser combinado com componentes de outros espaços tecnológicos.

Todo ambiente de desenvolvimento que utiliza componentes XML que realizam operações precisa utilizar os correspondentes componentes XML que realizam estruturas de dados (sentenças visuais, alfabetos e/ou gramáticas armazenadas em

documentos XML). Porém nem todo ambiente que utiliza uma estrutura XML utiliza suas correspondentes operações XML.

7.3 Sistemas

Como foi visto em seções anteriores, associado a cada modo de edição XML existe um tipo de editor XML que pode implementar as operações do referido modo de edição XML. As operações do modo de edição controlado por semântica podem ser implementadas em extensões de editores XML controlados por esquema. Já as operações dos modos de edição de código fonte, estrutural livre e controlado por esquema podem ser respectivamente implementados por editores de código fonte, editores XML e editores XML controlados por esquema.

A tabela 7.3 mostra nas duas primeiras colunas os editores ADBG e seus modos de edição. A terceira coluna mostra os modos de edição XML que realizam os correspondentes modos de edição de ADBG. Já a quarta coluna mostra os editores XML que podem realizar as correspondentes operações dos correspondentes editores ADBG.

Tabela 7.3: Editores ADBG e correspondentes componentes XML

Editores ADBG	**Modos de edição ADBG**		**Modos de edição XML**	**Editores XML**
Editor de sentença visual	Entidade básica		Código fonte, estrutural livre, controlado por esquema	Editor de código fonte, editor XML, editor XML controlado por esquema
	Controlada por modelo de sentença visual		Controlada por semântica	Ext. de editor XML controlado por esquema
Editor de alfabeto	Controlada por alfabeto	Visão Sintática	Controlada por semântica	Ext. de editor XML controlado por esquema
		Visão Semântica	Controlada por semântica	Ext. de editor XML controlado por esquema
	Controlada por meta-modelo de alfabeto	Visão Sintática	Controlada por semântica	Ext. de editor XML controlado por esquema
		Visão Semântica	Controlada por semântica	Ext. de editor XML controlado por esquema
Editor de gramática	Controlada por gramática	Visão Sintática	Controlada por semântica	Ext. de editor XML controlado por esquema
		Visão Semântica	Controlada por semântica	Ext. de editor XML controlado por esquema ou o ambiente de execução
	Controlada por meta-modelo de gramática	Visão Sintática	Controlada por semântica	Ext. de editor XML controlado por esquema
		Visão Semântica	Controlada por semântica	Ext. de editor XML controlado por esquema

ADBGs que utilizam editores XML típicos necessariamente implementam operações XML e utilizam estruturas de dados XML. Porém nem todo ADBG que utiliza estruturas e operações XML utilizam editores XML típicos.

7.4 Síntese

Os ADBGs atuais utilizam os componentes XML de uma maneira tradicional. Ou seja realizam dados em documentos XML e realizam a definição das estruturas de dados em esquemas XML.

Como novas estruturas de dados foram criadas e estas estruturas descrevem informações de áreas de domínio específicas (sentenças visuais, alfabetos etc) novas operações devem ser criadas para que as entidades XML sejam tratadas, editadas e visualizadas como sentenças visuais, alfabetos etc. Estas operações são dependentes de domínio logo pertencem ao modo de edição controlado por semântica do XML e podem ser realizadas em extensões de editores XML controlados por esquemas.

Somente as operações do modo de edição de entidades básicas é que podem ser realizadas por componentes XML que independam de uma área de domínio específica. As operações do modo de edição de entidades básicas podem ser realizadas por operações XML dos modos de edição de código fonte, estrutural livre ou controladas por esquema. Tais operações estão encapsuladas em típicos editores de código fonte, editores XML ou editores XML controlados por esquema.

7.5 ADBGs que utilizam componentes XML

No **Diagen** (BRIELER, 2008; MINAS, 2003) um desenvolvedor especifica a gramática com a ajuda do ADBG que provê os editores de entidades básicas, de sentença visual, de alfabeto e de gramática integrados em um único editor, o Visual Diagen. Após o desenvolvedor especificar a gramática o Diagen gera o editor visual projetado.

Todas as informações das gramáticas, alfabetos e sentenças visuais são armazenadas em documentos XML. As estruturas destes documentos estão descritas em esquemas XML.

O Visual Diagen não foi desenvolvido a partir do zero, ele é uma extensão de um editor XML controlado por esquema (MINAS, 2003). Este editor XML foi estendido de forma a conter operações dos 6 modos de edição típicos de ADBGs. O Diagen é um ADBG completo e utiliza todos os componentes XML citados na seção 7 (estrutura de dados, modos de edição e editores XML) da forma como eles foram descritos ao longo da seção 7.

Diaplan (DREWES, 2004) é uma linguagem baseada em regras que está sendo integrada ao Diagen. Esta integração utiliza a mesma estrutura do Diagen (portanto os mesmos componentes XML), porém a linguagem da gramática é diferente. O Diaplan utiliza *Shapely nested graph transformation* ao invés das *hipergraph transformations* utilizadas no Diagen. Basicamente são gramáticas diferentes com objetivos similares e implementadas sobre uma mesma estrutura.

O **Pounamu** (ZHU, 2007) também é uma ferramenta que gera editores visuais. Porém o Pounamu não é um ADBG completo, ele possui as operações de um editor de sentenças visuais e as operações de um editor de alfabeto. Porém no lugar de um editor de gramáticas o Pounamu provê um editor de meta-modelos. Portanto, para o desenvolvedor especificar um editor visual ele precisa especificar um alfabeto e um meta-modelo no lugar de um alfabeto e uma gramática.

As sentenças visuais, alfabetos e meta-modelos são armazenados em documentos XML e as estruturas destes documentos estão especificadas em esquemas XML. Adicionalmente os editores Pounamu editam documentos XML que descrevem sentenças visuais, alfabetos e meta-modelos através de operações do modo de edição controlado por semântica do XML.

O ADBG **Genial** (BOTTONI, 2004) gera sistemas visuais interativos a partir de gramáticas vcarw. As gramáticas e alfabetos são armazenados em documentos XML cuja estrutura é definida em um esquema XML. Gramáticas e alfabetos são editados em seus respectivos editores, tais editores possuem operações típicas do modo de edição controlado por semântica do XML.

O **VLCC** (*visual language compiler-compiler*) e seu descendente **VLDesk** (Costagliola, 2005) são ADBGs que geram sistemas visuais interativos a partir de gramáticas *positional grammars* e *extended positional grammars*. Estes ambientes de desenvolvimento são ADBGs completos, porém o uso de componentes XML é parcial e combinado com componentes de outros espaços tecnológicos. Sentenças visuais podem ser armazenadas em documentos XML, mais precisamente no formato GXL. Alfabetos e gramáticas são armazenados em um mesmo documento XML. A estrutura destes dados é especificada em esquemas XML.

No início do processo de edição pode-se importar sentenças visuais, alfabetos e gramáticas que estão no formato XML. No fim do processo de edição pode-se exportar estes documentos de volta para o formato XML. Entretanto as sentenças visuais, alfabetos e gramáticas são editados em formatos fora do espaço tecnológico XML. Portanto não são utilizados editores XML padrão. Adicionalmente somente as operações XML de verificação sintática é que são utilizadas (tanto no processo de exportação quanto no processo de importação). Obviamente quando os ambientes VLCC e VLDesk não estão disponíveis ou não são desejados pode-se utilizar editores XML para editar sentenças visuais, alfabetos e gramáticas no modo de edição de entidade básica.

Os ADBGs **AGG**, **GENGED** (ERMEL, 2004)**,** (seu descendente **Tiger** (ERMEL, 2007)) e **PROGRES** (ASCHENBRENNER, 2003) também podem exportar e importar sentenças visuais e/ou alfabetos para o formatos XML. Portanto estes ADBGs utilizam os componentes XML de maneira similar ao VLCC. Outros ADBGs como o **AToM3** (GUERRA, 2007) apenas exportam sentenças visuais, alfabetos e gramáticas para formatos XML. Nestes casos não é possível utilizar editores XML para editar sentenças, alfabetos e gramáticas no modo de edição de entidade básica.

8 COMPARAÇÃO E ANÁLISE

Neste capítulo serão avaliadas as diferenças entre o uso de componentes XML definido no framework SV-XML e o uso de componentes XML nos demais ADBGs. Ao final desta seção serão discutidas as vantagens e desvantagem da abordagem do SV-XML.

Os ADBGs que estão de acordo com o framework SV-XML e os demais ADBGs que utilizam componentes XML possuem em comum o fato de serem típicos ambientes de desenvolvimento baseados em gramática. Entretanto em diversos pontos o uso de componentes XML nos ADBGs baseados no SV-XML difere-se do uso dos componentes XML nas demais abordagens. Tais diferenças serão apontadas e analisadas nas próximas subseções.

8.1 Estrutura de dados

Em outros ADBGs as sentenças visuais são realizadas em documentos XML e o modelo de sentença visual é realizado em um esquema XML.

Em ambientes baseados no SV-XML as sentenças visuais também são realizadas em documentos XML, porém o modelo de sentença visual é realizado em uma especificação XML, ao invés de um esquema XML. Até o momento nenhum outro ADBG utilizou uma especificação XML para realizar o modelo de sentença visual, esta é uma proposta original desta tese. Adicionalmente nenhum ADBG utilizou o modelo XML como um modelo de sentença visual. Esta também é uma contribuição original desta tese.

Em outros ADBGs os alfabetos são realizados em documentos XML e o meta-modelo de alfabeto é realizado em esquemas XML.

Já em ambientes baseados no SV-XML os alfabetos são realizados em esquemas XML e o meta-modelo de alfabeto é realizado em um meta-modelo de esquema XML. Esta também é uma proposta original desta tese.

As gramáticas são realizadas em documentos XML e o meta-modelo de gramática é realizado em um esquema XML. Esta configuração ocorre tanto em ambientes baseados no SV-XML quanto em outros ADBGs que utilizam componentes XML para realizar gramáticas e meta-modelos de gramáticas.

A tabela 8.1 mostra estruturas de dados típicas de ADBGs e como estas estruturas são realizadas no SV-XML e nas demais abordagens.

Tabela 8.1: Comparação da realização de estruturas de dados

Estrutura de dados de ADBGs	**Realização no SV-XML**	**Realização em outros ADBGs**
Modelo de sentença visual	Especificação XML	Esquema XML
Sentença visual	Documento XML bem formado	Documento XML válido
Meta-modelo alfabeto	Meta-modelo de esquema XML	Esquema XML
Alfabeto	Esquema XML	Documento XML
Meta-modelo de gramática	Esquema XML	Esquema XML
Gramática	Documento XML	Documento XML

8.1.1 Análise

A origem das diferenças entre a abordagem desta tese e as demais abordagens está no **modelo de sentença visual**. Todas as demais diferenças entre a abordagem SV-XML e as demais abordagens são conseqüências da escolha do SV-XML como modelo de sentença visual.

O modelo de sentença visual de um ambiente baseado no SV-XML é realizado em uma especificação do modelo XML. Já nas demais abordagens o modelo de sentença visual é realizado em um esquema XML.

O padrão XML determina que uma especificação XML defina um conjunto de regras sintáticas que devem ser aplicadas em um documento XML. Se o documento estiver de acordo com estas regras é dito que o documento está de acordo com a especificação XML.

