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

RICARDO DE MOURA RIVALDO

GraphSchema – Uma Linguagem Visual Para a Criação de Modelos de Contratos com SML

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. José Palazzo Moreira de Oliveira
Orientador

Porto Alegre, 29 de maio de 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Rivaldo, Ricardo de Moura

GraphSchema – Uma Linguagem Visual Para a Criação de Modelos de Contratos com SML / Ricardo de Moura Rivaldo – Porto Alegre: Programa de Pós-Graduação em Computação, 2007.

82 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2007. Orientador: Prof. Dr. José Palazzo Moreira de Oliveira.

1. Modelagem de Documentos de Texto. 2. SML 3. Linguagem Gráfica para XML. I. Oliveira, José Palazzo Moreira. II. Título.

"*Se tivésseis a fé do tamanho de um grão de mostarda, diríeis a esta montanha: Transporta-te daí para ali e ela se transportaria, e nada vos seria impossível*".
(S. MATEUS, cap. XVII, vv. 14 a 20.)

SUMÁRIO

LISTA DE TABELAS 7
LISTA DE FIGURAS 8
RESUMO 9
ABSTRACT 10
1 INTRODUÇÃO 11
2 ESTRUTURA DE DADOS E METADADOS 12
2.1 Dados Não Estruturados 12
2.2 Metadados 14
2.3 Memória Organizacional e Integração de Informações 15
3 DOCUMENTOS DE TEXTO EM LINGUAGEM NATURAL 16
3.1 Enriquecimento de Documento (Document Enrichment) 16
3.2 Indexação Conceitual 17
3.3 Ferramentas Para o Enriquecimento de Documentos de Texto 17
3.3.1 Editor de XML 18
3.3.2 Editor de Texto Guiado Por XML 18
3.4 A Pobreza Semântica do XML 19
4 SERVICE MODELING LANGUAGE - SML 21
4.1 Descrição da Linguagem SML 22
4.1.1 Referências 23
4.1.2 Função deref() 23
4.1.2.1 Múltiplos esquemas de endereçamento 23
4.1.2.2 Restrições Para Validar o Elemento Referenciado 24
4.1.2.3 Restrições de Identidade 24
4.1.3 Regras 25
4.2 Ontologias 25
4.2.1 Ontologias Fracas e Fortes 26
4.2.2 Componentes de Uma Ontologia 26
4.3 SML Como uma Linguagem de Criação de Ontologias 27
4.4 SML Como Suporte Para o Enriquecimento de Documentos 28
5 MODELAGEM DE DOCUMENTOS DE TEXTO LEGAIS 30
5.1 Contexto 30
5.2 Contratos e a Origem do GraphSchema 31
5.3 Estruturação de documentos Legais com XML 33

5.4 Macro-Nível, Micro-Nível 34
5.5 Técnicas para o Esboço Automatizado de Documentos Jurídicos 35
5.6 Esboço Automatizado de Documentos 35
5.6.1 Tipos de Sistemas de Esboço Automatizado de Documentos Legais 36
5.6.1.1 Sistemas Informativos e Funções de Ajuda 36
5.6.1.2 Sistemas de Montagem e Geração de Texto 37
5.6.1.3 Sistemas de Verificação 37
5.6.1.4 Sistemas de Tradução 37
5.6.2 Classificação do GraphSchema 38
5.7 Padrões existentes 38
5.7.1 LegalXML eContracts 38
5.7.2 NIREditor 40
5.7.3 MetaLex 41
5.7.4 Contract Expression Language (CEL) 42
5.7.5 Rule-based XML 43

6 REPRESENTAÇÕES GRÁFICAS E XML 44

6.1 Linguagens Gráficas Para Tratamento de Dados Não Estruturados 46
6.1.1 Linguagens Baseadas no Paradigma de Grafo 46
6.1.1.1 G-Log 46
6.1.1.2 XML-GL 47
6.1.1.3 VXT 47
6.1.2 Linguagens Baseadas No Paradigma de Documento 47
6.1.2.1 Xing 47
6.1.2.2 FoXQ 50
6.1.2.3 CCSL e VCCSL 50

7 GRAPHSCHEMA 52

7.1 Introdução 52
7.2 Definição 52
7.2.1 Overview 53
7.2.2 Elementos 55
7.2.2.1 Model Library 56
7.2.2.2 Document Model 56
7.2.2.3 Item Model 57
7.2.2.4 Item Model Atomic 57
7.2.2.5 Item Model Composite 57
7.2.2.6 Default Text 58
7.2.2.7 Rules 58
7.2.2.7.1 Restrições Unitárias 59
7.2.2.7.2 Restrições de Relacionamento 59
7.3 Exemplo de Aplicação 60

8 IMPLEMENTAÇÃO 65

8.1 Projeto 65
8.1.1 Ferramentas Utilizadas 65
8.1.1.1 Biblioteca Para Interface Gráfica 67
8.1.1.2 Biblioteca para Manipulação de Árvore de Objetos 70
8.2 Modelo de Objetos 71

9 CONCLUSÃO 73

9.1 Principais Contribuições 74
9.1.1 Principais Dificuldades Encontradas na Implementação do Software 75

9.1.1.1 Identificação de elementos 75
9.1.1.2 Implementação 75
9.2 Trabalhos futuros 75
REFERÊNCIAS 77

LISTA DE TABELAS

Tabela 2.1: Diferenças entre dados tradicionais e semi-estruturados (HEUSER et al.). 14
Tabela 4.1: SML, Restrições Para Validar o Elemento Referenciado..... 24
Tabela 4.2: SML, Restrições de Identidade..... 24
Tabela 4.3: Exemplo de Regra em Schematron 25
Tabela 5.1: Técnicas Para o o Esboço Automatizado de Documentos Legais..... 35
Tabela 7.1: Restrições Unitárias..... 59
Tabela 7.2: Restrições de Relacionamento..... 60

LISTA DE FIGURAS

Figura 4.1: Espectro Ontológico (LASSILA, 2001)..... 26
Figura 5.1: Processo de Criação de um Documento usando Esquema do Documento.. 33
Figura 5.2:NIREditor..... 41
Figura 5.3:Rule Based XML GUI 43
Figura 6.1: Dados XML Representados como Grafo 44
Figura 6.2:Tela de Um Editor de XML Orientado a Dados 45
Figura 6.3: Dados XML Representados em VXT 47
Figura 6.4:XML Para Exemplos do Xing..... 48
Figura 6.5:Representação Gráfica de Dados em Xing 48
Figura 6.6: Consulta a livros cujo autor é “Knuth” 49
Figura 6.7: Resultado da Consulta a Livros 49
Figura 6.8: Exemplo Gráfico em VCCSL 51
Figura 6.9: XSL equivalente ao exemplo gráfico..... 51
Figura 7.1: Representação de um item em GraphSchema..... 53
Figura 7.2: Variações Possíveis nos Itens 54
Figura 7.3: Construtores GraphSchema..... 56
Figura 7.4: GraphSchema Para Um Contrato de Compra e Venda de Imóveis 61
Figura 7.5: Código SML coorespondende ao Contrato de Compra/Venda de Imóveis. 63
Figura 7.6: Texto de Contrato de Compra/Venda de Imóvel 64
Figura 8.1: Modelo Completo das Classes Implementadas no GraphSchema..... 66

RESUMO

É usual falar da onipresença dos documentos de texto e na quantidade de informação não estruturada, armazenada sob a forma de arquivos com documentos de texto em linguagem natural. Este fato torna-se mais dramático no domínio jurídico, onde o texto é a ferramenta básica de trabalho dos profissionais da área, tanto na forma das fontes de consulta, i.e., a legislação, como no principal produto da atividade jurídica, especificamente a criação de documentos escritos.

Desde a invenção do editor de texto existem iniciativas de utilização de tecnologias da informação para auxiliar a geração, armazenamento e consulta de documentos jurídicos. Dentre os diversos ramos da atividade jurídica, a criação de contratos é especialmente importante, devido a sua onipresença nas interações entre os agentes sociais, sejam elas pessoas físicas, jurídicas ou agentes de governo.

Com foco na criação de modelos de contratos, este trabalho introduz a linguagem gráfica GraphSchema. Projetada para usuários finais, GraphSchema utiliza uma representação visual para criação de modelos de contratos jurídicos, permitindo a modelagem dos conceitos, relacionamentos e restrições entre estes. A representação visual é diretamente mapeada na linguagem SML, uma extensão do XML Schema.

Ao possibilitar a criação de modelos conceituais de contratos diretamente por parte dos usuários finais sem forçar um vocabulário ou ontologia específicos, GraphSchema e, conseqüentemente, a utilização de SML, apresenta vantagens quando comparado com a utilização de XML Schema, RDF e OWL. Mas principalmente apresenta vantagens quando comparada com outras abordagens baseadas em definição de vocabulários e utilização de ontologias formais. Estas vantagens decorrem de sua simplicidade e flexibilidade que permite a utilização de padrões existentes para a definição de modelos de contratos, tais como, o padrão eContracts definido pelo consórcio LegalXML. Deste modo, GraphSchema apresenta-se como uma opção para a implementação e aplicação prática deste padrão.

A disponibilidade de uma linguagem para usuários não técnicos permitirá a criação de contratos com marcação *a priori*, quando utilizado em conjunto com editores de texto guiados por XML. Isto irá abrir caminho para o aumento da produtividade na criação de contratos e documentos jurídicos.

Palavras-Chave: SML, Modelagem de Documentos de Texto, contratos legais, eContracts, Ontologias, Linguagem Visual XML

GraphSchema – A Visual Language to Create Contract Models with SML

ABSTRACT

It is common place to talk about the widespread presence of text documents and unstructured information stored in natural language text documents file format. This fact is still more dramatic to law professionals where text is the basic tool for their work. Those texts came from multiple sources like research documents and legislation and also are the main product from law activities, i.e., text documents which are created by law professionals.

Since the first text editor there are several initiatives to use information technologies to help the generation, storage and search of law documents. From all documents, legal contracts generation is especially important due to its ubiquity and use by all social actors like common people, companies and government agencies.

This work main focus is legal contract model generation. GraphSchema graphical language is introduced as a proposed solution to enable users to create contract models without help from a computer professional. It uses a visual representation to create legal contracts models, where concepts, relationships between those and constraints can be represented in a visual paradigm which can be understood by users. The graphical representation is translated to SML, a XML Schema extension.

On enabling final user conceptual contract modeling without forcing a restrict vocabulary or ontology, GraphSchema and. by consequence, the use of SML, has several advantages in comparison with the use of simple XML Schema, RDF and OWL. But mainly show advantages when compared with other approaches based on vocabulary definition and formal ontology usage. Those advantages are mainly due to its simplicity and flexibility which enable the use of existing standards to define contract models like the eContracts standard defined by LegalXML consortium. This way, GraphSchema appears as an option to implement and use this standard in real world cases.

The availability of a language directed towards non technical user will enable the contracts creation with tag markup from the beginning when used with XML guided text editors. This opens a door to productivity grow on contracts and legal documents creation.

Keywords: SML, Text Document modeling, legal contracts, eContracts, Ontologies, XML Visual Language.

1 INTRODUÇÃO

Esta dissertação apresenta uma linguagem gráfica para a modelagem de contratos legais e a implementação do protótipo de uma ferramenta capaz de criar modelos usando esta linguagem. A linguagem gráfica é uma representa visual da linguagem SML.

São apresentados os conceitos envolvidos nesta pesquisa. Inicia-se pela definição de metadados. Passando então para a definição de documentos de texto em linguagem natural dentro do domínio deste estudo e como estes se encaixam na criação de memórias organizacionais através do conceito de enriquecimento de documentos. A seguir, são apresentadas ferramentas que unem documentos de texto e seus metadados.

Para a representação de metadados e modelos de documentos de texto, é feita uma análise do padrão de marcação de texto XML e como a linguagem SML pode ser usada para a representação de conhecimentos através de uma ontologia. Na seqüência, é feita uma análise da modelagem de documentos de textos legais e os problemas envolvidos.

Para uma comparação com trabalhos existentes, é feita uma caracterização do que é entendido por sistemas de esboço automatizado de documentos *(automatic document drafting)*, passando-se a apresentação de padrões e sistemas existentes.

GraphSchema permite que o usuário possa utilizar o sistema sem conhecimento da linguagem interna no qual o modelo é representado, para isto são avaliadas representações gráficas do XML e avaliadas como base para criação do GraphSchema.Também é descrito a implementação de um protótipo denominado de GraphSchema Editor.