O padrão XML também determina que um esquema XML especifique um conjunto de regras sintáticas que devem ser aplicadas em um documento XML. Se o documento estiver de acordo com estas regras é dito que o documento está de acordo com o referido esquema.

As regras especificadas num esquema XML são todas as regras da especificação XML mais um conjunto de regras especificadas no esquema XML. Logo qualquer esquema XML é mais complexo que uma especificação XML.

Como o modelo de sentença visual do SV-XML é realizado numa especificação XML e o modelo de sentença visual das demais abordagens é realizado em um esquema XML pode-se concluir que qualquer realização do modelo de sentença visual de outras abordagens é mais complexa que a realização do modelo de sentença visual do SV-XML.

Esta simplicidade beneficia tanto a compreensão de pessoas quanto a interpretação dos sistemas.

Para uma pessoa compreender a realização do modelo de sentença visual do SV-XML basta compreender as regras e entidades da especificação XML. Para compreender a realização dos modelos de outras abordagens é preciso compreender as

regras e entidades da especificação XML, mais as regras e entidades de um meta-modelo de esquemas XML (compreender como funciona uma DTD ou um XML schema) e finalmente compreender as regras definidas no esquema XML que define o modelo de sentença visual.

Para um sistema interpretar sentenças visuais de acordo com uma especificação XML, este sistema precisar conter a implementação das operações de um interpretador do modelo XML. No espaço tecnológico XML o interpretador mais conhecido é o parser XML. Um parser XML interpreta um documento XML formando uma árvore de objetos que pode ser acessada e manipulada por operações do sistema.

Por outro lado, para interpretar sentenças visuais de acordo com modelos de outras abordagens, um sistema precisa conter a implementação das operações de um interpretador do modelo XML e a implementação de operações relacionadas ao esquema. No espaço tecnológico XML o meio mais difundido de interpretar documentos de esquemas XML é utilizar um parser XML. O parser XML transforma o documento XML em uma árvore de objetos. Em seguida um conjunto extra de operações é aplicado sobre a árvore de objetos para interpretá-la de acordo com o esquema XML e eventualmente transformá-la num formato que facilite o acesso e manipulação de entidades do referido esquema.

Portanto, por definição os interpretadores de outros modelos são mais complexos que os interpretadores do modelo SV-XML já que estes precisam conter as operações do modelo SV-XML mais um conjunto extra de operações relacionadas ao esquema. Outras vantagens do uso do modelo de sentença visual SV-XML são refletidas nas demais operações e sistemas que editam sentenças visuais. Tais vantagens são analisadas na próxima seção.

Tanto as **sentenças visuais** do SV-XML quanto as sentenças visuais das demais abordagens são realizadas em documentos XML. Porém no SV-XML as sentenças visuais só precisam ser documentos XML bem-formados. Já nas demais abordagens as sentenças visuais precisam ser documentos XML bem formados e válidos (de acordo com um esquema). Portanto, entre outras operações, as verificações sintáticas das sentenças visuais do SV-XML são mais simples que as verificações sintáticas de sentenças visuais das demais abordagens.

O **meta-modelo de alfabeto** de um ambiente baseado no SV-XML é realizado em um meta-modelo de esquema XML. Já nas demais abordagens o meta-modelo de alfabeto é realizado em um esquema XML. O **alfabeto** de um ambiente baseado no SV-XML é realizado em um esquema XML. Já nas demais abordagens o alfabeto é realizado em um documento XML. A vantagem das estruturas do SV-XML é o reuso de operações e sistemas XML de maneiras não exploradas pelas outras abordagens. Maiores detalhes são dados na próxima seção.

Tanto em um ambiente baseado no SV-XML quanto em ambientes de outras abordagens os **meta-modelos de gramática** e as **gramáticas** são respectivamente realizados em um esquema XML e em um documento XML.

Portanto esta tese propõe que novos componentes XML sejam utilizados para realizar sentenças visuais, alfabetos e seus modelos. A principal vantagem destes componentes é que os mesmos são mais simples que os componentes XML utilizados em outros ADBGs. Adicionalmente, o simples fato destes componentes XML realizarem estruturas de dados de uma maneira diferente das demais abordagens já abre

novas possibilidades de se explorar componentes XML que realizem operações e sistemas em ADBGs.

8.2 Operações e sistemas

No SV-XML as operações do modo de edição de entidades básicas são realizadas por operações do modo de edição de código fonte do XML. Nas demais abordagens as operações do modo de edição de entidades básicas são realizadas por operações dos modos de operação de código fonte, estrutural livre e/ou controlado por semântica.

No SV-XML existe a seguinte correspondência entre os modos de edição típicos de ADBGs e os modos de edição XML que realizam as operações destes modos (esta correspondência é uma proposta original desta tese):

- Edição controlada por modelo de sentença visual – edição estrutural livre.
- Visão sintática de edição controlada por alfabeto – edição controlada por esquema.
- Visão sintática de edição controlada por meta-modelo de alfabeto – edição de esquema.
- Visão semântica de edição controlada por meta-modelo de alfabeto - edição de esquema controlada por semântica.

Nas demais abordagens estas 4 visões típicas de ADBGs são realizadas por operações XML do modo de edição controlada por semântica.

Os demais modos de edição e visões típicas de ADBGs não citados são realizados por operações XML do modo de edição controlada por semântica. Tanto no SV-XML quanto nas demais abordagens.

Na primeira coluna da tabela 8.2 são mostrados os modos de edição de ADBGs. No SV-XML as operações destes modos de edição são realizadas por operações do modo de edição XML mostrado na segunda coluna da tabela. Nas demais abordagens, as operações destes modos de edição são realizadas por operações do modo de edição XML mostrado na terceira coluna da tabela.

Tabela 8.2: Comparação dos componentes XML que realizam operações de ADBGs

Modos de edição de ADBGs		**Realização no SV-XML**	**Realização em outros ADBGs**
Entidades básicas		Código fonte	Código fonte, estrutural livre, controlado por esquema
Controlada por modelo de sentença visual		Estrutural livre	Controlada por semântica
Controlada por alfabeto	Visão Sintática	Controlada por esquema	Controlada por semântica
	Visão Semântica	Controlada por semântica	Controlada por semântica
Controlada por meta-modelo de alfabeto	Visão Sintática	Edição de esquema	Controlada por semântica
	Visão Semântica	Edição de esquema contr. por semântica	Controlada por semântica
Controlada por gramática	Visão Sintática	Controlada por semântica	Controlada por semântica
	Visão Semântica	Controlada por semântica	Controlada por semântica
Controlada por meta-modelo de gramática	Visão Sintática	Controlada por semântica	Controlada por semântica
	Visão Semântica	Controlada por semântica	Controlada por semântica

As diferenças nos modos de edição são refletidas nos editores que podem ser usados para realizar estas operações. Tais diferenças podem ser vistas na tabela 8.3.

Na primeira coluna da tabela 8.3 são mostrados os modos de edição de ADBGs. No SV-XML as operações destes modos de edição são realizadas pelos editores XML mostrados na segunda coluna da tabela. Nas demais abordagens as operações destes modos de edição são realizadas pelos editores XML mostrados na terceira coluna da tabela.

Tabela 8.3: Comparação dos componentes XML que realizam editores de ADBGs

Modos de edição de ADBGs		**Editores XML utilizados no framework SV-XML**	**Editores XML utilizados em outros ADBGs**
Entidades básicas		Editor de código fonte	Editor de código fonte, editor XML, editor XML controlado por esquema
Controlada por modelo de sentença visual		Editor XML	Extensão de editor XML controlado por esquema
Controlada por alfabeto	Visão Sintática	Editor XML controlado por esquema	Extensão de editor XML controlado por esquema
	Visão Semântica	Extensão de editor XML controlado por esquema	Extensão de editor XML controlado por esquema
Controlada por meta-modelo de alfabeto	Visão Sintática	Editor de esquema	Extensão de editor XML controlado por esquema
	Visão Semântica	Extensão de editor de esquema XML	Extensão de editor XML controlado por esquema
Controlada por gramática	Visão Sintática	Extensão de editor XML controlado por esquema	Extensão de editor XML controlado por esquema
	Visão Semântica	Extensão de editor XML controlado por esquema	Extensão de editor XML controlado por esquema
Controlada por meta-modelo de gramática	Visão Sintática	Extensão de editor XML controlado por esquema	Extensão de editor XML controlado por esquema
	Visão Semântica	Extensão de editor XML controlado por esquema	Extensão de editor XML controlado por esquema

São propostas originais desta tese o uso do editor XML, do editor XML controlado por esquema, do editor de esquema e da extensão de editor de esquema XML para respectivamente realizar as operações do modo de edição controlada por modelo de sentença visual, da visão sintática do modo de edição controlada por alfabeto, da visão sintática do modo de edição controlada por meta-modelo de alfabeto e da visão semântica do modo de edição controlada por meta-modelo de alfabeto.

8.2.1 Análise - propagação da complexidade

Os modos de edição XML de código fonte, estrutural livre, controlado por esquema e de edição de esquemas são independentes da área de domínio. Ou seja, as mesmas operações que podem ser aplicadas em documentos de uma área de domínio podem ser aplicadas em documentos de qualquer outra área de domínio. Independentemente da área de domínio as operações destas visões destacam e facilitam a edição de entidades de seu respectivo modelo (caracteres, entidades do modelo XML, entidades de um esquema XML etc). Vários sistemas que realizam estas operações já estão a disposição e podem ser utilizados para auxiliar a edição de documentos de qualquer área de domínio.

No caso dos demais ADBGs, as operações XML independentes de domínio só podem realizar as operações de edição de entidades básicas. Nestes ADBGs um editor XML controlado por esquema tem mais recursos que auxiliam a correta edição de sentenças visuais que um editor de código fonte, mas mesmo assim ele continua sendo apenas um editor de entidades básicas. Ele não auxilia a editar os dados de uma sentença visual como dados de uma sentença visual, ele não destaca as estruturas do modelo de sentença visual. Ao invés disto ele destaca as estruturas do modelo XML.

Já no caso do SV-XML as operações XML independentes de domínio podem realizar operações de: (i) edição de entidades básicas, (ii) edição controlada por modelo de sentença visual, (iii) visão sintática de edição controlada por alfabeto e (iv) visão sintática de edição controlada por meta-modelo de alfabeto.

As operações do modo estrutural livre do XML são de fato operações controladas por modelo de sentença visual. Isto ocorre porque qualquer editor XML padrão auxilia a edição de documentos XML destacando as entidades do modelo XML. Como o modelo XML é o modelo de sentença visual do SV-XML, qualquer editor XML padrão auxilia a edição de sentenças visuais destacando entidades do modelo de sentença visual.

De forma análoga, um editor XML controlado por esquema contém de fato operações de uma visão sintática do modo de edição controlado por alfabeto. Estes editores auxiliam a edição de sentenças visuais destacando entidades de um alfabeto, pois no SV-XML os alfabetos são realizados por esquemas XML. Já um editor de esquemas XML contém de fato operações de uma visão sintática do modo de edição controlado por meta-modelo de alfabeto. Estes editores auxiliam a edição de alfabetos destacando entidades do meta-modelo de alfabeto, pois no SV-XML os meta-modelos de alfabetos são realizados por meta-modelos de esquemas XML.

Em síntese, no SV-XML as operações XML independentes de domínio são suficientes para atender todas as necessidades e requisitos de **4** tipos de visões típicas de ADBGs (Dentre os 10 tipos de visões possíveis). Já nas demais abordagens, as mesmas operações XML independentes de domínio são suficientes para atender todas as necessidades e requisitos de apenas **1** tipo de visão típica de ADBGs (a visão do modo de edição de entidades básicas).

A utilização de sistemas e editores XML padrão também torna clara e evidente a diferença de complexidade entre a abordagem proposta nesta tese e as demais abordagens.

Num ADBG baseado no SV-XML, um editor XML controlado por esquema é suficiente para atender todas as necessidades e requisitos das visões sintáticas do modo de edição controlado por alfabeto. Este editor contém um editor XML e um editor de texto que por sua vez são suficientes para atender todas as necessidades e requisitos das visões sintáticas do modo de edição controlado por modelo de sentença visual e de entidades básicas respectivamente. Portanto um editor XML controlado por esquema contém as operações e algoritmos capazes de editar as estruturas de dados de sentenças visuais como um conjunto de entidades básicas, como um conjunto de entidades do modelo de sentença visual e como um conjunto de entidades tipadas de um alfabeto.

Já em um ADBG baseado em outro modelo de sentença visual, exatamente o mesmo editor XML controlado por esquema só é suficiente para atender as necessidades e requisitos do modo de edição de entidades básicas. Para editar uma sentença visual como um conjunto de entidades do modelo de sentença visual é preciso que novas

operações e algoritmos sejam adicionados ao editor. Portanto o editor precisa ser estendido para atender a estas necessidades. Para editar uma sentença visual como um conjunto de entidades tipadas de um alfabeto é preciso que novas operações e algoritmos sejam adicionados ao editor (além das já citadas anteriormente). Portanto o editor precisa ser novamente estendido para atender a estas necessidades.

Assim, para um editor controlado por esquema realizar em outros ADBGs as mesmas operações que ele realiza em um ambiente baseado no SV-XML, ele precisa ser estendido. E, logicamente, estas extensões são mais complexas (com mais operações e algoritmos) que o editor original.

Como já foi dito esta diferença de complexidade tem origem na definição do componente que realiza o modelo de sentença visual do SV-XML. Os componentes que realizam os modelos de sentença visual dos demais ADBGs são mais complexos que o uma especificação XML. Esta complexidade é propagada para as demais estruturas e operações do sistema. Uma demonstração desta complexidade superior em outras abordagens é que no SV-XML um editor XML controlado por esquema realiza operações de uma visão sintática do modo de edição controlado por alfabeto. Já nos demais ADBGs o mesmo editor só pode realizar visões de um editor de entidades básicas.

8.2.2 Análise- Integração entre recursos XML independentes de domínio e ADBGs

Além de prover componentes XML mais simples que os correspondentes componentes XML utilizados em outros ADBGs, a proposta desta tese aumenta o suporte que operações e sistemas XML independentes de domínio podem dar aos ADBGs. No SV-XML, todas as necessidades e requisitos de 4 tipos de visões de ADBGs podem ser realizadas exclusivamente por operações XML independentes de domínio. Nas demais abordagens só as necessidades e requisitos de 1 tipo de visão podem ser atendidas exclusivamente por operações XML independentes de domínio.

Do ponto de vista do desenvolvedor de sistemas visuais interativos, o suporte a componentes XML independentes de domínios permite a utilização e integração de sistemas externos (fora do ADBG) no processo de especificação de sistemas visuais interativos. Quando é possível utilizar sistemas externos pode-se editar/visualizar os documentos do ADBG mesmo quando o ADBG não está disponível ou não é desejado. Adicionalmente têm-se muitas alternativas de edição. Desenvolvedores podem, por exemplo, utilizar editores XML externos com os quais estão mais acostumados, que tem mais recursos ou que tem diferentes interfaces. Depois estes documentos podem ser interpretados pelo ADBG como se tivessem sido editados no ADBG.

Já do ponto de vista dos desenvolvedores de ADBGs, o suporte a componentes XML independentes de domínio facilita a implementação destes ADBGs já que grande parte do ambiente pode reutilizar sistemas e operações XML prontos, amplamente testados e difundidos.

Portanto, em ambos os casos, o suporte a componentes XML independentes de domínio possibilitam que recursos XML já prontos, amplamente testados e difundidos sejam integrados aos ADBGs sem a necessidade de adaptações nestes recursos. No SV-XML este tipo de integração pode ocorrer em 4 dos 10 tipos de visões existentes. Já nas demais abordagens este tipo de integração só pode ocorrer em 1 tipo de visão.

Uma desvantagem que a proposta desta tese pode causar para desenvolvedores de ADBGs é a necessidade de adaptação a uma nova arquitetura. De forma geral, desenvolvedores que utilizam componentes XML estão mais acostumados a definir dados em documentos XML e estruturas de dados em esquemas XML. No caso do framework SV-XML os desenvolvedores teriam que se adaptar a uma arquitetura onde os dados de um alfabeto são definidos em um esquema XML e a estrutura destes dados é definida em um meta-modelo de esquema XML. Adicionalmente os desenvolvedores teriam que se adaptar ao fato que as sentenças visuais precisam ser apenas documentos XML bem formados no lugar de documentos XML válidos.

9 CONCLUSÃO

Esta tese apresenta como principais resultados a demonstração de que o modelo XML pode ser o modelo de sentença visual de um ADBG e a proposta de um framework de projeto físico de ADBGs formado por um conjunto de componentes XML. Sem perder funcionalidades, os ADBGs construídos a partir do framework proposto terão componentes XML mais simples que os componentes XML de ADBGs construídos a partir de outras abordagens. Adicionalmente os componentes XML aqui propostos aumentam o suporte que operações e sistemas XML independentes de domínio podem dar aos ADBGs. Para alcançar e demonstrar estes resultados a presente pesquisa executou os passos descritos a seguir.

Primeiramente foi definida a utilização do modelo XML nesta pesquisa. A interpretação original do modelo XML define que suas entidades especificam uma estrutura de dados. Sendo que tal estrutura pode descrever qualquer tipo de informação de qualquer área de domínio. Esta tese propôs uma interpretação mais específica do modelo XML. Foi proposto que as entidades do modelo XML sejam interpretadas como entidades de modelos de sentença visual.

Um mapeamento entre entidades do modelo XML e entidades típicas de modelos de sentença visual demonstrou que o modelo XML pode apropriadamente ser um modelo de sentença visual tendo as características típicas de um modelo de sentença visual baseado em atributos. Adicionalmente, foi formalmente demonstrada a expressividade do modelo XML interrpretado como um modelo de sentença visual (SV-XML). O modelo SV-XML é capaz de representar qualquer sentença visual definida por uma gramática livre do contexto, uma *Constraint Multiset Grammar*, uma *Picture Layout Grammar* ou uma gramática vCARW. Portanto o SV-XML é tão expressivo quanto os principais modelos de sentença visual atualmente utilizados, como por exemplo, os modelos utilizados nos ADBGs Genial, Diagen, GenGed e VLDesk.

Após a definição do modelo de sentença visual, foi especificado como um ADBG que utiliza este modelo pode ser realizado por componentes XML. O padrão XML determina que a realização do modelo XML seja uma especificação do modelo XML como, por exemplo, a especificação XML 1.1 da W3C. Por isso foi definido que uma especificação XML realiza o modelo de sentença visual. Assim como o modelo de sentença visual as demais estruturas de dados de um ADBG que segue as recomendações do framework SV-XML (sentenças visuais, meta-modelo de alfabeto, alfabeto, meta-modelo de gramática e gramática) são realizadas por componentes XML (documentos XML, esquemas XML e meta-modelos de esquemas XML). Também

através de mapeamentos foi demonstrada a compatibilidade entre as estruturas de dados típicas de ADBGs e os componentes XML que realizam estas estruturas no SV-XML.

A seguir foram definidas as operações que poderiam manipular as estruturas de um ADGB SV-XML. Logo foi feito um mapeamento entre as operações típicas de ADBGs e os componentes XML que realizam estas operações. Neste mapeamento as operações típicas de ADBGs e típicas do espaço tecnológico XML foram primeiramente categorizadas em modos de edição segundo os critérios de Quint (2004). Basicamente este critério classifica as operações de acordo com conjunto de regras que as controlam. Depois de categorizadas, os modos de edição de ADBGs foram mapeados para os modos de edição XML. Este mapeamento define que as operações dos modos de edição ADBG são realizadas por componentes XML dos correspondentes modos de edição XML. Para todos os modos de edição ADBG foi demonstrado que existe um modo de edição XML que pode realizar suas operações.

No espaço tecnológico XML, associada a cada modo de edição XML existe um conjunto de componentes e artefatos XML que podem ser utilizados para a implementação destes modos de edição. Estes artefatos vão desde técnicas de desenvolvimento de softwares e recomendações até APIs, ferramentas e sistemas XML. Por isso, além de definir características gerais sobre como os modos de edição ADBG são realizados, o referido mapeamento de operações indiretamente informa um conjunto de artefatos XML que podem ser (re)utilizado para a implementação de cada modo de edição de um ADBG SV-XML.

Nesta tese foram abordados alguns sistemas XML que podem ser utilizados como parte de um ADBG SV-XML. Estes sistemas são editores típicos da tecnologia XML que já encapsulam operações e estruturas de dados que podem ser utilizadas em ADBGs. Para facilitar o uso de editores do XML em ADBGs SV-XML esta tese propôs um mapeamento entre os editores típicos de ADBGs (editor de sentença visual, editor de alfabeto e editor de gramática) e editores do XML (editor de código fonte, editor XML, editor controlado por esquema, extensão de editor controlado por esquema, editor de esquema e extensão de editor controlado por esquema). Os editores XML do mapeamento são sistemas que realizam as funções dos correspondentes sistemas típicos de ADBGs. Foi demonstrado que para cada editor ADBG existe um conjunto de editores XML que pode realizá-lo.

Assim, as estruturas de dados, operações e sistemas do framework SV-XML foram especificadas. Estas especificações são um framework de projeto físico que pode auxiliar a implementação e construção de ADBGs que utilizam componentes XML.

O uso do framework é modular e flexível. Por definição, o modelo de sentença visual de um ADBG SV-XML é sempre o modelo XML e a sua realização é uma especificação XML. Todas as demais estruturas de dados, operações e sistemas do framework SV-XML são opcionais. Pode-se utilizar apenas as estruturas de dados do framework e não suas operações e sistemas. Alternativamente, pode-se utilizar as estruturas de dados e operações do SV-XML, porém estas operações e estruturas podem estar encapsuladas em sistemas diferentes dos aqui apresentados. Também é possível utilizar o suporte a um modelo ou meta-modelo do ADBG e não o suporte a outro modelo. Por exemplo, pode-se utilizar o suporte ao alfabeto do SV-XML e não o seu suporte a gramática. Enfim o uso do framework SV-XML pode ser parcial e combinado com outros componentes XML e até mesmo com componentes fora do espaço

tecnológico XML. Contudo quando alguma estrutura de dados, operação ou sistema do framework não é utilizada pode-se perder algumas de suas vantagens e características.