Esta dissertação está dividida da seguinte forma: No capítulo 1 é introduzido o GraphSchema e conceitos gerais abordados na dissertação. No capítulo 2 são apresentados os conceitos de metadados e dados não estruturados. No capítulo 3 é definido o conceito de documento de texto em linguagem natural. Já no capítulo 4 é descrita a linguagem SML e como esta pode ser considerada uma linguagem de descrição de ontologias. No capítulo 5 é apresentada a Modelagem de textos legais, técnicas normalmente utilizadas para o esboço automatizado de documentos bem como os padrões existentes para a formatação de modelos de contratos. No capítulo 6 é feito uma avaliação das abordagens existentes para a representação gráfica e visual do XML, posteriormente usadas como base para a definição do GraphSchema. A partir do capítulo 7 é descrita a linguagem GraphSchema e a sua implementação é apresentada no capítulo 8. Finalmente, no capítulo 9 são apresentadas as conclusões desta dissertação , suas contribuições e trabalhos futuros.

2 ESTRUTURA DE DADOS E METADADOS

2.1Dados Não Estruturados

Fazendo uma síntese, pode-se dizer que o mundo do armazenamento digital de informação se divide em repositórios de informação estruturada e repositórios de informação não estruturada.

A informação estruturada engloba todos os sistemas de banco de dados, relacionais ou outros, mas também engloba todo tipo de armazenamento em arquivos que contenham estruturas definidas de informação (campos e registros).

Já a informação não estruturada engloba todos os arquivos que não possuem estrutura interna de representação de seus subcomponentes, tipicamente arquivos de texto ou multimídia (sons e imagens). Cabe observar que, mesmo em estruturas de armazenamento estruturado, podem-se ter informações não estruturadas, como é o caso de atributos *blob* em bancos de dados, onde são armazenados textos e informações multimídia.

Também é um conceito comum em discussões a respeito de formatos de informação que a maior parte da informação disponível encontra-se na forma de textos em linguagem natural (McCALLUM, 2005) – sejam documentos em formato de arquivos de texto comuns – associados a editores de texto - ou mesmo documentos disponíveis on-line em formatos HTML e semelhantes. Neste sentido, a internet é considerada o maior repositório de informação não estruturada existente, dada a sua natureza de documentos de texto em linguagem natural.

Atualmente, a maior atividade na definição de padrões da internet está justamente na criação de técnicas que permitam a estruturação semântica da informação contida nos textos em contraposição à estruturação do formato de apresentação atualmente difundido (HTML/CSS e outros), tanto na forma de classificação estrutural utilizando tecnologia XML, quanto na questão da classificação semântica propriamente dita, sendo este o caso do movimento em direção a web semântica apoiada nos conceitos de representação do conhecimento na forma de ontologias.

Muitas propostas têm surgido no sentido de estruturar a informação não estruturada, ou tentar tirar "ordem do caos" (NATALYA, 2005), pela obtenção ou associação de estrutura aos dados *a posteriori* ou *a priori* em relação ao momento em que os dados foram gerados. Estas técnicas, de um modo geral, tentam descobrir ou associar uma

estrutura semântica, ou metadados a esta informação que está representada em formato não estruturado.

As técnicas *a posteriori*, onde os metadados são criados a partir de textos existentes, baseiam-se em diversos métodos de mineração de dados (LAENDER et al.) apoiadas ou não por metadados ou modelo semântico externo (McCALLUM, 2005), tipicamente ontologias (DORNELES, 2000), (VANZIN), (BATRES, 2005).

Já as técnicas de tratamento *a priori* propõem a inclusão da estrutura semântica durante a criação do texto com a utilização de novos modelos que possam ser usados para estruturar e/ou representar dados não estruturados (BUNEMAN et al., 1997) e algumas chegam a considerar que XML é o modelo que faltava para estruturação destes dados (GOLDMAN et al., 1999). Estas técnicas, principalmente as que utilizam XML, também são conhecidas como técnicas de marcação ou anotação, devido à vinculação de estruturas subjacentes aos dados originais.

Dizer que dados são "não estruturados" leva a pensar que estes não possuem qualquer tipo de estrutura ou que esta não pode ser explicitada. No entanto, toda a informação, ou dado, possui uma estrutura. O que ocorre é que muitas vezes ela não pode ser formalizada usando técnicas de modelagem existente, ou a mesma está tão associada à própria informação que torna-se difícil separar uma estrutura semântica. Este é o caso de textos em linguagem natural, onde esta estrutura está implícita no próprio texto, principalmente no seu significado, i.e., semântica.

Alguns autores utilizam a palavra em inglês *Schema*, com o significado de uma representação ou modelo da estrutura, seja ela na dimensão do significado (semântica) ou do formato de representação e armazenamento (sintática) da informação em análise. Em português, pode-se usar a palavra estrutura ou esquema. A primeira refere-se a um aspecto de organização dos dados, já a segunda seria relativa a uma representação desta organização como, por exemplo, o esquema relacional que representa a organização da estrutura dos dados em um banco relacional.

Schema, em inglês, é geralmente usado porque alguns autores comparam a falta de estrutura dos dados não estruturados à organização dos dados em um modelo relacional de banco de dados (SPERBERG-MCQUEEN, 2005). A tendência nestes casos é culpar os dados, isto é, seria por alguma característica intrínseca dos dados a incapacidade de se ajustarem ao modelo relacional. O contraponto a esta visão é examinar o modelo relacional e perceber que ele não é realmente capaz de representar toda a gama de informações necessárias à atividade humana (SPERBERG-McQUEEN, 2005).

Então, de um modo mais amplo, dizer que alguns dados são não estruturados pode ser entendido como uma incapacidade dos modelos de dados existentes, especificamente, os modelos de bancos de dados relacionais, de representar de maneira adequada a informação, e então, neste caso, esta informação será denominada de não estruturada ou semi-estruturada.

Ignorando o problema do modelo relacional, pode-se dizer que dados semi-estruturados (ou não estruturados) referem-se a dados com uma ou mais das seguintes características (FLORESCU, 1998):

- A estrutura não é conhecida *a priori* e pode estar implícita nos dados;
- A estrutura é relativamente grande em comparação com o tamanho dos dados e pode ser freqüentemente alterada;
- A estrutura é descritiva, ou indicativa, ao invés de prescritiva, ou seja, a

estrutura descreve o estado atual dos dados, porém dados com desvios em relação à estrutura são aceitos como válidos.

- Os dados não são fortemente tipados (*strongly typed*), i.e., para objetos diferentes os valores de um mesmo atributo podem ser de tipos de dados diferentes;

Uma discussão mais detalhada sobre dados não estruturados pode ser encontrada em (HEUSER et al.), resumida na tabela a seguir com as diferenças entre o que se entende por dados tradicionais e os dados não estruturados:

Tabela 2.1: Diferenças entre dados tradicionais e semi-estruturados (HEUSER et al.)

Dados Tradicionais	Dados Semi-Estruturados
Esquema predefinido	Nem sempre há um esquema predefinido
Estrutura regular	Estrutura irregular
Estrutura independente dos dados	Estrutura embutida no dado
Estrutura reduzida	Estrutura extensa
Estrutura fracamente evolutiva	Estrutura fortemente evolutiva
Estrutura prescritiva	Estrutura descritiva
Distinção entre estrutura e dado é clara	Distinção entre estrutura e dado não é clara

Como o interesse desta dissertação é focado em documentos de texto em linguagem natural, utilizando a tabela de Heuser et al.,(2000) para classificar documentos de texto pode-se dizer que textos são dados não estruturados que possuem uma estrutura embutida nos dados, descritiva e irregular.

2.2 Metadados

Metadados são, literalmente, dados que descrevem dados. Nesta perspectiva, pode-se considerar que os metadados são informações que resumem, enriquecem ou complementam os objetos ou serviços referenciados, produzindo assim um potencial incremento de informação ou enriquecimento semântico dos dados (BORBINHA, 2005).

A capacidade dos metadados de prover informações que de outro modo não estaria disponível permite a interpretação dos dados além de seu próprio conteúdo, possibilitando a validação, decodificação e transformação de dados originais.

O *Getty Research Institute* define três dimensões para os metadados (SWETLAND):

- *Conteúdo:* relativo ao que o objeto contém ou a respeito do que ele trata. Isto é *intrínseco* a um objeto de informação;
- *Contexto:* indica os aspectos *quem*, *qual*, *porque*, *onde*, *quando* e *como* associados ao objeto de informação, sendo extrínseco a este objeto;
- *Estrutura:* relacionada ao conjunto formal de associações dentro dos objetos, ou entre eles, e pode tanto ser intrínseca como extrínseca.

É importante caracterizar a diferença entre o elemento de metadados (informação a respeito de outra informação) e a estrutura que irá guiar a coleta dos metadados necessários para determinada aplicação ou classe de documentos.

Quanto à associação dos dados aos seus metadados. É possível embutir os metadados no documento, como é o caso do XML e os *tags* <META>. Os metadados podem também estar separados dos dados, mas acessíveis através de um ponto comum, tipicamente o próprio documento que contém os dados irá conter indicadores apontando para os seus respectivos metadados. A terceira forma é o modelo da biblioteca tradicional, isto é, os dados estão fisicamente separados dos seus metadados e a ligação tem que ser feita pelo usuário interessado, por exemplo, a ficha descritiva de um livro.

2.3 Memória Organizacional e Integração de Informações

Historicamente, a memória das organizações é composta, além dos modelos de conhecimento contido no cérebro dos seus colaboradores, de diversos arquivos com documentos produzidos nos processos de trabalho desta organização. Estes documentos podem ser manuais de diversos tipos, cartas, memorandos, outros documentos de troca de informação com clientes e fornecedores e outros tipos de documentos. Esta memória organizacional, com o advento do computador, transferiu-se para discos e outros meios de armazenamento dos servidores e/ou dos computadores dos agentes que produzem estes documentos. A informação aqui referida é aquela dita não-estruturada, e a maior parte desta informação provavelmente está na forma de documentos de texto em linguagem natural.

A importância dos documentos de texto é reconhecida, tanto que a abordagem tradicional de gestão do conhecimento é tipicamente um processo focado no armazenamento e recuperação de documentos, o que termina levando a um desacoplamento entre os processos de criação, importação, captura, pesquisa e acesso ao conhecimento, e também, no limite, resulta na perda do conhecimento tácito do profissional que gerou o documento (STAAB et al., 2001).

3 DOCUMENTOS DE TEXTO EM LINGUAGEM NATURAL

Grande parte da informação e troca de informação é realizada na forma de documentos de texto, que provavelmente devem ser a maioria das informações disponíveis em qualquer esfera de atuação humana. Sendo assim, a transformação de texto em conhecimento é parte importante da gestão do conhecimento (O'LEARY, 1998).

Uma das práticas adotadas para a transformação de texto em conhecimento é associar metadados aos documentos na forma de ontologias ou outras descrições de alto nível.

A seguir serão descritas duas abordagens: *Document Enrichment*, ou enriquecimento de documentos (MOTTA et al., 2000) e *Indexation Conceptuelle*, ou Indexação Conceitual (PRIÉ, 2000).

3.1 Enriquecimento de Documento (Document Enrichment)

O *document enrichment* é definido como a agregação de metaconhecimento a documentos de texto de modo a aumentar o valor deste documento do ponto de vista da gestão do conhecimento, ou seja, aumentar as possibilidades e modos como este documento pode ser usado como fonte de conhecimento em atividades intensivas em conhecimento.

Uma abordagem é a utilização de uma ontologia associada ao documento, conhecida como "guiada por ontologias" (*ontology driven*). É importante fazer uma distinção de que o conhecimento formalizado nestas ontologias não se destina a ser uma tradução do que é informalmente especificado no documento associado, isto é, o significado do texto propriamente dito. A base do enriquecimento de documento é identificar os metadados de contexto não contido no texto, possibilitando assim a pesquisa e tratamentos semânticos do texto além do significado expresso no mesmo.

Uma representação, seja na forma gráfica seja textual, pode ser enriquecida de diversas maneiras:

- Disponibilizando informação sobre o contexto no qual foi criada;
- Disponibilizando informação sobre o contexto no qual pode ser utilizada;
- Fazendo referências (links) às outras representações e informações de mesma natureza e relevantes para o domínio ou contexto de uso;
- Fazendo referências (links) às outras representações e informações de natureza diferente, mas relevantes para o domínio ou contexto de uso;

- Relacionando conhecimento formal sobre o domínio ao texto.

Na suas pesquisa, Motta contrapõe a abordagem de criação de ontologia utilizada no seu trabalho, definida como *top-down*, isto é, a partir do texto é preenchida uma ontologia previamente definida. Em comparação com a abordagem tradicional, *bottom-up,* que é feita com a marcação do texto ou criação de ontologias capazes de guiar a criação do documento.

Neste aspecto, Van der Vet (1998) define a abordagem *bottom-up* de maneira diferente, porém, não exclusiva. Nesta definição, a ontologia é construída a partir dos componentes básicos que compõem os conceitos do domínio. Estes componentes são modelados de forma a também serem conceitos compostos. Esta abordagem é semelhante ao modo como são construídas ontologias baseadas em *Description Logic* (BAADER et al., 2004).

3.2 Indexação Conceitual

Um conceito similar ao enriquecimento de documento é chamado de indexação conceitual por Yannick Prié (PRIÉ, 2000). A indexação conceitual é definida como o conjunto de conhecimento simbólico que permite a exploração dos documentos. Exploração é usada no sentido de que a intenção do autor pode não estar completamente explicitada na estrutura do documento, já que o texto, por mais claro que seja, pode dar margem a interpretações diferentes da pretendida. Além disto, um documento é criado dentro de um contexto e pode ser interpretado em outro contexto. A indexação conceitual tenta instrumentalizar a indexação de modo a permitir não somente a recuperação dos documentos, mas também tornar-se fonte de conhecimento para processos de inferência automática e busca de conhecimentos embutidos nos documentos, em contraste com a busca por texto simplesmente.

Ou seja, trata-se da explicitalização das estruturas e dos conceitos contidos nos documentos, ou que a ele podem/são associados, permitindo uma melhor exploração tanto por homem como por máquina. Diferentemente da abordagem de enriquecimento de documento, a indexação conceitual não utiliza uma ontologia. Esta técnica faz anotação do texto utilizando marcação XML interna ou externa ao documento.

3.3 Ferramentas Para o Enriquecimento de Documentos de Texto

Os documentos de texto em linguagem natural são importante parte da memória organizacional e é desejável, dentro deste contexto, que o máximo de conhecimento seja armazenado e facilmente recuperável onde, quando e no modo em que for necessário. Para isto, deve-se contar com ferramentas que suportem a marcação de texto.

Para uma aplicação prática, e eventual prova de conceito e validação de resultados de realizar o enriquecimento de documentos de texto, imagina-se um sistema capaz de unificar a modelagem sintática (estrutural, mas não no sentido lingüístico) com a modelagem conceitual (semântica) do documento. Sendo que o resultado prático desta modelagem seria a geração de documentos XML capazes de guiar a criação dos documentos de texto devidamente enriquecidos com seus metadados.

Documento de texto é um arquivo com o texto e, para os propósitos deste trabalho, estaria codificado utilizando o formato XML definido por documentos XML.

Documento XML é um arquivo contendo estruturas na "linguagem" XML (DTDs, XMLSchemas, XSLT, XQuery e outros) com a especificação das tags e instruções para

garantir a validade de um pacote de dados XML e/ou definir as transformações a serem aplicadas a estes dados, entre outros.

A utilização de um editor de texto guiado por XML é o caminho para o enriquecimento do documento, já que realizará a marcação *a priori* sem necessidade de complexos tratamentos de extração de informação. E, com o enriquecimento de documentos, ainda pode ser um importante auxílio para os métodos de estruturação *a posteriori*, caso sejam necessários.

3.3.1 Editor de XML

O nome "Editor de XML" normalmente é associado a um programa utilizado para codificar documentos XML, processo comumente denominado *authoring*. Guardando as devidas diferenças, pode-se dizer que um editor de XML é um ambiente de programação para as linguagens XML, já que ele permite o desenvolvimento, edição e teste de documentos XML a serem usados por alguma aplicação. A idéia é auxiliar o usuário facilitando a criação das prolixas e, muitas vezes, complexas estruturas XML.

Tipicamente estes programas são utilizados por programadores no sentido normalmente entendido, isto é, o responsável pela criação de código fonte da aplicação em que serão utilizados os documentos XML que estão sendo criados.

3.3.2 Editor de Texto Guiado Por XML

Por outro lado, um Editor de Texto Guiado Por XML, ou ***XML-based editing system*** como é algumas vezes referenciado, é um editor de textos no sentido clássico, isto é, um programa destinado à criação de documentos de texto. A diferença de programas conhecidos como OpenOffice e Word para um ***XML-based editing system*** é que este utiliza um ou mais documentos XML como guia para o formato do texto. Este documento XML irá determinar o tipo de dado que pode ser inserido em cada parte do documento, constituindo-se de algum modo em um editor de texto guiado por XML, já que obriga o usuário a ficar em conformidade com a estrutura definida no documento XML. Esta estrutura pode ser, por exemplo, restrições aos tipos de dados possíveis ou obrigatoriedade de determinadas informações relativas aos itens do documento.

Existem trabalhos acadêmicos a respeito de editores de texto guiados por XML, bem como produtos comerciais disponíveis no mercado, que se caracterizam como Editor de Texto Guiado por XML.

Como exemplo de pesquisa pode ser citado o editor UXO (MILDE, 2001), descrito como um sistema de edição baseada em XML. Ele permite a combinação de dados estruturados marcados (*tagged*) com informações importadas de banco de dados relacionais integrados usando uma interface JDBC.

Já um exemplo de aplicação prática, pode ser citado o projeto *Vex (http://vex.sourceforge.net/index.html)*, que é um editor para documentos XML, onde a estrutura XML é escondida do usuário e apresentada de forma gráfica permitindo a edição de documentos XML como se fossem simples documentos de texto, sendo que o editor trata de realizar a marcação do texto à medida que ele é editado, controlando quais elementos podem ser usados em determinada posição do texto, de acordo com um esquema DTD previamente definido. Vex é um editor de textos e não um editor de dados XML.

3.4 A Pobreza Semântica do XML

Documentos de texto já foram caracterizados como dados não estruturados que possuem uma estrutura embutida nos dados, descritiva e irregular.

Dizer que texto possui estrutura embutida é equivalente a dizer que textos não possuem estrutura explícita, mas uma estrutura semântica implícita, onde cada parte do texto tem um significado específico e relaciona-se com outras partes deste mesmo texto, e com informações e conhecimentos associados ao domínio e contexto no qual o texto foi produzido.

Por exemplo, em um contrato ou escritura de venda de um imóvel, uma pessoa que leia o texto consegue distinguir claramente estruturas como:

- As pessoas envolvidas e qual seu papel no contrato (comprador ou vendedor);
- O objeto do contrato, isto é, o imóvel que está sendo negociado;
- Características padrão associadas a pessoas (nome, endereço, documentos, etc) e imóveis (tipo, tamanho, valor, etc);
- Relacionamentos do ato de venda com as leis e restrições existentes;
- Cláusulas descritas no texto que fazem restrições a condições do negócio e salvaguardas legais às pessoas participantes.

Para contemplar casos como os descritos, é preciso utilizar uma representação dos metadados do documento que seja capaz de representar as três dimensões dos metadados, isto é, conteúdo (semântica), contexto e estrutura.

Uma tecnologia muito promissora é o padrão XML que é adequado para representar alguns aspectos da estrutura, mas não a semântica e contexto do texto. Isto se deve ao fato que XML não possui semântica intrínseca (GIL, 2002), (KUDRASS, 2001), (PEASE, 2002) e (WELTY, 2000).

A falta de semântica nos padrões XML levaram à procura de soluções capazes de representar a semântica dos dados codificados em XML.

Independente do propósito, a maior parte dos trabalhos propõe a associação da representação estrutural dos dados (XML) com uma representação semântica que possa preservar os metadados ou ainda enriquecer o conteúdo dos dados em questão. Grande parte dos trabalhos recentes utiliza uma ontologia para tentar alcançar este enriquecimento.

A utilização de ontologias para enriquecer dados XML e facilitar consultas e acesso por parte de usuários finais é uma técnica muito estudada. Grande parte dos conceitos utilizados por estes trabalhos usa como referência a pesquisa de Michael Erdmann (ERDMANN, 2000) e (ERDMANN, 1999), que resume as justificativas para a utilização de ontologias associadas a estruturas em XML. Segundo Erdmann, XML é somente uma linguagem de representação para especificar a estrutura de documentos, ou seja, sua dimensão sintática. A estrutura do documento pode representar algumas propriedades semânticas porem limitadas.

Para permitir a interpretação semântica completa de documentos marcados com XML, esta marcação precisa ser complementada por um modelo conceitual que

represente a semântica completa do documento. Sendo que a principal deficiência concentra-se na impossibilidade de representar adequadamente os relacionamentos e restrições existentes entre os componentes do modelo.

Erdmann reafirma a falta de semântica nas representações em XML e preconiza o uso de uma estrutura de representação complementar ao XML, especificamente uma ontologia, que irá definir o significado dos tags utilizados para permitir a interpretação e consulta aos documentos com base no seu significado real.

Estendendo esta idéia para o uso em documentos de texto, pode-se também utilizar uma ontologia para o enriquecimento destes, acrescentando metadados que preservem as três dimensões, isto é, a descrição formal do conteúdo, contexto e estrutura do texto.

A utilização de uma ontologia, deste modo, vem ao encontro dos conceitos de enriquecimento de documentos e indexação conceitual apresentados nos capítulos anteriores, tornando-se efetivamente o instrumento que viabiliza este enriquecimento.

4 SERVICE MODELING LANGUAGE - SML

Estruturar informação em formato de texto é um dos motivos que levaram à criação do XML. Existem muitas linguagens para a modelagem de documentos em XML (XML), a atual solução padronizada pelo W3C é denominada XMLSchema (XML Schema).

XMLSchema, no entanto, apenas especifica a estrutura sintática do texto, isto é, quais os itens que podem aparecer, onde poderão aparecer e algumas restrições de cardinalidade. No entanto, XMLSchema não possui mecanismo para a representação semântica, principalmente no tocante a impossibilidade de representar adequadamente os relacionamentos e restrições existentes entre os componentes do modelo (ERDMANN, 2000).

Porém, considerando as suas deficiências, como representar conceitos, relacionamentos e estrutura mais complexas de documentos de texto? A solução mais citada é associar ao esquema ou documento XML uma ontologia (GUARINO, 1995), (LASSILA, 2001), (GÓMEZ-PÉREZ et al., 2004), (STAAB & STUDER, 2004) que descreva os conceitos e relacionamentos do documento (MOTTA et al., 2000), (ERDMANN, 2000). Porém, estas linguagens são bastante complexas e fogem ao conhecimento dos profissionais de informática, o que torna difícil seu uso. Por outro lado, XMLSchema vem sendo usado amplamente no desenvolvimento dos mais diversos tipos de sistema e já é incorporada às ferramentas utilizadas no dia-a-dia pelos profissionais de informática.

Para uma solução que seja ao mesmo tempo simples e de amplo conhecimento e que possua maior expressividade semântica que o XMLSchema, GraphSchema propõe o uso da Service Modeling Language ou SML (SML), uma linguagem derivada do XMLSchema. Esta abordagem é inédita, tendo em vista que esta linguagem foi projetada para a modelagem de ambientes computacionais (RIVALDO et al., 2007), (RIVALDO et al., 2008) e também pelo fato que SML teve seu primeiro *draft* liberado para o público em julho de 2006 e, até a data da conclusão deste trabalho, ainda encontra-se em processo de padronização pelo W3C (SML WD).

SML utiliza como base o XMLSchema, o qual foi estendido com o acréscimo de referências interdocumentos (de um documento para outro) e de regras de validação expressas utilizando-se da linguagem Schematron, padronizada pela ISO/IEC (ISO/IEC 19757-3). Esta combinação de XMLSchema estendido com Schematron endereça o problema existente na utilização do XMLSchema para a descrição dos conceitos de documento de texto, isto é, a capacidade de descrever restrições *(constraints)* e regras complexas de relacionamento entre os elementos do texto, além de, especificamente para o caso de contratos (documentos de texto "legal"), possibilitar a referência entre documentos relacionados.

4.1 Descrição da Linguagem SML

Na nomenclatura padrão do SML, um modelo é um conjunto inter-relacionado de documentos, ou arquivos, XML divididos em duas classes: Documentos de definição e documentos de dados, também chamados de instâncias. Os documentos de definição descrevem o esquema do modelo, i.e., como as instâncias, que são documento que contêm os dados propriamente ditos, devem ser estruturadas. Portanto, para SML, um modelo é o conjunto da especificação dos documentos e os documentos propriamente ditos.

Conforme já mencionado, SML usa um perfil do XML Schema 1.0 (XML Schema Structures) como linguagem de definição de esquema (estrutura). Já as regras são expressas em Schematron, que utiliza expressões XPath 1.0 (XPath) para criar asserções sobre os valores das instâncias e permitir a sua validação. Estas regras de validação são avaliadas como expressões booleanas sobre os dados em uma operação chamada de validação que consiste em confrontar as instâncias com a definição (esquema e regras). Dependendo do resultado da avaliação da regra, uma mensagem definida no corpo da regra é gerada. Esta mensagem é construída como parte da regra, sendo de escolha do responsável pela modelagem.

Resumindo, a estrutura de um modelo SML é capturada de dois modos:

1. **Esquemas *(Schemas)***: Estes são restrições na estrutura e conteúdo dos documentos, expressas em uma linguagem XMLSchema estendida.