O framework auxilia a implementação de qualquer ADBG cuja gramática seja compatível com um modelo de sentença visual baseado em atributos. Esta limitação deve-se as características do modelo de sentença visual adotado.

O passo seguinte a especificação do framework foi a implementação de um ADBG SV-XML. O ADBG implementado chama-se Vitransf. Esta implementação demonstrou concretamente a viabilidade e aplicabilidade de um ADBG SV-XML. O ambiente de desenvolvimento do Vitransf disponibiliza um conjunto de visões de editores XML padronizados. Com a ajuda destes editores o desenvolvedor especifica uma gramática VCARW. Os dados desta gramática são armazenados em documentos XML. Um conjunto de rotinas do Vitransf pode interpretar estes documentos XML e gerar o sistema visual interativo especificado na gramática.

O sistema visual interativo gerado é um documento SVG e um conjunto de ECMAScripts. O documento SVG descreve gráficos 2D, os scripts descrevem alterações que podem ocorrer nestes gráficos. As alterações são disparadas por eventos XML que podem ocorrer na interação entre o usuário e o documento SVG. Navegadores Web padrão com suporte a SVG e ECMAScripts podem interpretar os documentos e scripts gerados pelo ambinete de desenvolvimento. Portanto, navegadores Web padrão, como o Internet Explorer e o Mozilla, são os ambientes de execução do Vitransf. Para demonstrar a aplicabilidade do Vitransf foram apresentados dois estudos de caso onde sistemas visuais interativos foram especificados através de uma gramática Vitransf e executados em navegadores Web.

Complementaram a pesquisa relatada nesta tese o estudo e a descrição das características dos ADBGs existentes bem como a análise e comparação entre as abordagens existentes e as propostas desta tese. Tal análise é detalhada nas próximas sub-seções.

9.1 Propostas originais

Outros ADBGs utilizam componentes XML para realizar estruturas de dados, operações e sistemas, entretanto o framework de ADBGs SV-XML possui as seguintes propostas originais.

Nas estruturas de dados do framework SV-XML são propostas originais:

- o uso do modelo XML como modelo de sentença visual;
- o uso de uma especificação XML para realizar o modelo de sentença visual;
- o uso de documentos XML bem formados para realizar sentenças visuais;
- o uso de esquemas XML para realizar alfabetos;
- o uso de meta-modelos de esquema XML para realizar os meta-modelos de alfabeto;

Nas operações do framework SV-XML são propostas originais:

- o uso de componentes XML do modo de edição estrutural livre para realizar operações do modo de edição controlada por modelo de sentença visual ;

- o uso de componentes XML do modo de edição controlada por esquema para realizar operações de uma visão sintática do modo de edição controlada por alfabeto;
- o uso de componentes XML do modo de edição de esquema para realizar operações da visão sintática do modo de edição controlada por meta-modelo de alfabeto ;
- o uso de componentes XML do modo de edição de esquema controlada por semântica para realizar operações da visão semântica de edição controlada por meta-modelo de alfabeto.

Nos sistemas do framework SV-XML são propostas originais:

- o uso do editor XML para realizar as operações do modo de edição controlada por modelo de sentença visual;
- o uso do editor XML controlado por esquema para realizar as operações da visão sintática do modo de edição controlada por alfabeto;
- o uso do editor de esquema XML para realizar as operações da visão sintática do modo de edição controlada por meta-modelo de alfabeto;
- o uso da extensão de editor de esquema XML para realizar as operações da visão semântica do modo de edição controlada por meta-modelo de alfabeto.

A implementação do protótipo Vitransf também é uma contribuição original pois implementa as propostas originais aqui citadas. A tabela 5.3 e a figura 5.2 sintetizam o framework SV-XML, já a tabela 6.2 sintetiza o uso do framework SV-XML na implementação do Vitransf.

Quint et al. (2004) dividiu as operações XML de acordo com as regras que comandam estas operações. Já o uso destes critérios para dividir e categorizar as operações dos ADBGs é uma proposta original desta tese. Portanto a definição dos modos de edição dos ADBGs é uma contribuição desta tese. Consequentemente os seguintes mapeamentos também são propostas originais desta tese:

- o mapeamento entre os modos de edição dos ADBGs e os editores ADBGs que encapsulam as operações destes modos de edição (editor de sentença visual, editor de alfabeto e editor de gramática). As relações deste mapeamento são válidas para qualquer ADBG, independentemente do ADBG utilizar ou não componentes XML;
- o mapeamento entre os modos de edição dos ADBGs existentes e modos de edição XML. Os modos de edição ADBG do mapeamento descrevem as funcionalidades do ADBG. Já os correspondentes modos de edição XML descrevem os componentes XML utilizados por estes ADBGs para realizar suas funcionalidades;
- o mapeamento entre os modos de edição dos ADBGs existentes e os editores XML típicos utilizados para realizar as operações destes modos de edição.

Tais mapeamentos estão sintetizados na tabela 5.3.

9.2 Contribuições

Esta seção analisa que contribuições/vantagens as propostas originais desta tese tem em relação aos trabalhos e propostas existentes.

Uma contribuição secundária desta tese foi a especificação dos modos de edição de ADBGs. Os critérios utilizados nesta taxonomia são claros, diretos e já foram aplicados e validados em outras áreas de domínio. Nos ADBGs estes critérios foram aplicados da mesma maneira e com os mesmos objetivos que a aplicação destes critérios no espaço tecnológico XML. Portanto os benefícios desta aplicação nos dois espaços tecnológicos são similares. Os modos de edição podem ser usados para análise e verificação das funcionalidades e operações de um ADBG. Esta tese utilizou amplamente os modos de edição exatamente com estes objetivos: verificação e análise de funcionalidades e operações de ADBGs. O mapeamento dos modos de edição de ADBGs para os editores que encapsulam estes modos de edição, por exemplo, permitiu uma descrição compacta e de alto nível de várias características de cada editor.

Adicionalmente, tanto os modos de edição de ADBGs propostos quanto o referido mapeamento são válidos para qualquer ADBG, independentemente do ADBG utilizar ou não componentes XML. Isto amplia as possibilidades de uso destas especificações e mapeamentos.

A literatura da área provê taxonomias que classificam a expressividade de gramáticas de ADBGs, os tipos de sentenças visuais e interações que podem ser geradas em ADBGs, as interfaces utilizadas para especificar gramáticas, os formalismos utilizados para especificar gramáticas entre outras opções. Entretanto não foi encontrada na literatura da área nenhuma taxonomia que classifique todas as operações de edição dos ADBGs.

Outra contribuição secundária desta tese foi a implementação do Vitransf que poderá servir de plataforma de testes para futuras pesquisas nos temas discutidos nesta tese, principalmente em pesquisas que relacionam os espaços tecnológicos XML e linguagens visuais.

Já as contribuições primárias e objetivos principais desta tese estão relacionados com o modelo e o framework SV-XML. A seguir serão destacadas as contribuições do uso de componentes XML no SV-XML em relação ao uso de componentes XML em outros ADBGs.

O modelo de sentença visual de um ambiente baseado no SV-XML é realizado por uma especificação XML. Já nas demais abordagens o modelo de sentença visual é realizado por um esquema XML. Logo qualquer realização do modelo de sentença visual de outras abordagens é mais complexa que a realização do SV-XML. Esta simplicidade facilita a interpretação de sentenças visuais tanto por pessoas quanto por sistemas automatizados.

No SV-XML as sentenças visuais são realizadas por documentos XML bem-formados. Já nas demais abordagens as sentenças visuais são realizadas por documentos XML bem formados e válidos (de acordo com um esquema). Portanto as verificações sintáticas das sentenças visuais do SV-XML são mais simples que as verificações sintáticas de sentenças visuais realizadas em documentos XML de outros ADBGs.

A maior simplicidade do modelo de sentença visual e das sentenças visuais do SV-XML é propagada para as demais estruturas de dados do SV-XML e para as operações e

sistemas que utilizam estas estruturas. Duas demonstrações da maior simplicidade dos sistemas do SV-XML em relação aos sistemas dos demais ADBGs são descritas a seguir:

- Um editor XML controlado por esquema realiza no SV-XML as operações e algoritmos capazes de editar as estruturas de dados de sentenças visuais como um conjunto de entidades básicas, como um conjunto de entidades do modelo de sentença visual e como um conjunto de entidades tipadas de um alfabeto. Já em um ADBG baseado em outro modelo de sentença visual, exatamente o mesmo editor XML controlado por esquema só é suficiente para atender as necessidades e requisitos do modo de edição de entidades básicas. Portanto, para um editor controlado por esquema realizar em outros ADBGs as mesmas operações que ele realiza em um ambiente baseado no SV-XML, ele precisa ser extendido. Logicamente, estas extensões são mais complexas (com mais operações e algoritmos) que o editor original.
- No SV-XML, todas as necessidades e requisitos de 4 tipos de visões de ADBGs podem ser realizadas exclusivamente por operações XML independentes de domínio. Nas demais abordagens só as necessidades e requisitos de 1 tipo de visão podem ser realizadas exclusivamente por operações XML independentes de domínio.

Do ponto de vista do desenvolvedor de sistemas visuais interativos e do desenvolvedor de ADBGs, a menor complexidade do SV-XML significa manipular componentes XML mais simples que os componentes XML disponíveis em outros ADBGs.

Outra vantagem do SV-XML em relação às outras propostas é que o SV-XML aumenta o suporte que operações e sistemas XML independentes de domínio podem dar aos ADBGs. Este suporte possibilita que componentes XML já prontos, amplamente testados e difundidos sejam integrados aos ADBGs sem a necessidade de adaptações nestes componentes. No SV-XML este tipo de integração pode ocorrer em 4 dos 10 tipos de visões existentes. Já nas demais abordagens este tipo de integração só pode ocorrer em 1 tipo de visão.

Do ponto de vista do desenvolvedor de sistemas visuais interativos, o suporte a componentes XML independentes de domínios permite a utilização e integração de sistemas externos (fora do ADBG) no processo de especificação de sistemas visuais interativos. Quando é possível utilizar sistemas externos pode-se editar/visualizar os documentos do ADBG mesmo quando o ADBG não está disponível ou não é desejado. Adicionalmente têm-se muitas alternativas de edição. Desenvolvedores podem, por exemplo, utilizar editores XML externos com os quais estão mais acostumados, que tem mais recursos ou que tem diferentes interfaces. Depois estes documentos podem ser interpretados pelo ADBG como se tivessem sido editados no ADBG.

Já do ponto de vista dos desenvolvedores de ADBGs, o suporte a recursos XML independentes de domínios facilita a implementação destes ADBGs, pois, grande parte do ambiente pode reutilizar sistemas e operações XML já prontos, amplamente testados e difundidos. O potencial de reuso de componentes XML do framework SV-XML também foi demonstrado na implementação do Vitransf. O ambiente de execução do Vitransf é um navegador Web padrão por isso não precisou ser implementado. Quanto ao ambiente de desenvolvimento do Vitransf, dentre as 6 visões implementadas 5 visões reutilizaram ferramentas XML padronizadas, nenhuma implementação foi feita nestas

ferramentas para que elas fossem reaproveitadas no Vitransf. Somente a visão semântica das regras Vitransf precisou ser implementada a parte, mesmo assim muitos recursos básicos do XML foram reaproveitados.

O reuso dos componentes XML facilitou a implementação do Vitransf já que grande parte do sistema reutilizou componentes XML já prontos, amplamente testados e difundidos. Restou para ser codificada apenas a implementação da lógica da gramática que estava concentrada em apenas dois conjuntos de rotinas. Um destes conjuntos de rotinas interpreta documentos XML (a gramática Vitransf) e a partir deles gera um sistema visual interativo. O outro conjunto de rotinas implementadas especifica a interface gráfica da visão semântica das regras Vitransf.

Em síntese as principais vantagens do uso de componentes XML no SV-XML em relação ao uso de componentes XML em outros ADBGs são:

- O uso de estruturas de dados, operações e sistemas XML mais simples que outras abordagens.
- O aumento do suporte que operações e sistemas XML independentes de domínio podem dar aos ADBGs.

Tais conclusões validam as duas últimas hipóteses desta tese.

Por fim, como ja foi citado, no framework SV-XML o modelo XML é o modelo de sentença visual e os componentes são componentes XML compatíveis com este modelo. Tais características do SV-XML validam a primeira hipótese desta tese.

9.3 Trabalhos futuros

Esta tese propôs que diferentes componentes XML sejam utilizados para realizar sentenças visuais e alfabetos em ADBGs. Adicionalmente foram dadas as características gerais das operações e sistemas que podem manipular estas estruturas. O (re)uso de algumas operações e sistemas XML no SV-XML foram detalhados. Este foi o caso das operações de verificação sintática, das operações de detecção/reação a eventos XML e dos editores XML.

Entretanto muitos outros sistemas e operações XML podem explorar estas novas estruturas e se adaptar ao SV-XML. Alguns sistemas e operações XML podem ser utilizados em vários ADBGs e fazer parte do framework SV-XML, outros podem ser utilizados em um ADBG específico.

O Vitransf, por exemplo, reutilizou a semântica do SVG, suas operações e editores para definir a aparência e os comandos dos símbolos de uma sentença visual. O reuso de uma linguagem XML da forma que o SVG foi reutilizado no Vitransf só é possível quando o modelo de sentença visual do ADBG é uma especificação XML e o alfabeto é um esquema XML. Logo, o Vitransf apresentou uma nova maneira de se explorar linguagens XML em ADBGs.

Portanto as propostas desta tese abrem novas possibilidades de se explorar componentes XML em ADBGs. A exploração e estudo destas novas possibilidades é uma linha promissora e extensa de pesquisas e trabalhos futuros já que existem centenas de operações e sistemas XML e novas operações e sistemas são criados todos os dias.

Esta tese apresentou novos componentes XML que podem representar, definir e manipular alfabetos e sentenças visuais. Entretanto os componentes XML que podem

representar, definir e manipular gramáticas são os mesmos de outras abordagens. Uma linha de pesquisas e trabalhos futuros interessante é o estudo de novos componetes XML que possam representar, definir e manipular gramáticas.

O motivo desta tese não porpor novos componentes XML para representar gramáticas é que as gramáticas são bem mais complexas e variadas que os alfabetos e sentenças visuais. As gramáticas têm vários ítens e estes ítens variam muito de gramática para gramática. Algumas gramáticas têm sentenças visuais iniciais, outras não; algumas têm axiomas outras não; algumas têm símbolos terminais e não terminais, outras têm somente símbolos terminais; as regras de algumas gramáticas contém 2 partes (eg. LHS e RHS) outras contém 3 partes, outras 4, outras 1; algumas são combinadas com meta-modelos; algumas gramáticas são baseadas em textos, outras em grafos e assim por diante.

Uma abordagem interessante para esta linha de pesquisa é estudar cada item de cada tipo de gramática a parte. Neste estudo novas maneiras de se representar gramáticas e seus meta-modelos podem ser pesquisadas. Depois os resultados destes estudos poderiam extender o framework SV-XML ou gerar especializações do SV-XML para tipos de gramáticas específicos (um SV-XML para gramáticas de grafos, outro para gramáticas baseadas em símbolos com atributos etc). Devido ao grande número de tipos de gramáticas e de componentes XML existentes, esta também é uma linha de pesquisa promissora e extensa.

Por fim, embora os estudos desta tese estejam focalizados em ambientes de desenvolvimento baseados em gramática (ADBGs), as propostas aqui apresentadas podem ser utilizadas em outros tipos de ambientes de desenvolvimento. Foi demonstrado, por exemplo, que os componentes aqui propostos podem ser úteis em ambientes híbridos como o VLDesk (COSTAGLIOLA, 2005). Tais ambientes são baseados em gramáticas e em meta-modelos. Também foi demonstrado que os componentes aqui propostos podem ser úteis em ambientes baseados em meta-modelos que utilizam alfabetos como, por exemplo, o Pounamu (ZHU, 2007). De forma mais geral pode-se utilizar os componentes propostos nesta tese em qualquer ambiente de desenvolvimento que precise de componentes XML para representar alfabetos e sentenças visuais. Entretanto são necessários estudos mais detalhados sobre os efeitos da aplicação das propostas desta tese em outros tipos de ambiente. Tais estudos são portanto um interessante tema para futuras pesquisas.

REFERÊNCIAS

ASCHENBRENNER, P.; SCHURR, A. Generating interactive animations from visual specifications. In: IEEE SYMPOSIUM ON HUMAN CENTRIC COMPUTING LANGUAGES AND ENVIRONMENTS, HCC, 2003, Auckland, New Zeland. **Proceedings…** Piscataway, NJ: IEEE Computer Society, 2003. p.169-176.

BARDOHL, R. et al. Application of Graph Transformation to Visual Languages. In: **Handbook of Graph Grammars and Computing by Graph Transformation**. Singapore: World Scientific, 1999. v.2, p.105-180.

BARDOHL, R. et al. **GENGED**: Generation of Graphical Environments for Design. Part I: User Manual. Berlin: Institut für Softwaretechnik und Theoretische Informatik Technische Universität, 2003. Relatório Técnico. Disponível em: < http://tfs.cs.tu-berlin.de/genged/detail.html>. Acesso em: abril 2008.

BOTTONI, P. et al. The theory of visual sentences to formalize interactive visual messages. In: FERRI F. (Ed.). **Visual Languages for Interactive Computing**: Definitions and Formalization. Hershey, USA: Idea Group Publishers, 2007. p.1-21.

BOTTONI, P. et al. Definition of visual processes in a language for expressing transitions. **Journal of Visual Languages and Computing**, London, v.15, n.3-4, p.211-242, 2004.

BOTTONI, P. Dynamic aspects of visual modelling languages. **Electronic Notes in Theoretical Computer Science**, Amsterdam, v.82, n.7, p. 120-132, 2003.

BOTTONI, P.; COSTABILE, M.F.; MUSSIO, P. Specification and dialogue control of visual interaction through visual rewriting systems. **ACM Transactions on Programming Languages and Systems**, New York, v.21, n.6, p.1077 – 1136, 1999.

BRAY, T. et al. **eXtensible Markup Language (XML) 1.1 2nd ed**. World Wide Web Consortium recommendation. 2006a. Disponível em: <http://www.w3.org/TR/xml11/>. Acesso em: maio 2008.
BRAY,T. et al. **Namespaces in XML 1.0 2nd ed**. World Wide Web Consortium recommendation, 2006b. Disponível em: <http://www.w3.org/TR/xml-names/> Acesso em: maio 2008.

BRIELER, F.; MINAS, M. Ambiguity resolution for sketched diagrams by syntax analysis based on graph grammars. In: INTERNATIONAL WORKSHOP ON GRAPH TRANSFORMATION AND VISUAL MODELING TECHNIQUES, GT-VMT, 7., 2008. **Proceedings…** [S.l.: s.n.], 2008.

CLARK, J.; MAKOTO, M. **RELAX NG Specification**. [S.l.]: Organization for the Advancement of Structured Information Standards (OASIS), 2001, Relatório Técnico.

COSTAGLIOLA, G.; DEUFEMIA, V.; POLESE, G.; RISI, M. Building syntax-aware editors for visual languages. **Journal of Visual Languages and Computing**, London, v.16, n.6, p. 508-540, 2005.

COSTAGLIOLA, G.; DEUFEMIA, V.; POLESE, G. A framework for modeling and implementing visual notations with applications to software engineering. **ACM Transactions on Software Engeneering Methodologies**, New York, v.13, n.4, p. 431–487, 2004.

COSTAGLIOLA, G. et al. A classification framework to support the design of visual languages. **Journal of Visual Languages and Computing**, London, v.13, n.6, p. 573-600, 2002.

COWAN. J.; TOBIN, R. **XML Information Set 2nd ed.** World Wide Web Consortium recommendation. 2004. Disponível em: < http://www.w3.org/TR/xml-infoset/> Acesso em: maio 2008.

DREWES, F. et al. Rule-based programming with DiaPlan. In: INTERNATIONAL WORKSHOP ON GRAPH-BASED TOOLS, 2004, Rome, Italy. **Proceedings…** [S.l.: s.n.], 2004. p. 15–26.

ERMEL, C.; EHRIG, K. Visualization, Simulation and Analysis of Reconfigurable Systems using TIGER. In: APPLICATIONS OF GRAPH TRANSFORMATION WITH INDUSTRIAL RELEVANCE, 2007, Kassel, Germany. **Proceedings…** [S.l.: s.n.], 2007. p. 261-266.

ERMEL,C.; BARDOHL,R. Scenario Animation for Visual Behavior Models: A Generic Approach. **Journal on Software and Systems Modeling**, Heidelberg, v.3, n.2, p.164-177, 2004.

FERRAIOLO, J.; FUJISAWA, J., JACKSON, D. **Scalable Vector Graphics (SVG) 1.1 Specification.** W3C Recommendation. 2003. Disponível em: <http://www.w3.org/TR/SVG11/> Acesso em: maio 2008.

FERRI, F. (Ed.). **Visual Languages for Interactive Computing**: Definitions and Formalization. Hershey,USA: Idea Group Publishers, 2007.

FURNAS, G. W.; QU, Y. Using pixel rewrites for shape-rich interaction. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, SIGCHI, 2003. **Proceedings…** New York, NY: ACM Press, 2003. p.369-376.

GOLIN, E. J. Parsing Visual Languages with Picture Layout Grammars. **Journal of Visual Languages and Computing,** London, v.2, p. 371 – 393, 1991.

GUERRA, E. ; LARA, J. Meta-modelling and graph transformation for the definition of multi-view visual languages. FERRI F. (Ed.). **Visual Languages for Interactive Computing:** Definitions and Formalization**.** Hershey, USA: Idea Group Publishers, 2007. p. 74-101.

HELM, R.; MARRIOTT, K. A Declarative Specification and Semantics of Visual Languages. **Journal of Visual Languages and Computing**, London, v.2, p. 311– 331, 1991.

HOLT, R. C. et al. GXL: a graph-based standard exchange format for reengineering. **Science of Computer Programming**, Amsterdam, v.60, n.2, p.149-170, 2006.

ISO/IEC. **ISO/IEC 19757**: Document Schema Definition Languages (DSDL). [S.l.], 2004. Disponível em: < http://dsdl.org/> Acesso em: maio 2008.