2. **Regras *(Rules)***: São expressões booleanas que avaliam a estrutura e conteúdo dos documentos no modelo, incluindo tanto os esquemas como as instâncias. Ou seja, as regras podem impor restrições aos dados, mas também podem ser usadas para criar restrições complementares que não são possíveis de serem expressas utilizando XMLSchema.

A especificação SML define a operação de validação, que consiste em checar se todos os dados em um modelo satisfazem os esquemas e as regras.

Esta especificação foca-se na definição das extensões ao XMLSchema para referências interdocumentos, a definição de como as regras Schematron devem ser incluídas no modelo e o processo de validação (SML WD).

Diz-se que um determinado conjunto de documentos é um modelo SML válido se ele satisfaz os seguintes critérios:

1. Todos os documentos do modelo devem ser document XML bem formados (*well-formed*)

2. Cada documento XML Schema pertencente à definição deve satisfazer as condições definidas na especificação do XMLSchema (*XML Schema Structures*)

3. As regras expressas em Schematron nos modelos de definição de documento devem ser expressões Schematron válidas de acordo com a especificação ISO (*ISO/IEC 19757-3*)

4. Para cada instância no modelo, a propriedade "validade" do elemento raiz e todos seus atributos e descendentes não pode ser falsa, quando a validade do esquema é avaliada por um validador que esteja em conformidade com os documentos XMLSchema presentes nos documentos de definição (esquemas) do modelo (*XML Schema Structures*)

5. Cada documento no modelo deve satisfazer as regras Schematron aplicáveis a ele, isto é, as que foram definidas no modelo.

6. Cada document no modelo deve satisfazer as regras `sml:acyclic` e `sml:target`.

Exemplos podem ser encontrados na especificação SML (SML WD).

4.1.1 Referências

O suporte para referências interdocumentos inclui múltiplos esquemas de endereçamento, restrições no tipo de elemento referenciado e a possibilidade de definir restrições de identidade do tipo *key*, *unique* e *key reference* através de referências interdocumentos.

Uma referência em SML é um link de um elemento em um modelo SML para outro elemento do mesmo modelo, porém não necessariamente no mesmo documento, ou arquivo, XML. As referências devem obrigatoriamente ser suportadas pela implementação do validador SML.

A semântica das referências inclui a possibilidade de um elemento alvo ser referenciado a partir de múltiplos elementos origem e a obrigatoriedade de uma referência não nula obter como resultado no máximo um elemento do modelo. Uma referência nula é aquela que o resultado da avaliação da função `deref()` é vazia.

4.1.2 Função deref()

A resolução de referências é realizada pela função `deref()`, sendo que um validador SML deve obrigatoriamente implementar esta função. Trata-se de uma função que estende o XPath com a capacidade de resolver referências expressas no padrão escolhido pela implementação do validador. Esta função obtém um conjunto de nodos de elementos correspondentes aos elementos referenciados (destino) pelo conjunto de nodos de elementos definidores da referência (origem).

4.1.2.1 Múltiplos esquemas de endereçamento

SML pode usar padrões tais como URI Schema (XML Schema Datatypes) e Endpoint References (EPRs) (WS-Addressing Core) para representar referências. Porém estes esquemas são apenas sugestões, sendo que a especificação deixa a escolha final para a implementação do validador. Os elementos representando referências devem ser identificados por `sml:ref="true"` ou `sml:ref="1"` onde `sml:ref` é um atributo SML global cuja declaração e a seguinte:

```
<xs:attribute name="ref" type="xs:boolean" />
```

4.1.2.2 Restrições Para Validar o Elemento Referenciado

SML disponibiliza os seguintes tipos de restrições *(constraints)* para garantir a validade de elementos referenciados:

Tabela 4.1: SML, Restrições Para Validar o Elemento Referenciado

Nome	Descrição
`sml:acyclic`	Especifica se a referência deve ou não ser acíclica, isto é, se uma referência pode percorrer outras referências e eventualmente voltar à referência de origem, criando um ciclo. Geralmente as referências devem ser acíclicas.
`sml:targetElement`	Usada para restringir o nome do elemento alvo.
`sml:targetType`	Usada para restringir o tipo do elemento alvo.
`sml:targetRequired`	Usada para especificar que o elemento alvo deve obrigatoriamente estar presente no modelo, isto é, não pode resultar em uma referência nula.

4.1.2.3 Restrições de Identidade

Um validador de modelos deve suportar as seguintes restrições de identidade através de referências:

Tabela 4.2: SML, Restrições de Identidade

Nome	Descrição
`sml:key`	Similar ao `xs:key` do XMLSchema, exceto que a expressão XPath e campos selecionadores podem usar a função `smlfn:deref`
`sml:unique`	Similar ao `xs:unique` exceto que a expressão XPath e campos selecionadores podem usar a função `smlfn:deref`
`sml:keyref`	Similar ao `xs:keyref` exceto que a expressão XPath e campos selecionadores podem usar a função `smlfn:deref`

4.1.3 Regras

SML utiliza um perfil do Schematron para incluir suporte para regras definidas pelo usuário. Schematron, por sua vez, faz uso de expressões XPath, sendo que em SML XPath é estendido com a função `smlfn:deref`. Conforme já citado, um validador de modelos SML que queira estar de acordo com a especificação da linguagem deve obrigatoriamente suportar e avaliar expressões XPath 1.0 acrescidas com a função `smlfn:deref()` no corpo das regras Schematron.

As regras definidas pelos usuários são especificadas usando os elementos Schematron `sch:assert` e `sch:report`. O exemplo a seguir obtido na especificação SML (SML WD) apresenta duas regras:

- Um endereço IPv4 deve conter quatro bytes;
- Um endereço Ipv6 deve conter dezeseis bytes.

Tabela 4.3: Exemplo de Regra em Schematron

```
<xs:simpleType  name="IPAddressVersionType">
       <xs:restriction base="xs:string" >
           <xs:enumeration value="V4" />
           <xs:enumeration value="V6" />
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="IPAddress">
        <xs:annotation>
            <xs:appinfo>
                <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
                   <sch:ns prefix="tns" uri="urn:IPAddress" />
                   <sch:pattern id="Length">
                      <sch:rule context=".">
                         <sch:assert test="tns:version != 'V4' or count(tns:address) = 4">
                              A v4 IP address must have 4 bytes.
                         </sch:assert>
                         <sch:assert test="tns:version != 'V6' or count(tns:address) = 16">
                              A v6 IP address must have 16 bytes.
                         </sch:assert>
                      </sch:rule>
                   </sch:pattern>
                </sch:schema>
            </xs:appinfo>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="version" type="tns:IPAddressVersionType" />
            <xs:element name="address" type="xs:byte" minOccurs="4" maxOccurs="16" />
        </xs:sequence>
</xs:complexType>
```

4.2 Ontologias

Nos itens anteriores foi utilizado o conceito de ontologia sem apresentar uma definição. Para isto, é preciso definir domínio de conhecimento, que é um subconjunto do conhecimento sobre o mundo real restrito a uma área ou especialidade da ciência ou outra área de atuação humana.

Entendendo o conceito de domínio de conhecimento, pode-se definir uma ontologia como um modelo, ou representação, do conhecimento no nível simbólico (NEWELL, 1982) de um domínio.

Sendo uma representação ou modelo, para criar uma ontologia faz-se necessária a utilização de algum tipo de formalismo, estrutura ou linguagem (GÓMEZ-PÉREZ et al., 2004).

A criação de uma ontologia seria então a tentativa de representar total ou parcialmente algum tipo de conhecimento de modo a explicitá-lo e permitir o compartilhamento e uso deste por agentes humanos ou máquinas. A questão de utilização da ontologia por uma máquina está diretamente ligada ao grau de formalismo que a mesma irá possuir.

Pode-se entender que a utilização de uma ontologia permitirá definir com maior ou menor precisão o contexto, representação do conhecimento e vocabulário utilizados para um domínio, dependendo do seu grau de completude, aderência ao "senso comum" dos especialistas no domínio representado e adequação do formalismo utilizado.

4.2.1 Ontologias Fracas e Fortes

A definição de ontologia como linguagem de representação do conhecimento permite classificar as diferentes linguagens de ontologia existentes em um espectro ontológico em grau crescente de formalismo matemático. Surge então uma linha divisória imaginária onde se passa de definições informais para definições formais (LASSILA, 2001):

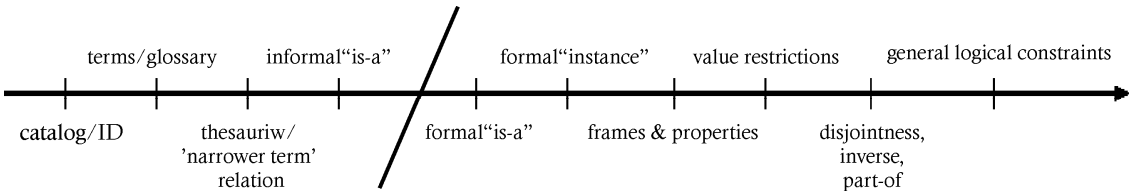


Figura 4.1: Espectro Ontológico (LASSILA, 2001)

Nesta classificação, as ontologias transitam entre uma lista de termos, ou vocabulário controlado, diferentes formas de representação de conhecimento baseadas em teorias clássicas (BARR & FEIGENBAUM, 1981), tais como, redes semânticas (Sowa-sm) e *frames* (MINSKY, 1975), *multiple hierarchical restricted domains* (Taxonomias, Partonomias, Topologia, Temporalidade) (BÉJAR et al., 2004) chegando até representação baseadas em lógica formal (BAADER et al., 2004), entre outros formalismos.

Esta classificação abre caminho para um grande debate sobre o que é uma ontologia e como representá-la, mas, ao mesmo tempo, permite classificar grande parte das linguagens de marcação de texto como linguagens de representação de ontologias, bastando classificar corretamente esta linguagem em relação ao grau de formalismo utilizado.

4.2.2 Componentes de Uma Ontologia

Sendo uma ontologia a representação do conhecimento de um domínio, ela deve possuir uma linguagem com estruturas, ou construtos, capazes de representar os conceitos, relacionamentos entre estes conceitos, regras e restrições e valores do domínio em questão. A palavra linguagem é usada no sentido mais amplo possível de linguagem de programação de computadores englobando desde simples representação de dados até complexos sistemas formais baseados em lógica. Deste modo, o conceito

de linguagem pode englobar representações gráficas ou textuais capazes de serem processadas por máquina e também entendida por seres humanos.

Idealmente, uma ontologia deve ser facilmente entendida por humanos, pois contém algum formalismo de modo a não possuir ambigüidade nos conceitos representados, já que a ambigüidade causaria problemas para o processamento por máquina.

As linguagens de representação de ontologias possuem construtos e estruturas que refletem o formalismo no qual são baseadas (GÓMEZ-PÉREZ et al., 2004).

Recentemente, com a introdução do conceito de *semantic web*, tem ocorrido uma consolidação das formas de representação de ontologias em direção a alguns padrões, principalmente baseados nos modelos de *frames*, redes semânticas e *description logic* (DING et al., 2005), porém, ainda assim, existem inúmeras linguagens adequadas a diferentes propósitos e domínios (GÓMEZ-PÉREZ et al., 2004), (JASPER, 1999).

Uma ontologia deve expressar conceitos do mundo que deseja representar. Tipicamente uma ontologia terá os seguintes construtores (GRUBER, 1993):

- Conceitos (ou classes);
- Relações entre os conceitos;
- Funções;
- Axiomas lógicos com restrições necessárias a interpretação não ambígua dos conceitos e relações;
- Relacionamentos:
  - Classificação;
  - Especialização;
  - Agregação;
  - Associação de conjunto;
  - Associações definidas com semântica particular ao domínio, por exemplo:
    - pertence;
    - desenvolvido por;
    - restringe;
    - evolui para;
    - usa;

4.3 SML Como uma Linguagem de Criação de Ontologias

A utilização de ontologias é uma área de grande atividade na pesquisa de estruturação de documentos de texto seja diretamente, seja para acrescentar uma dimensão semântica à estruturação de textos com XML (ERDMANN, 1999).

Conforme apresentado anteriormente, as ontologias podem ser classificadas de acordo com seu grau de formalismo matemático/lógico. A complexidade das linguagens ditas fortes tem levado a pesquisa de alternativas mais simples, geralmente variações do XML. SML pode ser classificada como uma linguagem de descrição de ontologias sem

um formalismo lógico, ou seja, uma linguagem de descrição de ontologias do tipo *informal is-a* (LASSILA, 2001).

A ontologias leves expressas em SML tem sido utilizada com sucesso em algumas aplicações comerciais e projetos de pesquisa. Esta abordagem está presente, por exemplo, em produtos de software de gerenciamento de redes disponíveis no mercado tais como Microsoft SMS e IBM Tivoli.