KURTEV, I.; BÉZIVIN, J.; AKSIT, M. Technological Spaces: An Initial Appraisal. CONFEDERATED INTERNATIONAL CONFERENCES, CoopIS, DOA and ODBASE, 2002, Irvine, CA, USA. **Proceedings…** [S.l.: s.n.], 2002.

LARA J.; VANGHELUWE, H. Defining visual notations and their manipulation through meta-modelling and graph transformation. **Journal of Visual Language and Computing**, London, v.15, n.3-4, p.309-330, 2004.

McCARRON, S. et al. **XML Events. An events syntax for XML**. World Wide Web Consortium recommendation. 2003. Disponível em: <http://www.w3.org/TR/xml-events/>. Acesso em: maio 2008.

MINAS, M. VisualDiaGen - a tool for visually specifying and generating visual editors. In: INTERNATIONAL WORKSHOP ON APPLICATIONS OF GRAPH TRANSFORMATIONS WITH INDUSTRIAL RELEVANCE, AGTIVE, 2., 2003, Charlottesville, VA. **Revised selected and invited papers.** Berlin: Springer, 2004. p.398-412. (Lecture Notes in Computer Science, v.3062)

MINAS. M. Bootstrapping visual components of the DiaGen specification tool with DiaGen. In: INTERNATIONAL WORKSHOP ON APPLICATIONS OF GRAPH TRANSFORMATIONS WITH INDUSTRIAL RELEVANCE, AGTIVE, 2003, Charlottesville, VA. **Proceedings…** [S.l.: s.n.], 2003. p.391-405.

MURATA, M. et al. Taxonomy of XML schema languages using formal language theory. **ACM Transactions on Internet Technology**, New York, v.5, n.4, p.660-704, 2005.

MURATA, M. (Ed.). **ISO/IEC JTC1 SC34**: Namespace-based Validation Dispatching Language. [S.l.]: ISO/IEC, 2004.

NAJORK, M. A.; KAPLAN, S. M. Specifying visual languages with conditional set rewrite systems. In: IEEE WORKSHOP ON VISUAL LANGUAGES, 1993, New York. **Proceedings…**[S.l.]: IEEE Computer Society, 1993. p.12–18.

NICOLA, M.; JOHN, J. XML parsing: a threat to database performance. INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 2003, New Orleans, LA, USA. **Proceedings...**New York: ACM Press, 2003. p.175-178.

OMG. **OMG Unified Modeling Language Specification (OMG UML), Superstructure,** V2.1.2. [S.l.], 2007.

PIXLEY, T. et al. **Document Object Model (DOM) Level 2 Events Specification**. World Wide Web Consortium recommendation. 2000. Disponível em: < http://www.w3.org/TR/DOM-Level-2-Events/>. Acesso em: maio 2008.

QUINT, V.; VATTON, I. Editing with Style. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING, DocEng, 2007. **Proceedings…**[S.l.]: ACM Press, 2007. p.151-160.

QUINT, V.; VATTON, I. Techniques for Authoring Complex XML Documents. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING, DocEng, 2004. **Proceedings…**[S.l.]: ACM Press, 2004. p. 115-123.

QUINT, V. **The Languages of Thot.** Grenoble: Imag, 2000. Report, Opera project, .2000. Translated by E. Munson. Disponível em: < http://www.w3.org/Amaya/User/languages.toc.html>. Acesso em maio 2008.

REPENNING, A. Bending the Rules: Steps toward Semantically Enriched Graphical Rewrite Rules, In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1995, Darmstadt, Germany. **Proceedings…**[S.l.]: IEEE Computer Society, 1995. p. 226-233.

SMITH, D. C. Building personal tools by programming. **Communications of the ACM,** New York, v.43, n.8, p. 92-95, 2000.

STAGECAST INC**. Stagecast Creator's Guide.** EUA, 2007. Relatório técnico. Disponível em: < http://www.stagecast.com/>. Acesso em: abril 2008.

TAENTZER, G. et al. **Tiger project, user manual**. Berlin: Institut für Softwaretechnik und Theoretische Informatik Technische Universität Berlin, 2006. Relatório Técnico.

TELECKEN, T.L.; MUNSON, E.V.; LIMA J.V. Applying markup language resources in the specification of visual alphabets and visual sentences. In: ACM SYMPOSIUM ON APPLIED COMPUTING, SAC, 2008, Fortaleza, Brazil. **Proceedings…**New York, NY,USA: ACM Press, 2008. p.222-227.

TELECKEN, T. L.; LIMA, J. V. Increasing XML Interoperability in Visual Rewriting Systems. In: SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB, WebMedia, 2005. **Proceedings…**New York, NY,USA: ACM Press, 2005a. p. 1-8.

TELECKEN, T. L.; LIMA, J. V. Generic Editing of Visual Languages based on SVG standard. In: IFIP ACADEMY ON THE STATE OF SOFTWARE THEORY AND PRACTICE - PHD COLLOQUIUM, 2005, Porto Alegre, Brasil. **Proceedings…** [S.l.: s.n.], 2005b.

TELECKEN, T. L.; LIMA, J. V.; FRANÇA, M. B. Process modeling architectures with namespace and XML technology. In: CENTRO LATINOAMERICANO DE ESTUDIOS EN INFORMÁTICA, CLEI, 2004, Arequipa, Peru. **Proceedings…** [S.l.: s.n.], 2004a. p.54-65.

TELECKEN, T. L.; STEINMACHER, I.; LIMA, J. V. Amaya Workflow: Uma ferramentas de definição de processos. In: BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB - DEMOS AND TOOLS, WebMedia, 2004, Ribeirão Preto, SP. **Proceedings…** [S.l.: s.n.], 2004b. p.295-296.

TELECKEN, T. L.; LIMA, J. V. Métodos de descrição e avaliação de ferramentas de definição de processos de workflow. In: BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB - WCSCW, WebMedia, 2004, Ribeirão Preto, SP. **Proceedings…** [S.l.: s.n.], 2004c. p.149-146.

VLIST, E. **Relax NG**. Sebastopol, CA: O'Reilly, 2003.

ZHU, N. et al. Pounamu: A meta-tool for exploratory domain-specific visual language tool development. **Journal of Systems and Software**, New York, v.80, n.8, p.1390-1407, 2007.

ANEXO A ESPECIFICAÇÃO FORMAL DOS SISTEMAS VCARW E EVCARW

Neste anexo será descrito o modelo VCARW (Visual Conditional Attributed Rewriting System) proposto por Bottoni et al. (2007). Primeiramente serão mostrados os modelos de sentença visual, alfabeto e gramática do VCARW. Em seguida serão feitas algumas considerações sobre a expressividade destes modelos.

Os modelos de sentença visual, alfabeto e gramática a seguir foram implementados no ADBG Genial (BOTTONI, 2004).

O modelo de sentença visual VCARW - Sintaxe

Uma **sentença visual** é formada por um conjunto de *símbolos*, cada *símbolo u* é definido pela tupla $(t, u_1,...,u_j)$, onde *t* é o *nome* do *símbolo* e cada termo u_n é o *valor* de um *atributo* de *u*. A figura A.1 mostra o diagrama de classe do modelo de sentença visual VCARW.

Quando uma sentença visual está de acordo com um alfabeto visual Σ, ela é formada por um conjunto de símbolos $u = (n_i, u_1,...,u_j)$, onde n_i é o *nome de símbolo* de um *tipo de símbolo* t_i, tal que $t_i \in \Sigma$. Cada termo u_j referencia um atributo a_j e tem um valor do domínio δ_{aj} sendo que $a_j \in A_i$.

Quando uma sentença visual está de acordo com uma gramática, a sentença obedece as restrições do alfabeto utilizado pela gramática e as restrições da própria gramática.

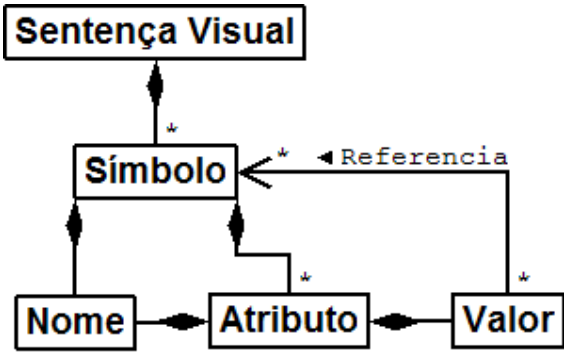


Figura A.1: Modelo de sentença visual VCARW

O modelo de sentença visual VCARW - Semântica

A semântica de uma sentença visual define como um sistema computacional deve interpretar esta sentença visual.

Seja $\Gamma = \{\gamma_1,...,\gamma_r\}$ um conjunto finito de *geradores de classes geométricas*, cada um denotado por um *tipo de símbolo* em Σ. Para cada classe $\gamma_i \in \Gamma$ é assumida a existência de algum procedimento recursivo ρ_i que decide quais arranjos de sinais do espaço bi-dimensional farão parte de γ_i. (no caso de uma tela de computador, ρ_i define os pixels da tela que farão parte de γ_i e as características destes pixeis) ρ_i também dispara procedimentos adicionais relacionados com a interpretação de t_i.

Assim, em uma sentença visual, cada *símbolo u* do tipo t_i é interpretado como um arranjo de sinais e procedimentos especificados em $\rho_i(u)$. Portanto a semântica de uma sentença visual é a interpretação de todos os seus *símbolos*.

O modelo de alfabeto visual VCARW

Um alfabeto limita o conjunto de sentenças visuais válidas de um sistema de linguagem visual. Mais precisamente, o alfabeto especifica que somente símbolos definidos no alfabeto podem estar presentes nas sentenças visuais de um sistema de linguagem visual.

Formalmente, um alfabeto visual VCARW $\Sigma = \{t_1,...,t_r\}$ é um conjunto finito de *tipos de símbolos* onde cada *tipo de símbolo* t_i é composto de um *nome de símbolo* e um conjunto $A_i = \{a_1,...,a_n\}$ de *atributos*. Cada *atributo* $a_j \in A_i$ possui um *nome de atributo* e um domínio de valores δ_{ai}. δ_{ai} é um bem-definido (com respeito a um conjunto de operações) domínio algébrico. Eventualmente δ_{ai} pode permitir que o valor do atributo a_i referencie símbolos da atual sentença visual. Tais referências são importantes para se estabelecer ligações semânticas entre símbolos de uma sentença visual.

O modelo de gramática VCARW

A gramática define como os símbolos podem ser combinados para formar sentenças visuais válidas, de forma análoga as gramáticas de linguagens textuais que definem como letras de um alfabeto podem ser combinadas para formarem palavras válidas.

Adicionalmente a gramática VCARW especifica como uma sentença válida vs_1 pode ser transformada em outra sentença válida vs_2 e quais eventos podem disparar estas transformações. É importante lembrar que o modelo VCARW define um sistema de linguagem visual como uma sucessão de sentenças visuais disparadas/controlads por eventos. Portanto a gramática é capaz de especificar um sistema de linguagem visual (como um editor gráfico, uma simulação gráfica ou uma animação) precisamente e sem ambigüidades.