SML também já foi aplicada com sucesso na descrição de modelos de ambientes computacionais para gerenciamento de ambiente de grid computacional (RIVALDO et al., 2007) e (RIVALDO et al., 2008).

4.4 SML Como Suporte Para o Enriquecimento de Documentos

Como foi visto anteriormente, uma das características de uma ontologia é a possibilidade de definição das relações entre os conceitos representados e restrições aplicáveis a este conceito. Usando este aspecto de uma ontologia que descreva formalmente os conceitos do domínio do documento, pode-se especular que seria possível avaliar se um documento de texto está completo, isto é, se contemplam todas as informações e relações que o domínio daquele documento exige.

Isto poderia ser usado, por exemplo, para garantir que um documento jurídico em que um dos participantes seja menor de idade e, portando, incapaz perante a lei, contenha também as informações do representante legal deste menor: Pai e Mãe, ou quem estiver responsável perante a lei por aquele menor. O que seria impossível de ser realizado somente com a utilização de XML Schema.

Ao pesquisar uma linguagem para a representação dos modelos de contrato, uma das principais questões levantadas foi a complexidade das linguagens de descrição de ontologias, não apenas no tocante ao conhecimento necessário para sua utilização, mas também pela dificuldade de obter ferramentas para seu processamento e, também, pela dificuldade de escolha de uma entre as muitas linguagens disponíveis. A solução adotada pelo GraphSchema baseia-se na redução da complexidade do modelo gerado internamente.

Por isto foi descartada a utilização de uma linguagem ontológica formal em favor de uma abordagem nova, baseando-se nos seguintes critérios:

- Por ser uma extensão do XMLSchema, SML é bastante simples se comparada, por exemplo, com OWL;
- As extensões do SML contemplam as necessidades de modelagem de documentos no tocante a referências inter e intradocumentos;
- A capacidade de criação de regras em Schematron complementam as deficiências do XMLSchema, permitindo a representação de modelos complexos e relações entre os diversos elementos do documento;
- A disponibilidade de ferramentas para tratamento de XMLSchema e Schematron é muito maior que para outras linguagens, devido a ampla gama de linguagens existentes em nichos específicos;

- Schematron é baseado em XPath, que possui uma capacidade de processamento procedural não presente em linguagens baseadas em lógica, facilitando a criação das regras para os modelos;
- Os modelos são simples XML, portanto também são passíveis de serem implementados em uma linguagem gráfica compreensível pelo usuário;
- O formato de documento adotado pela linguagem gráfica do GraphSchema é diretamente traduzido em XMLSchema, isolando o usuário da linguagem interna na qual os modelos são expressos.

5 MODELAGEM DE DOCUMENTOS DE TEXTO LEGAIS

5.1 Contexto

Todos, em algum momento, necessitam de algum tipo de contrato.

Estes textos jurídicos são caracterizados por terem algumas estruturas de texto padronizadas (ou padronizáveis) que devem ser adaptadas a cada caso. Estas estruturas, geralmente textos, representam itens do documento, tais como dados das pessoas e seus representantes, atos jurídicos, documentos, declarações necessárias e bens envolvidos, entre outros.

A enorme diversificação dos itens componentes do texto, devido à imensa gama de combinações e situação legais possíveis, gera uma situação onde, apesar de composto por textos padronizáveis, os documentos terminam sempre tendo a necessidade de adaptação direta em editor de texto. E, mais importante, é que esta diversidade implica na necessidade de conhecimento de inúmeras leis aplicáveis a cada detalhe particular do texto final gerado, o que leva à necessidade de revisão por parte de um profissional extremamente especializado.

Finalmente, a possibilidade de uma determinada pessoa retornar para confecção de outros documentos, ou mesmo para outros serviços, exige o armazenamento dos dados para reutilização, quando necessário, envolvendo a necessidade de pesquisa nos dados e documentos do passado.

Para facilitar o trabalho, sistemas existentes baseiam-se numa combinação de modelos de texto pré-formatados, com os atos jurídicos conhecidos e outros itens, com formatação padronizada dos textos. A adaptação do texto é feita com a marcação de todas as palavras que necessitam ser alteradas em função das variações do número e gênero de pessoas e bens envolvidos, sendo que normalmente não é garantida a completude e correção intra-itens do texto gerado. Esta arquitetura exige enorme esforço e intervenção de pessoal especializado para criação e manutenção de modelos e da já citada revisão dos aspectos jurídicos do texto.

Devido à complexidade inerente ao problema, os sistemas existentes são pouco flexíveis, voltados para resolução de problemas específicos. Como, por exemplo, sistemas para tabelionatos que não são adequados para a utilização em outras entidades com necessidades semelhantes tais como: registro de imóveis, imobiliárias e escritórios de advocacia.

Com esta estrutura fixa ou pouco flexível, estes sistemas servem apenas como passo inicial para a geração de um texto que será realmente trabalhado em um editor de texto

padrão, que é o modo que se encontra para dar a necessária flexibilidade ao sistema. Este formato final de texto nos leva a mais um aspecto: a questão do armazenamento. Devido ao formato final do produto do sistema ser essencialmente um texto, ou "o texto" dos documentos, a informação originalmente inserida para a geração do documento pode, e tipicamente é, diferente da informação final contida, exigindo muito retrabalho e esforço por parte dos utilizadores para manter os dois coerentes. E, na maior parte dos casos, a recuperação de informação e, principalmente, a utilização da mesma em outras aplicações é simplesmente inexistente.

5.2 Contratos e a Origem do GraphSchema

A automação tem sido a mola propulsora do aumento de produtividade da sociedade moderna. Algumas tecnologias são tão importantes que conseguem romper paradigmas e levar a atividade à qual for aplicada a um novo patamar. Na área jurídica, apesar de inúmeros esforços, esta revolução ainda não aconteceu. Estudos (MOUNTAIN, 2007) apontam a existência de três barreiras principais:

- Falta de pessoal especializado
- Regras contra praticas não autorizadas
- Falta de capital

Segundo (MOUNTAIN, 2007), a geração de documentos de texto é a mais fundamental tecnologia aplicável à área jurídica, pois está no coração do trabalho legal. Por isto, tecnologias que permitam o aumento da produtividade neste trabalho possuem um enorme potencial em relação à produtividade da atividade jurídica, e podem até ser consideradas como uma tecnologia que permite mudança de paradigmas neste ramo de atividade.

A proposta de GraphSchema enquadra-se na classificação de geração de contratos no macro-nível, e este trabalho mais especificamente na montagem de modelos que possam ser usados posteriormente para a geração e validação destes documentos.

GraphSchema originou-se na experiência profissional do autor quanto à necessidade de aumento da produtividade na geração de contratos e outros tipos de escrituras públicas ou particulares em tabelionatos, escritórios de advocacia e outras atividades que exijam constante geração de contratos. Para este tipo de atividade, a solução mais comum é a combinação de modelos de texto previamente preparados e utilização de um editor de texto para completar o texto.

O problema deste método é que ele não consegue garantir a validade jurídica dos documentos gerados, pois, para isto, é necessária a análise e revisão por parte de um profissional especializado. A disponibilidade de um sistema que permita a garantia da validade do documento, sem necessidade de envolvimento de um profissional altamente especializado, abre caminho para o aumento da produtividade (MOUNTAIN, 2007).

Uma solução possível está na criação de contratos a partir de uma ferramenta, tipicamente um editor de texto, que guiará o usuário para a criação de textos em consonância com modelos conceituais de documentos alvo.

A criação do modelo exige que o profissional responsável seja altamente especializado, pois deve considerar todos os aspectos envolvidos tais como cláusulas necessárias, documentos obrigatórios, inter-relacionamentos entre itens do documento e conformidade com a legislação vigente e princípios legais. Já a criação do documento

pode ser realizada por um profissional com menos conhecimento ou experiência, se este puder contar com um sistema que garanta o atendimento aos modelos existentes.

Esta idéia, no entanto, não é nova, sendo este conceito geralmente denominado “editor de texto guiado por XML”. Trata-se de separar a etapa de modelagem da etapa de criação do documento.

Uma ferramenta capaz de realizar esta tarefa é bastante complexa, não somente do ponto de vista da engenharia de software necessária para o seu desenvolvimento, mas, principalmente, na geração de modelos conceituais de documentos. Isto se deve à dificuldade de aprendizado de linguagens de modelagem de dados ou representação de conhecimento, especificamente XML e as tecnologias associadas. A linguagem GraphSchema propõe-se a resolver este problema utilizando uma representação gráfica que seja compreensível ao usuário.

A linguagem GraphSchema isoladamente não apresenta uma solução completa para o usuário. Isto porque, além da linguagem e da uma ferramenta capaz de auxiliar o usuário na criação de modelos utilizando esta linguagem, são necessários outros componentes. Especificamente, é necessário que os modelos sejam utilizados de alguma forma. Para isso, GraphSchema deve ser associado a um editor de textos capaz de utilizar os modelos para guiar o usuário na confecção do texto.

Neste contexto, a Figura 5.1 a seguir apresenta o processo de criação de um documento.

Inicialmente o modelo do documento em GraphSchema é criado por um editor de modelos, por exemplo, o protótipo desenvolvido como parte desta dissertação, aqui chamado simplesmente de , GraphSchema Editor. Este não somente permite a criação de modelos gráficos como também transforma a representação gráfica em um modelo SML.

O Este modelo criado é que será finalmente utilizado por um editor de texto guiado por SML para a criação do documento propriamente ditode texto, de acordo com o esquema pre-definido.

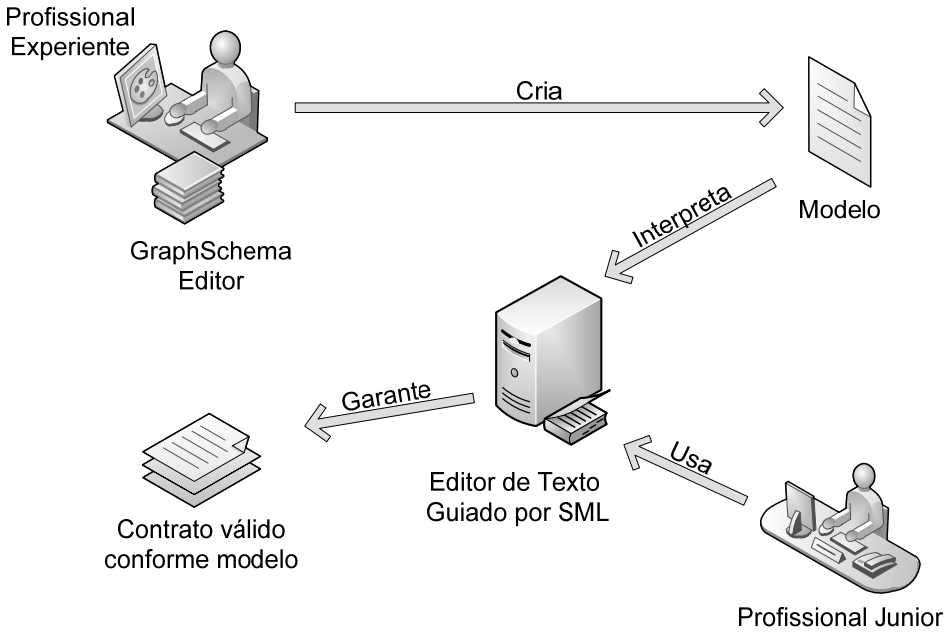


Figura 5.1: Processo de Criação de um Documento usando Esquema do Documento

5.3 Estruturação de documentos Legais com XML

Ao mesmo tempo em que pressupõe ou até mesmo exigem regras rígidas na sua confecção, documentos legais podem ter uma ampla variação no tamanho. Estes documentos são geralmente organizados em itens e subitens que referenciam não somente itens do próprio documento como também itens de outros documentos. Devido a estas características, documentos de texto no domínio legal são bastante propícios à modelagem e utilização de XML (HATTER, 2006). Mas a utilização de XML irá exigir diversas considerações, algumas específicas do domínio legal e outras genéricas aplicáveis à formatação de documentos com XML.

Em primeiro lugar, têm-se considerações relativas ao modelo e sua interação com os usuários, escopo, evolução ao longo do seu ciclo de vida e como o modelo de dados, ou mais especificamente do documento, será representado. Este aspecto é de suma relevância na proposta do GraphSchema.

Do ponto de vista deste trabalho, a questão mais importante é quem irá ser responsável pela modelagem, tendo em vista que o modelo do documento pode sofrer alterações com o tempo, o que leva à questão de ser ou não este modelo compreensível ao usuário e acessível para modificação por parte deste. Deve-se considerar também qual o escopo pretendido, pois este está intimamente relacionado à perspectiva do responsável pela modelagem.

Em segundo lugar, o formato no qual será expresso o modelo, do ponto de vista do usuário, deve ser o mais simples possível para permitir a sua compreensão. Porém, na representação interna, aquela que o usuário não enxerga, este modelo deve suportar requisitos como:

- Utilização e/ou compatibilidade com padrões existentes;
- Expressividade da notação, que deve ser capaz de representar com acuidade os

detalhes do domínio escolhido, ao mesmo tempo em que não deve introduzir complexidades que evitem ou dificultem a criação de ferramentas para o usuário final;

- Validação de um documento em relação ao modelo;
- Referências internas, relativas a conteúdo do próprio documento;
- Referências externas, relativas a conteúdo contido em outros documentos baseados ou não no mesmo modelo de documento;
- Reuso e modularização dos seus componentes.
- Disponibilidade de ferramentas capazes de suportar a tecnologia escolhida.

A representação do modelo de dados de documentos utilizando XMLSchema e DTD, por si só não é capaz de garantir o atendimento a estes requisitos. Faz-se necessário a utilização conjunta, ou alternativa, de linguagens de descrição capazes, não apenas de expressar a estrutura sintática do documento, como também sua estrutura semântica, permitindo a validação tanto do modelo de dados como de características que não podem ser expressas somente na estrutura do documento tais como inter-relacionamento de itens ou restrições complexas. Para isto, pode-se complementar XMLSchema e DTD ou utilizar independentemente linguagens de validação tais como RELAX NG ou Schematron. Devido às características da maioria dos contratos (Daskalopulu, 1997), a utilização de linguagens de representação de ontologias tais como RDF e OWL introduzem um grau de complexidade desnecessários, ao mesmo tempo em que não possuem ferramentas padronizadas para processamento algorítmico, baseando-se em complexos modelos de prova lógica (regressão).

Uma análise em alto nível da adequação da representação do modelo de dados e das linguagens de validação é apresentada em (HATTER, 2006). Nesta análise são separadas as capacidades de modelagem dos dados, ou estrutura, das capacidades de validação para as diversas técnicas apresentadas. A linguagem SML abrange aspectos tanto de modelagem de dados como de validação, com a possibilidade de criação de referências interdocumentos. Portanto, com base neste estudo, podemos afirmar que SML é uma alternativa mais abrangente, i.e, com maior expressividade semântica, que as linguagens apresentadas já que estas de modo geral contemplam apenas uma característica, i.e., ou são linguagens de modelagem ou são linguagens de validação.

5.4 Macro-Nível, Micro-Nível

A geração automatizada de contratos pode ser discutida em dois níveis (DASKALOPULU, 1998):

- No macro-nível *(macro-level)* o enfoque é a produção de documentos com uma estrutura coerente em relação a um modelo previamente definido, sem, no entanto, preocupar-se com a validação do que o documento significa, pois este tratamento é assumido como responsabilidade do modelo.
- No micro-nível *(micro-level)* o enfoque é a produção de documentos bem formados que, além de implementar um modelo previamente definido, ainda preocupa-se com a adequação do modelo e o significado individual dos componentes do documento, entrando em detalhes da estrutura semântica do documento.

O macro-nível caracteriza-se por três aspectos principais (DASKALOPULU, 1995):

- A separação entre a representação e os mecanismos ou ferramentas que irão manipular o documento;
- A modelagem da estrutura e inter-relacionamento entre os itens constituintes do documento, mas não com o significado do texto propriamente dito;
- O processo de criação do documento está sujeito a um conjunto de regras que especificam sua estrutura e restrições entre itens.

5.5 Técnicas para o Esboço Automatizado de Documentos Jurídicos

No artigo *"Techniques for automated drafting of judicial documents"* (BRANTING, 1998), o problema de criação de documentos é apresentado sob o ponto de vista de classes de documentos, descrevendo três técnicas diferentes a serem aplicadas conforme o tipo de documento e objetivo pretendido. A Tabela 5.1 compara as três técnicas.

Tabela 5.1: Técnicas Para o o Esboço Automatizado de Documentos Legais

Técnica	Dificuldade de Geração	Dificuldade de Manutenção e Validação	Escopo	Expressividade Semântica
Procedural	Media	Alta	Baixo	Baixa
Baseada em templates	Baixa	Baixa	Baixo	Baixa
Baseada em discurso	Alta	Baixa	Alto	Alta

A técnica baseada em templates como, por exemplo, a utilização de modelos de texto prontos em editores de texto padrão, mostra-se como a mais fácil e acessível para o usuário final, porém apresenta um baixo grau de expressividade semântica do texto final. Isto se deve ao fato que a utilização de templates apenas ajuda a gerar o texto, mas não acrescenta metadados, i.e., não realiza o enriquecimento do documento e também tem escopo limitado quanto ao tipo de documento aplicável. Além disto, não contempla a referência inter e intradocumentos.

O ideal do ponto de vista do usuário seria um sistema que apresentasse baixo grau de dificuldade na criação de modelos, permitisse a validação de regras simples e complexas e, principalmente, que seja capaz de representar com acuidade a estrutura conceitual do documento, sem adicionar trabalho para o usuário.

5.6 Esboço Automatizado de Documentos

Dada a sua importância o esboço automatizado de documentos apresenta-se como uma classe específica de ferramentas e possui inúmeros estudos dedicados ao assunto (SARTOR, 2006), (MOENS, 2006), (MERCATALI, 2000), (LAURITSEN, 2007), (BRANTING et al., 1997), (BRANTING et al., 1999), (BIAGIOLI, 2005), (BIAGIOLI, 2003), (LEHTONEN , 2002), (DASKALOPULU, 1998), (DASKALOPULU, 1997), (GOTTSCHALK, 2007) e (BRANTING, 1998).

A nomenclatura para os sistemas que geram textos legais, no entanto, apresenta diversas denominações tais como: *automatic document drafting, document modeling systems*, *automatic document assembly, document assembly systems* entre outros.

Apesar de cada denominação representar ferramentas com funcionamento e arquitetura diferente, de modo geral, pode-se dizer que todos tratam da criação de texto legais com o auxílio de uma ferramenta de software.

Para os propósitos deste trabalho, serão considerados os sistemas de montagem de documentos que armazenam modelos *(templates)* criados por especialistas jurídicos utilizando as mais diversas formas de representação interna. Estes *templates* contêm porções de texto fixo juntamente com instruções precisas de como devem ser usados os elementos de texto disponíveis. Tais sistemas irão guiar e orientar o usuário nos detalhes relevantes para a criação do documento desejado. Geralmente, isto é feito através da interação entre o usuário e o sistema na forma de perguntas ou opções que o usuário deve responder ou escolher para a geração do documento. O sistema então utiliza as respostas e escolhas do usuário para automaticamente gerar um documento com base no conhecimento e modelos disponíveis no sistema.

Deste modo, define-se um sistema de suporte ao rascunho de textos legais como um programa ou sistema que automatiza, em parte ou no todo, a construção de documentos cujo conteúdo dependa de regras jurídicas (MOENS, 2006).

Restringe-se esta definição a documentos de texto em linguagem natural que contêm conteúdos relacionados à legislação e/ou práticas legais adotadas no país. A este texto, pode-se acrescentar metadados, no processo conhecido como enriquecimento de documentos.

Em particular, há interesse na classe de documentos relacionados a contratos, definidos como *contract drafting*, i.e., rascunho ou esboço de contratos.

5.6.1 Tipos de Sistemas de Esboço Automatizado de Documentos Legais

No artigo *"Improving Access to Legal Information: How Drafting Systems Help"* (MOENS, 2006), a autora avalia a importância dos sistemas de esboço de documentos legais (*Legal Drafting Systems*) sob a ótica da recuperação de informações, fazendo uma excelente análise histórica da evolução desta categoria de sistemas. Baseado na gama de funcionalidades obtida a partir desta análise histórica, os sistemas de rascunho de documentos legais são classificados em quatro categorias:

- Sistemas Informativos e Funções de Ajuda
- Sistemas de Montagem e Geração de Texto
- Sistemas de Verificação
- Sistemas de Tradução

Estas categorias permitem caracterizar o tipo de sistema proposto nesta dissertação e são descritas a seguir.

5.6.1.1 Sistemas Informativos e Funções de Ajuda

São programas de computador que oferecem informação ao usuário sobre o processo de contrução do documento *(drafting)*, sem, no entanto, criar nenhum tipo de obrigação, ou seja, o usuário decide se ele vai aceitar ou ignorar as dicas geradas pelo sistema. O sistema LEDA *(Legislative Drafting and Advisory system)* (VOERMANS, 1995) é um exemplo desta classe.

A principal vantagem dos sistemas informativos é que eles têm maior probabilidade de serem aceitos pelos usuários devido ao fato de não serem forçados a seguir regras, porém, como conseqüência, existe o problema do usuário simplesmente ignorar as recomendações do sistema.

5.6.1.2 Sistemas de Montagem e Geração de Texto

Exemplos de sistemas desta classe estão presentes em certas funcionalidades do sistema SOLON (DEBAENE et al., 2000) e no sistema DocuPlanner (BRANTING et al.,1999) e (BRANTING et al.,1997).

Esta classe de sistema constrói documentos legais utilizando informações disponibilizadas pelo usuário em conjunto com a representação do conhecimento sobre os aspectos formais e conteúdo de documentos legislativos contidos no sistema. O usuário é forçado a seguir a estrutura rígida disponibilizada pelo sistema, o que geralmento é percebido pelo usuário como um ataque a sua liberdade de expressão ou capacidade técnica, mesmo que os textos produzidos pelo sistema mostrem-se mais corretos e compreensíveis que textos gerados pelo usuário sem o auxílio deste.

5.6.1.3 Sistemas de Verificação

Exemplos desta classe são os sistemas Lexedit (MERCATALI, 2000) e EnAct (ARNOLD-MOORE, 1998) e (WILKINSON et al.,1998) e também estão presentes em certas funcionalidades do sistema SOLON (DEBAENE et al., 2000). Sistemas de Verificação dão ao usuário total liberdade, podendo este criar o texto do modo que achar mais adequado e, somente após ter o texto, o sistema irá ser usado para validar, ou verificar, a adequação do texto com diversos critérios tais como adequação a estruturas pré-definidas e adequação no uso da terminologia. Como resultado da verificação, o sistema irá sugerir correções, o que neste ponto ainda permite ao usuário ignorar as sugestões do sistema, deixando-o totalmente livre para decidir o formato final do texto.

5.6.1.4 Sistemas de Tradução

Estes sistemas contemplam a tradução automática ou semi-automática de textos “puros”, isto é, sem qualquer tipo de metadados ou marcação, em formatos que irão incluir marcação para processamento por máquina. Podendo ser considerados sistemas de marcação de texto *a posteriori*.

Nesta classe, encontram-se sistemas como "Norme in Rete" (NIR) (BIAGIOLI, 2003) e (BIAGIOLI, 2005) e também o sistema REGNET (KERRIGAN; LAW, 2003).

A informação proporcionada pela marcação facilitará a integração, sem intervenção humana, destes textos em sistemas de informação que oferecem serviços de recuperação de informação, suporte à decisão ou assistência para validação de adequação a normas. Normalmente esta classe de sistema é usada para a geração de legislações e regulamentações legais, dado que estes textos já existem em formato de texto em linguagem natural.

5.6.2 Classificação do GraphSchema

Com base nestas classificações, pode-se dizer que GraphSchema apresenta uma proposta inédita justamente por disponibilizar uma ferramenta que o usuário tome as decisões sobre a modelagem, aumentando assim a aceitação dos modelos e regras. Deste modo, GraphSchema aparece como uma ferramenta para o usuário final construir seus próprios padrões e não como um conjunto complexo de normas e modelos pré-definidos, como é comum a maioria dos sistemas estudados.

5.7 Padrões existentes

Além de sistemas propriamente ditos, é importante que se avalie alguns sistemas e padrões existentes, mostrando as contribuições específicas do GraphSchema.

5.7.1 LegalXML eContracts

A marcação de textos legais tem produzido inúmeros trabalhos de pesquisa, sistemas e padrões destinados não somente a permitir a troca de informação através de meios eletrônicos, como também preservar a dimensão conceitual dos textos legais através da inserção de metadados.

Um dos trabalhos mais completos é a familia de padrões LegalXML (http://www.legalxml.org). Desde a sua fundação em 1998, o grupo de trabalho dedicado à criação do padrão tem proposto uma estrutura normativa consistente para a padronização de documentos legais. Atualmente, os padrões definidos pelo LegalXML estão distribuídos em grupos de trabalho dentro do consórcio Oasis (Organization for the Advancement of Structured Information Standards), (http://www.oasis-open.org), uma entidade sem fins lucrativos, dedicada a desenvolver e incentivar a adoção de padrões abertos para a troca de informação.