Formalmente, uma gramática *Visual Conditional Attributed ReWriting system* (VCARW) é uma tupla $< \Sigma, P, \Rightarrow c >$, onde Σ é um alfabeto visual, *P* é um conjunto de regras de reescrita, que são pares de sentenças visuais (*antecedente, consequente*) sobre Σ, guardada por uma *condição* sobre o *antecedente* que precisa ser satisfeita para a regra ser aplicável (uma regra é escrita na forma $r=(ant, cons, cond))$. $e \Rightarrow c$ é a relação de reescrita que especifica como as regras em *P* são aplicadas nas sentenças visuais. É denotado que vs_1 diretamente gera vs_2 $(vs_1 \Rightarrow vs_2)$ se e apenas se $\exists r=(ant, cons, cond) \in P$ tal que é verdadeiro o seguinte:

1. vs_1 contém uma instância *A* de *ant*.
2. A condição *cond* é verdadeira em *A*.

3. vs_2 contém uma instância *C* de *cons* tal que os valores de seus atributos são iguais aqueles computados de *A* de acordo com as especificações em *cons*.
4. Todos outros símbolos de vs_1 são encontrados sem alterações em vs_2.
5. Não ha outros símbolos em vs_2.

É dito que vs_0 *gera* vs_n ($vs_0 \Rightarrow^* vs_n$) se uma sequencia de sentenças visuais $vs_0, vs_1,...,vs_{n-1}, vs_n$ existe tal que $vs_i \Rightarrow vs_{i+1}$ através de alguma regra em *P* para $i=0,...,n-1$. Dada a VCARW = $< \Sigma, P, \Rightarrow c >$ e um conjunto de sentenças visuais *Ax* chamados de axiomas, a linguagem gerada é o conjunto de sentenças visuais $L(vCARW, Ax)=\{vs \mid \exists vs_0 \in Ax,\ vs_0 \Rightarrow c^* vs\}$.

Uma variação do VCARW é o *enabling visual Conditional Attributed ReWriting system* (evCARW). Esta gramática extende o vCARW permitindo a definição explícita das ações que podem disparar a aplicação de uma regra vCARW em uma sentença de uma linguagem visual.

Uma gramática evCARW é uma tupla $D=< \Sigma, P, Ac, \epsilon, \emptyset, \mu, \Rightarrow e>$, onde Σ é um alfabeto. *P* é um conjunto de regras de reescrita, *Ac* é o conjunto de ações que podem disparar uma transformação nas sentenças visuais. ϵ e Ø são dois mapeamentos $\epsilon, \emptyset: P \Rightarrow \rho(Ac)$, chamados de mapeamentos *habilita* e *desabilita* respectivamente. μ é o mapeamento $\mu:Ac \Rightarrow \rho(P)$. A relação de reescrita $\Rightarrow e \subset (VL \times \rho(Ac))^2$ é definida no par (sentença visual,conjunto de ações habilitadas) como a seguir: $(vs,en) \Rightarrow e\ (vs', en')$ se e somente se $\exists p \in P$ e $\exists ac \in en \subset Ac$, tal que $p \in \mu(ac), vs \Rightarrow c\ vs'$, onde $en'=((en - \emptyset(p) - ac) \cup \epsilon(p)$.

Em palavras, toda regra pode habilitar ou proibir a subsequente aplicação de regras de algum conjunto. O fato de uma regra estar habilitada não implica que ela será aplicada nem que ela possa ser aplicada em qualquer situação. Em particular, é possível que nenhum conjunto de símbolos da atual sentença visual case com os padrões do *antecedente* de uma regra habilitada. Entretanto, o fato de uma regra estar desabilitada implica que ela não poderá ser aplicada, mesmo que uma instância de seu *antecedente* possa ser encontrada na atual sentença visual.

A linguagem gerada por um evCARW D começando de um conjunto de axiomas Ax com um conjunto inicial de ações habilitadas A é: $L(D,Ax,A)=\{vs \mid \exists ax \in Ax, \exists en \in \rho(Ac), (ax,A) \Rightarrow e^* (vs,en)\}$. Note que neste caso é implicitamente indicado o conjunto de ações (habilitadas) que poderão causar a transição de uma sentença para outra na linguagem.

Estas especificações foram retiradas de (BOTTONI, 2007). Também é possível deduzir os comandos de um sistema de linguagem visual através de uma gramática VCARW. Entretanto, tais comandos são pouco flexíveis: todas as gramáticas devem ter comandos semelhantes definidos por convenção (BOTTONI, 1999).

Expressividade

Bottoni et al. em (1999) demonstrou que uma gramática vCARW (gramática + alfabeto) é equivalente a gramática Conditional Set Rewrite System (CSRS) proposta por Najork e Kaplan (1993). Najork (1993) demosntrou que a gramática CSRS é uma generalização de gramáticas livres do contexto, Picture Layout Grammar (GOLIN, 1991) e Multiset Grammar (HELM, 1991). Portanto Bottoni (1999) declara que a

gramática VCARW é equivalente a uma gramática CSRS e pelo menos tão expressiva quanto uma gramática livre do contexto, uma *Constraint Multiset Grammar* e uma *Picture Layout Grammar*.

As instâncias do modelo de sentença visual, ou simplesmente as sentenças visuais, do VCARW são capazes de representar todas as sentenças visuais definidas por uma gramática VCARW. Logo o modelo de sentenças visuais do VCARW pode representar qualquer sentença visual definida por uma gramática livre do contexto, uma *Constraint Multiset Grammar* e uma *Picture Layout Grammar*.

ANEXO B GXL

O GXL (HOLT, 2006) é um formato padrão para a exportação de dados baseados em grafos. Este é um dos formatos de descrição de sentenças visuais mais utilizados em ADBGs. Os ADBGs AGG, GENGED (seu descendente Tiger), DiaGEN, Diaplan, PROGRES, VLCC e VLDesk podem exportar e/ou importar sentenças visuais e alfabetos para o formato GXL.

Através do GXL sentenças visuais são descritas como grafos onde os símbolos são descritos como nodos e os relacionamentos entre os símbolos são descritos como arestas. Os grafos que descrevem as sentenças visuais podem ter diferentes características, como por exemplo: *grafos hierárquicos*, onde um nodo pode conter um sub-grafo; *hipergrafos*, onde uma aresta ou conjunto de arestas podem descrever um relacionamento entre múltiplos nodos (no lugar de apenas relações binárias); etc. O modelo lógico do GXL é mostrado na figura B.1.

O diagrama de classes da figura B.1 especifica todos os recursos do GXL. Ele está dividido em duas partes. A primeira descreve como os grafos são descritos: um *graph* contém *GraphElements*, que são *Nodes*, *Relations* e *Edges*. Para suportar grafos hierárquicos, cada *GraphElement* pode conter outros *Graphs*. *Edges* armazenam relacionamentos binários e *Relations* armazenam relacionamentos múltiplos entre *GraphElements*. É importante notar que o GXL permite que arestas e hiper-arestas tenham conexões com outras arestas e hiper-arestas além de conexões com nodos. Tipos de *GraphElements* e *Graphs* podem ser definidos em documentos externos e referenciados em documentos GXL. A segunda parte do diagrama descreve os atributos e os tipos de dados associados.

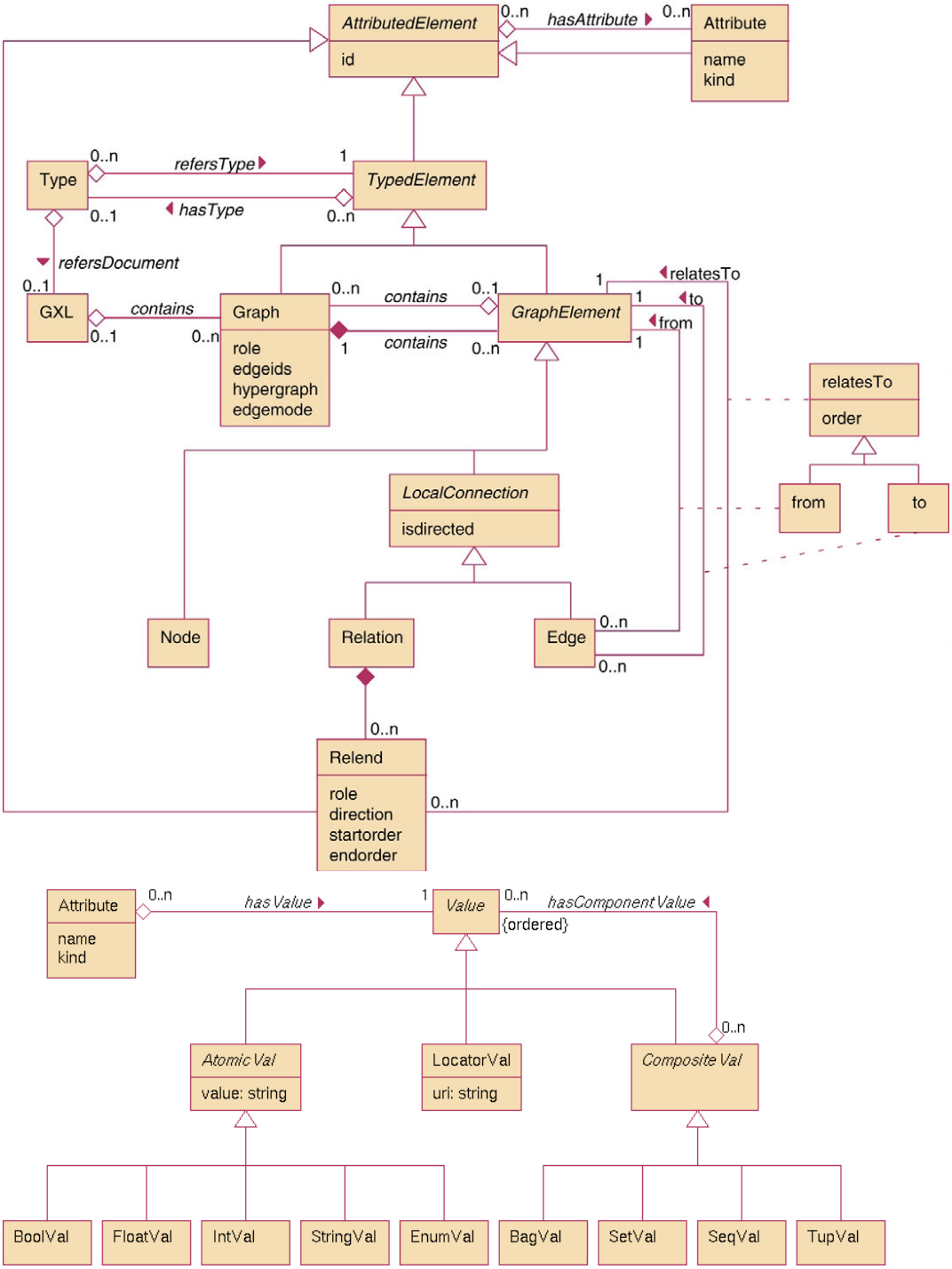


Figura B1: Modelo de sentença visual do GXL (HOLT,2006)