Especificamente, LegalXML desenvolveu o padrão eContracts, com a missão de promover a eficiente criação, manutenção, gerenciamento, compartilhamento e publicação de documentos e termos contratuais. O esquema eContracts propõe-se a descrever a estrutura genérica hierárquica de toda a gama de documentos contratuais, sendo, portanto, de propósito geral. Com isto, pretende disponibilizar uma estrutura padrão que será utilizada como suporte para processamento por máquina dos mais diversos tipos de contratos.

Na especificação do eContracts (eCONTRACTS, 2007), o grupo de trabalho entende e incentiva a transformação do padrão em uma base comum de formatos XML a serem utilizados por editores de texto guiados por XML, permitindo assim a confecção de contratos em conformidade com o padrão estabelecido.

Porém, ao estabelecer um padrão genérico, eContracts deixa livre para que a implementação do padrão seja feita de acordo com as necessidades das aplicações específicas. Neste ponto é que se pode encaixar o GraphSchema como uma ferramenta capaz de gerar modelos de documentos compatíveis com o padrão.

eContracts define a estrutura básica de um documento contratual através de uma hierarquia recursiva baseada em itens. Por ser recursiva, um item pode representar capítulos, partes, seções, claúsulas e outros elementos presentes em contratos. A partir desta hierarquia, eContracts especifica um vocabulário e uma série de atributos e termos comumente utilizados em contratos tais como: nome, endereço, partes, objeto, ato

jurídico e outros. Esta estrutura hierárquica é consistente com o "modelo de documento" adotado pela representação gráfica do SML implementada no GraphSchema.

Pode-se dizer que GraphSchema é uma excelente alternativa para a popularização do padrão eContracts, já que este define uma estrutura padrão enquanto GraphSchema permite ao usuário criar documentos compatíveis com esta estrutura. Para isto, é apenas necessário que seja criada uma biblioteca com o vocabulário especificado pelo eContracts e, a partir desta biblioteca, o usuário poderá criar seus modelos.

Mais ainda, a utilização de SML permite implementar integralmente o padrão, pois contempla tanto a especificação estrutural, como também possui a capacidade de implementar o elemento *simple condition*, que é previsto, mas não detalhado, na especificação do eContracts. Este elemento prevê que um contrato possa incluir regras para complementar a especificação, ou esquema, estrutural do contrato. Deste modo, SML mostra-se mais adequado que a utilização de XMLSchema, DTD, RelaxNG ou outras linguagens de especificação de documentos XML que não possuem suporte para a criação de regras.

5.7.2 NIREditor

O projeto NIR (BIAGIOLI,2005) destina-se a facilitar a navegação e recuperação de documentos em um ambiente distribuído. Para isto, definiu padrões XML e URN e definiu a criação de ferramentas que utilizem estes padrões.

NIR define dois padrões para a descrição normativa dos documentos: UNR *standard,* para referências cruzadas, e 3 padrões DTD, representando os modelos dos documentos. Basicamente, os documentos são definidos com dois tipos de elementos: Elementos estruturais e metadados.

Elementos estruturais descrevem o formato do texto normativo (perfil formal) e são:

- Elementos genéricos, tais como referências para outras leis, textos formatados (tabelas, listas, etc);
- Elementos normativos específicos tais como cabeçalhos, seções, artigos e parágrafos.

Dois tipos de metados são disponibilizados:

- Metadados Gerais (assunto, classificação, data da publicação e relacionamento entre atos);
- Metadados Analíticos (tipo de provimento jurídico: Adendos, Apelações e substituições) e regras tais como obrigações, definições, penalidades, etc.

Metadados gerais disponibilizam informação sobre o ato jurídico e os metadados analíticos descrevem a semântica dos provimentos.

As principais características do NIREditor são:

- Apoio para *drafting* de textos legais (perfil formal);
- Qualificação de documentos com metadados (perfil funcional);
- Manipulação de documentos legados de acordo com o padrão NIR-DTD definido.

Note-se que o foco do projeto NIR está na definição de padrões formais e ferramentas que o usuário possa usar para criar textos legais em conformidade com estes padrões. Porém, diferente da proposta do GraphSchema, o projeto NIR não teve a preocupação de disponibilizar ferramentas para que o usuário possa criar seus próprios modelos, já que o padrão NIR define formalmente como os documentos devem ser estruturados e quais os componentes possíveis de serem utilizados. Além disto, a interface disponível segue o tradicional padrão de apresentar a estrutura XML como uma árvore, conforme pode ser visto na Figura 5.2.

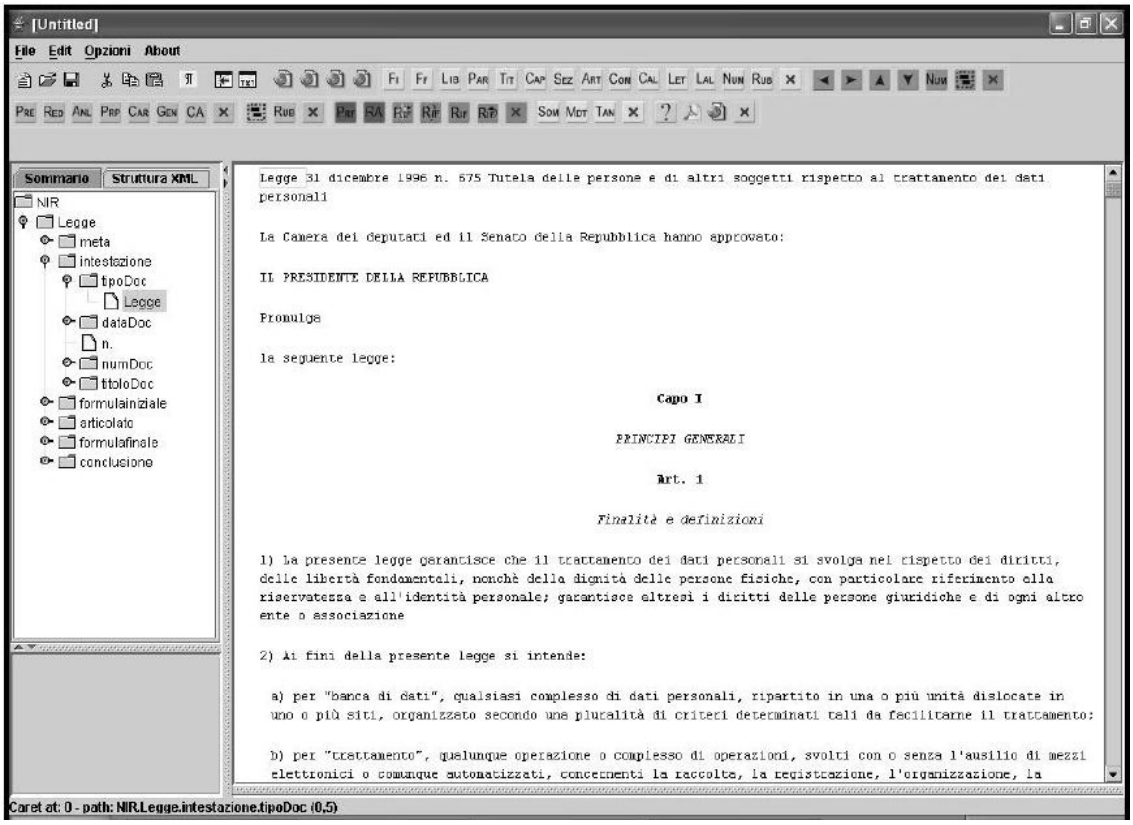


Figura 5.2:NIREditor

5.7.3 MetaLex

O padrão METALex (BOER et al., 2002), (BOER et al., 2003) nasceu devido à necessidade de levar em consideração todos os aspectos legais na ocasião da criação de documentos no âmbito da comunidade européia. Esta necessidade advém da crescente dificuldade de acesso a fontes de conhecimento sobre a estrutura legal e, principalmente, ao aumento da complexidade das legislações em nível municipal, regional, nacional, europeu e internacional, como conseqüência da unificação da Comunidade Européia.

METALex é um padrão e *framework* genérico XML e com previsão para facilidades de extensão para a codificação da estrutura e conteúdo de documentos legais e "para-legais", sendo que seu primeiro objetivo é servir como um formato de troca de documentos legais.

METALex difere de outros esquemas de metadados existentes em dois aspectos:

- ele é independente de linguagem e jurisdição;
- propõe-se a acomodar o uso do XML além dos serviços de apresentação e busca.

METALex define conceitos para descrição da legislação, gestão de citações (legais) e versionamento, tratamento independente de linguagem e jurisdição e mecanismos de extensão.

Os conceitos de descrição de legislação baseiam-se em três pontos de vista:

- formato: determina a classe e propósito do documento legal através da análise de seu formato e frases usadas;
- papel: além do texto, o papel *(role)* exercido pelo documento é de fundamental importância para sua classificação;
- conteúdo: a análise do conteúdo também é usada para a classificação de documentos.

Destinado a ser um padrão aberto, METALex foi lançado em setembro de 2002 (BOER et al., 2002) e, apesar do tempo existente desde o seu lançamento, a atividade registrada no site http://www.metalex.nl mostra que não houve muita evolução na aplicação e desenvolvimento do padrão. Isto reforça a tese que, apesar de necessários, os padrões de formatação não são suficientes para popularizar o uso de documentos marcados *(tagged)*, tanto devido à complexidade inerente à definição de um padrão capaz de atender a todas as necessidades, como na indisponibilidade de ferramentas para que os usuários possam modificar e adaptar os modelos em seus subdomínios específicos.

5.7.4 Contract Expression Language (CEL)

A Contract Expression Language – CEL (WANG et. al.) é uma linguagem baseada em XML projetada para expressar acordos contratuais entre diferentes partes (agentes), com o propósito de capturar e comunicar a informação contratual e possibilitar a execução e cumprimento do mesmo através de processamento por máquina em relação às permissões , obrigações e proibições. Além de modelar acordos contratuais usando os conceitos de obrigação, proibição e permissão definidos pela *deontic logic* (SARTOR, 2006).

CEL utiliza XML para definir sua sintaxe e semântica, criando uma nova linguagem baseada no MPEG REL (MPEG REL, 2003). Utiliza os conceitos relevante, elementos e tipos definidos no REL como seus blocos construtores básicos. Também se baseia no XMLSchema, o que permite a separação entre os elementos centrais e a especificação de seus relacionamentos na forma do seu modelo de dados retirado do vocabulário específico da aplicação.

Ao ser comparado com outros padrões, especificamente com OASIS LegalXML eContracts (eContracts), CEL tem seu foco na semântica dos contratos, portanto apresenta-se como uma abordagem complementar e não concorrente ao eContracts.

Do mesmo modo, ao ser comparado com GraphSchema, CEL preocupa-se em definir uma linguagem capaz de representar direitos e obrigações. Já GraphSchema assume que a utilização de uma linguagem existente pode suprir esta necessidade, sendo portanto, uma abordagem simplificada sobre o mesmo problema. Deste modo, o foco do CEL é a definição de uma linguagem, enquanto GraphSchema parte de uma linguagem existente e propõe a simplificação da criação de modelos pela transferência desta responsabilidade ao usuário final, através da criação de uma ferramenta que isole este usuário da linguagem na qual o modelo será expresso, permitindo que o usuário concentre-se na questão da modelagem e não em como esta modelagem é armazenada.

5.7.5 Rule-based XML

Rule Based XML (EGUCHI & LAURENCE, 2002) representa contratos e outros documentos comuns ao sistema legal dos EUA, utilizando XML não somente como linguagem de marcação do texto, mas também representa com XML as regras sobre os procedimentos de contratos e litígios. Rule Based XML não deve ser confundido com o projeto RuleML (BOLEY, 2002), que apenas trata de um formato para comunicação e transferência de regras codificadas em XML.

Uma base de regras *(rulebase)* é desenvolvida através da marcação de exemplos de XML que devem ser interpretados pelo sistema especialista.

Entre outras conclusões, os autores do Rule Base XML reafirmam a importância da inteligência artificial para o tratamento de litígios legais, porém, mais importante para o contexto do trabalho do GraphSchema, é a afirmação de que o uso de regras em XML "estendido" é adequado ao tratamento do conhecimento contido em contratos, reforçando a tese do GraphSchema. Também afirmam que esta solução é ideal por aproveitar-se da sinergia entre a marcação de texto em XML e o uso de inteligência artificial para interpretação de regras descritas em um formato de fácil processamento por máquina.

Como complemento, o artigo apresenta um sistema especialista que codifica as regras usando XML e demonstra uma GUI que irá auxiliar o usuário a escrever as regras para o motor de inferência. Na Figura 5.3, pode-se avaliar a GUI utilizada. Esta é complexa, não esconde o XML do usuário, portanto não apresenta a facilidade proposta pelo GraphSchema.