O modelo GXL é usado por muitos sistemas como um modelo de sentença visual. No espaço tecnológico XML, o modelo GXL é realizado em esquemas XML e suas sentenças visuais em documentos XML. A figura B.2 mostra um esquema XML do GXL. Este esquema é uma DTD. Os documentos GXL são documentos XML que estão de acordo com o referido esquema.

```
<!– extensions –>
<!ENTITY % gxl-extension "" >
<!ENTITY % graph-extension "" >
<!ENTITY % node-extension "" >
<!ENTITY % edge-extension "" >
<!ENTITY % rel-extension "" >
<!ENTITY % value-extension "" >
<!ENTITY % relend-extension "" >
<!ENTITY % gxl-attr-extension "" >
<!ENTITY % graph-attr-extension "" >
<!ENTITY % node-attr-extension "" >
<!ENTITY % edge-attr-extension "" >
<!ENTITY % rel-attr-extension "" >
<!ENTITY % relend-attr-extension "" >
<!– attribute values –>
<!ENTITY % val " locator | bool | int | float | string
| enum | seq | set | bag | tup
% value-extension;" >
<!– gxl –>
<!ELEMENT gxl (graph* %gxl-extension;) >
<!ATTLIST gxl
xmlns:xlink CDATA #FIXED
"www.w3.org/1999/xlink"
%gxl-attr-extension; >
<!– type –>
<!ELEMENT type EMPTY>
<!ATTLIST type
xlink:type (simple) #FIXED "simple"
xlink:href CDATA #REQUIRED >
<!– graph –>
<!ELEMENT graph (type? , attr* ,
( node | edge | rel )*
%graph-extension;) >
<!ATTLIST graph
id ID #REQUIRED
role NMTOKEN #IMPLIED
edgeids ( true | false ) "false"
hypergraph ( true | false ) "false"
edgemode ( directed | undirected |
defaultdirected | defaultundirected)
"directed"
%graph-attr-extension; >
<!– node –>
<!ELEMENT node (type? , attr*, graph*
%node-extension;) >
<!ATTLIST node
id ID #REQUIRED
%node-attr-extension; >
<!– edge –>
<!ELEMENT edge (type?, attr*, graph*
%edge-extension;) >
<!ATTLIST edge
id ID #IMPLIED
from IDREF #REQUIRED
to IDREF #REQUIRED
fromorder CDATA #IMPLIED
toorder CDATA #IMPLIED
isdirected ( true | false ) #IMPLIED
%edge-attr-extension; >
<!– rel –>
<!ELEMENT rel (type? , attr*, graph*, relend*
%rel-extension;) >
<!ATTLIST rel
id ID #IMPLIED
isdirected ( true | false ) #IMPLIED
%rel-attr-extension; >
<!– relend –>
<!ELEMENT relend (attr* %relend-extension;) >
<!ATTLIST relend
target IDREF #REQUIRED
role NMTOKEN #IMPLIED
direction ( in | out | none) #IMPLIED
startorder CDATA #IMPLIED
endorder CDATA #IMPLIED
%relend-attr-extension; >
<!– attr –>
<!ELEMENT attr (attr*, (%val;)) >
<!ATTLIST attr
id IDREF #IMPLIED
name NMTOKEN #REQUIRED
kind NMTOKEN #IMPLIED >
<!– locator –>
<!ELEMENT locator EMPTY >
<!ATTLIST locator
xlink:type (simple) #FIXED "simple"
xlink:href CDATA #IMPLIED >
<!– attribute values –>
<!ELEMENT bool (#PCDATA) >
<!ELEMENT int (#PCDATA) >
<!ELEMENT float (#PCDATA) >
<!ELEMENT string (#PCDATA) >
<!ELEMENT enum (#PCDATA) >
<!ELEMENT seq (%val;)* >
<!ELEMENT set (%val;)* >
<!ELEMENT bag (%val;)* >
<!ELEMENT tup (%val;)* >
```

Figura B.2: DTD GXL (HOLT,2006)

Alguns ADBGs como, por exemplo, o VLCC e o VLDesk, também utilizam o modelo GXL para armazenar alfabetos. Holt (2006) apresenta mais detalhes sobre o uso do GXL como formato de armazenamento de alfabetos.

Outros modelos de sentença visual baseados em grafos

O GXL é um modelo de sentença visual que descreve sentenças visuais como grafos. Ele suporta vários tipos de grafos como hipergrafos, grafos hierárquicos etc. Por suportar vários tipos de grafos o GXL vem despontando como um padrão para exportação e importação de sentenças visuais baseadas em grafos. Entretanto as

sentenças visuais deste modelo são muito complexas para serem editadas pelos editores dos ADBGs.

Muitos ADBGs utilizam grafos como a base de seus modelos de sentença visual. Entretanto os modelos apenas suportam um subconjunto dos tipos de grafos suportados pelo GXL.

As sentenças visuais do Atom3, Genged e Progress podem conter apenas *attributed graphs,* ou seja, grafos onde nodos e arestas podem ter atributos. Consequentemente o modelo de sentença visual destes ambientes é similar ao subconjunto das entidades e relacionamentos do modelo GXL que são capazes de descrever *attributed graphs*. Já os modelos de alfabetos descrevem documentos que podem especificar tipos em *attributed graphs.*

Em outro exemplo, as sentenças visuais do Diagen podem conter apenas *hipergraphs,* ou seja, grafos onde uma aresta ou conjunto de arestas podem descrever um relacionamento entre múltiplos nodos (no lugar de apenas relações binárias). Consequentemente o modelo de sentença visual do Diagen é similar ao subconjunto das entidades e relacionamentos do modelo GXL que são capazes de descrever *hipergraphs*. Os modelos de alfabetos, por sua vez, descrevem documentos que podem especificar tipos em *hipergraphs.*

ANEXO C PARTE DE UM ESQUEMA S

```
{ Thot structure schema for Vitransf Rules }
STRUCTURE SVG;
DEFPRES SVGP;
ATTR

  { global attributes for all SVG elements }
  {stdAttrs}
  id = text;
  xml_base = text;

  xml_space = xml_space_default,
xml_space_preserve;

  { attributes for internal processing }
  Unknown_attribute = text;
  Ghost_restruct = text;
  PseudoClass = text;
  Highlight = Yes_;       { to show the SVG
element corresponding to the
    current selection in the source view }
  Namespace = text;       { for children of
element foreignObject }
CONST
  C_Empty = ' ';

STRUCT
{ Document Structure }

 SVG
(ATTR requiredFeatures = text;     {testAttrs}
    requiredExtensions = text;    {testAttrs}
    systemLanguage = text;        {testAttrs}
    baseProfile = text;
    externalResourcesRequired = false, true;
    class = text;
    style_ = text;
{ PresentationAttributes-All }
    color = text;
{PresentationAttributes-Color}
    color_interpolation = auto, sRGB,
linearRGB, inherit;
{PresentationAttributes-Color}
    color_rendering = auto, optimizeSpeed,
optimizeQuality, inherit;
 {PresentationAttributes-Color}
    enable_background = text;
{PresentationAttributes-Containers}
    flood_color = text;
{PresentationAttributes-feFlood}
    flood_opacity = text;
{PresentationAttributes-feFlood}
    fill = text;
{PresentationAttributes-FillStroke}
    fill_opacity =  text;
{PresentationAttributes-FillStroke}
    fill_rule = nonzero, evenodd, inherit;
 {PresentationAttributes-FillStroke}
    stroke = text;
{PresentationAttributes-FillStroke}
    stroke_dasharray = text;
{PresentationAttributes-FillStroke}
    stroke_dashoffset = text;
{PresentationAttributes-FillStroke}
    stroke_linecap = butt, round, square, inherit;
 {PresentationAttributes-FillStroke}
    stroke_linejoin = miter, round, bevel,
inherit;
  {PresentationAttributes-FillStroke}
    stroke_miterlimit = text;
{PresentationAttributes-FillStroke}
    stroke_opacity = text;
{PresentationAttributes-FillStroke}
    stroke_width = text;
{PresentationAttributes-FillStroke}
    color_interpolation_filters = auto, sRGB,
linearRGB, inherit;
 {PresentationAttributes-FilterPrimitives}
    font_family = text;
{PresentationAttributes-FontSpecification}
    font_size = text;
{PresentationAttributes-FontSpecification}
    font_size_adjust=text;
{PresentationAttributes-FontSpecification}
    font_stretch = normal_, wider, narrower,
ultra_condensed, extra_condensed, condensed,
semi_condensed, semi_expanded, expanded,
extra_expanded, ultra_expanded, inherit;
 {PresentationAttributes-FontSpecification}
    font_style = normal_, italic, oblique_,
inherit;
{PresentationAttributes-FontSpecification}
```

font_variant = normal_, small_caps, inherit;
{PresentationAttributes-FontSpecification}
font_weight = normal_, bold_, bolder,
lighter, w100, w200, w300, w400, w500, w600,
w700, w800, w900, inherit;
{PresentationAttributes-FontSpecification}
stop_color = text;
{PresentationAttributes-Gradients}
stop_opacity = text;
{PresentationAttributes-Gradients}
clip_path = text;
{PresentationAttributes-Graphics}
clip_rule = nonzero, evenodd, inherit;
{PresentationAttributes-Graphics}
cursor_ = text;
{PresentationAttributes-Graphics}
display = inline, block, list_item, run_in,
compact, marker_,table, inline_table,
table_row_group, table_header_group,
table_footer_group, table_row,
table_column_group, table_column, table_cell,
table_caption, none, inherit;
{PresentationAttributes-Graphics}
filter_ = text;
{PresentationAttributes-Graphics}
image_rendering = auto, optimizeSpeed,
optimizeQuality, inherit;
{PresentationAttributes-Graphics}
mask_ = text;
{PresentationAttributes-Graphics}
opacity_ = text;
{PresentationAttributes-Graphics}
pointer_events = visiblePainted, visibleFill,
visibleStroke, visible, painted, fill__, stroke_,
all, none, inherit;
{PresentationAttributes-Graphics}
shape_rendering = auto, optimizeSpeed,
crispEdges, geometricPrecision, inherit;
{PresentationAttributes-Graphics}
text_rendering = auto, optimizeSpeed,
optimizeLegibility, geometricPrecision, inherit;
{PresentationAttributes-Graphics}
visibility_ = visible, hidden_, inherit;
{PresentationAttributes-Graphics}
color_profile_ = text;
{PresentationAttributes-Images}
lighting_color = text;
{PresentationAttributes-LightingEffects}
marker_start = text;
{PresentationAttributes-Markers}
marker_mid = text;
{PresentationAttributes-Markers}
marker_end = text;
{PresentationAttributes-Markers}
alignment_baseline = baseline, top,
before_edge, text_top, text_before_edge,
middle, bottom, after_edge, text_bottom,
text_after_edge, ideographic_, lower, hanging_,
mathematical_, inherit;
{PresentationAttributes-TextContentElement}
baseline_shift = text;
{PresentationAttributes-TextContentElements}
direction_ = ltr_, rtl_, inherit;
{PresentationAttributes-TextContentElement}
dominant_baseline = auto, autosense_script,
no_change, reset, ideographic_, lower,
hanging_, mathematical_, inherit;
{PresentationAttributes-TextContentElement}
glyph_orientation_horizontal = text;
{PresentationAttributes-TextContentElement}
glyph_orientation_vertical = text;
{PresentationAttributes-TextContentElement}
kerning = text;
{PresentationAttributes-TextContentElements}
letter_spacing = text;
{PresentationAttributes-TextContentElements}
text_anchor = start, middle, end__, inherit;

…