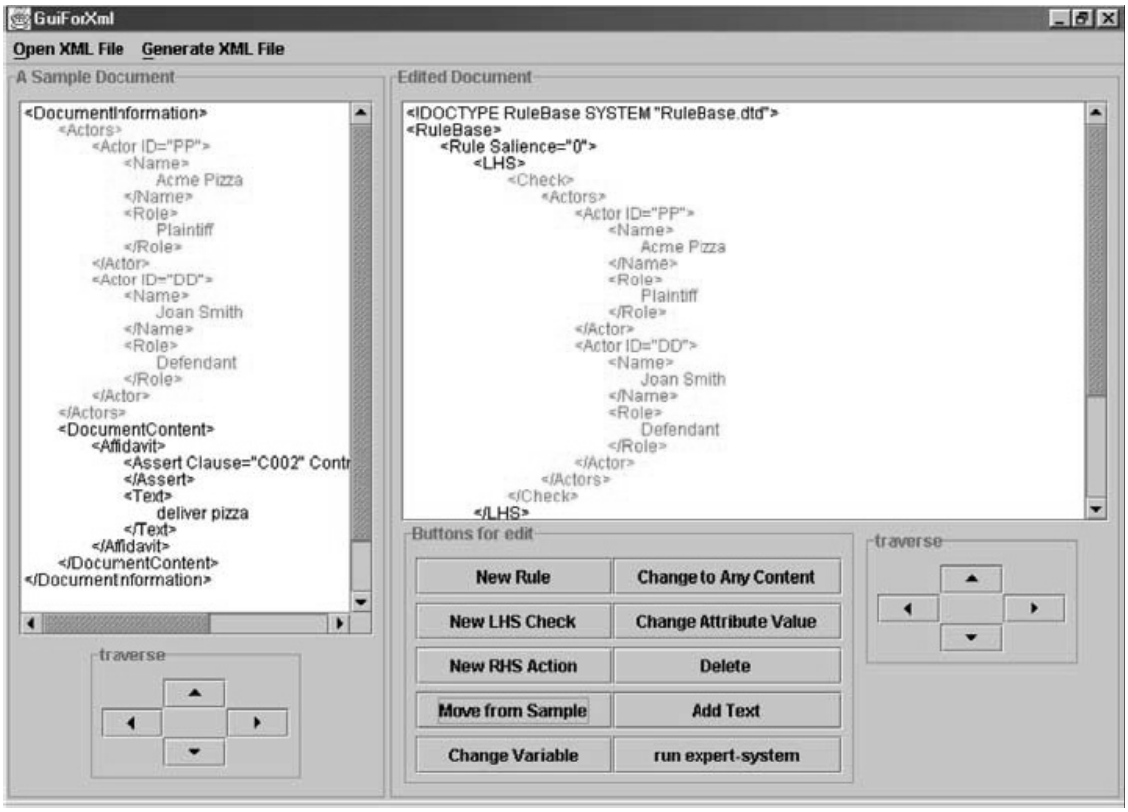


Figura 5.3:Rule Based XML GUI

6 REPRESENTAÇÕES GRÁFICAS E XML

Um aspecto importante desta pesquisa está associado ao modo como o usuário visualiza a representação do modelo. É importante resaltar que a representação aqui referida não é a representação interna dos dados, mas o modo como o usuário visualiza estes dados.

Buscando uma solução para viabilizar o entendimento do modelo sem expor a linguagem internar, buscou-se trabalhos relacionados a visualização de XML como ponto de partida para a criação da linguagem GraphSchema.

Historicamente, ao tratar computacionalmente os dados não estruturados, os autores têm utilizado uma abordagem de representação baseadas na idéia de estruturas em árvore ou grafo.

Por exemplo, a linguagem G-Log (PAREDAENS et al., 1995) utiliza uma representação em forma de grafo. Usando o mesmo paradigma, foram propostas linguagens e representações para dados não estruturados baseados neste modelo. Por exemplo, a Figura 6.1 representa um conjunto de dados sobre filmes (BUNEMAN, 1997):

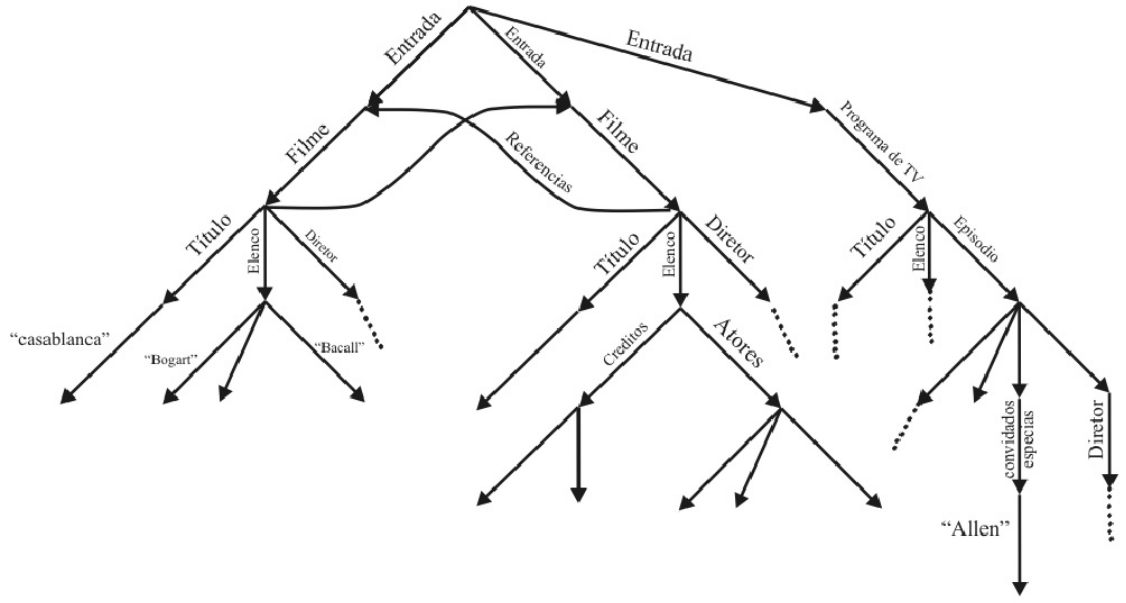


Figura 6.1: Dados XML Representados como Grafo

Os dados estão representados na forma de um grafo direcionado rotulado, onde as arestas representam os atributos, e as folhas representam os valores para os atributos. Também é importante observar que existe um vértice raiz, representando "o documento", ou "os dados". Este vértice serve de ponto de entrada para a navegação na hierarquia montada. Pode ser feita uma analogia direta entre este tipo de grafo e XML. Os *tags* em XML são sempre definidos com base no caminho percorrido até ele. Usando um exemplo simples, o nome do filme "Casablanca" é o valor para o *tag* /Entrada/Filme/Título.

A representação de dados não estruturados também é representada em formato de árvore em trabalhos ligados à recuperação e consulta em dados não estruturados na web (FLORESCU, 1998).

Novamente, analisando o problema da apresentação visual, e não da representação interna, os editores de XML existentes, sejam eles softwares acadêmicos, comerciais ou de código aberto, também utilizam o paradigma de grafos rotulados para auxiliar a criação *(authoring)* de documentos XML. Para ilustrar, a Figura 6.2, mostra a edição de uma estrutura em XMLSchema e foi obtida de um dos editores de XML líderes no mercado:

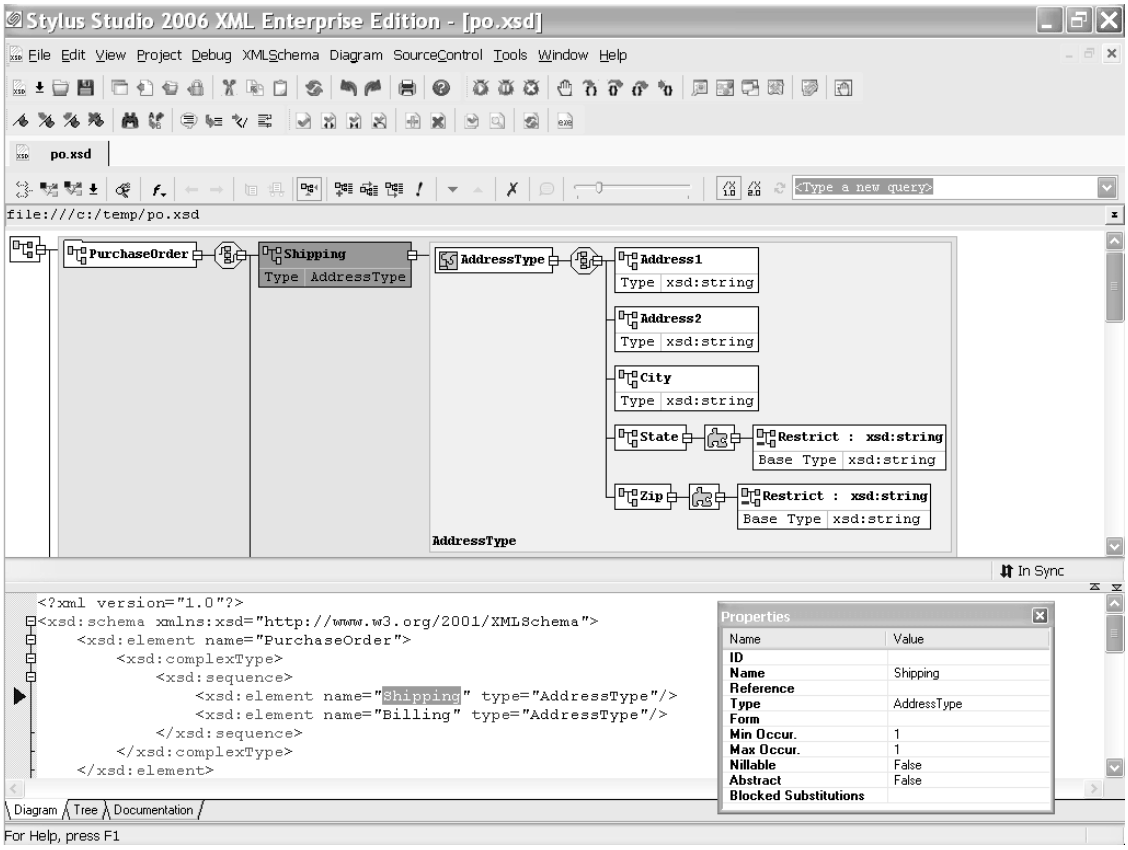


Figura 6.2:Tela de Um Editor de XML Orientado a Dados

Talvez estes programas realmente auxiliem a criação de modelos XML, dada a sua natureza complexa e prolixa, e também a necessidade de integração de múltiplos arquivos tais como XMLSchema e XSLTs. Ainda assim, a maioria destes programas são realmente ferramentas de programação, requerendo um conhecimento de XML por parte dos seus usuários.

No entanto, a idéia de usar os grafos ou árvores não esconde a complexidade, apesar de auxiliar na organização da atividade de criação e teste dos documentos XML.

Existem poucos trabalhos estudando a relação de produtividade entre o uso de uma ferramenta com linguagem visual e a produção de documentos XML. Os estudos existentes sobre linguagens visuais estão concentrados na questão de consulta a documentos XML. Entretanto, alguns destes trabalhos apresentam idéias para linguagens visuais que podem ser evoluídas de linguagens de consulta para linguagens de descrição do modelo conceitual de documentos de texto.

A maioria dos trabalhos de representação visual de XML reproduz o paradigma usado internamente, isto é uma árvore ou grafo para representar os documentos XML. Porém alguns autores trabalham com a idéia de visualização em estrutura de documento, isto é, o modelo é apresentado em formato gráfico que lembra um documento de texto. Este formato é mais intuitivo para um usuário de editores de texto. Nestas ferramentas, o usuário pode modificar a quantidade de texto exibido alterando o tamanho da fonte, isto é, fazendo operaçoes de zoom in e zoom out. A linguagem GraphSchema baseia-se exatamente neste conceito que consiste em visualizar a estrutura do documento: items, paragrafos, seções, etc. Sem necessidade de apresentar o texto propriamente dito.

6.1 Linguagens Gráficas Para Tratamento de Dados Não Estruturados

Não há particularmente interesse nas capacidades de consulta destas linguagens, mas nos gráficos utilizados que podem servir de base para uma linguagem de vizualização de modelos conceituais de documentos de texto.

Para entender a linguagem GraphSchema define-se paradigma de grafo e paradigma de documento. Esta classificação deve-se ao modo como a linguagem é vizualizada, ou representada graficamente.

No paradigma de grafo, a representação visual é, ou assemelha-se, a um grafo ou uma árvore.

No paradigma de documento esta mesma representação assemelha-se visualmente a um documento.

6.1.1 Linguagens Baseadas no Paradigma de Grafo

6.1.1.1 G-Log

G-Log (PAREDAENS et al., 1995) (1992, mas publicado só em 1995) lançou a idéia. Na realidade, é um formalismo matemático, baseado na lógica de primeira ordem, que é visualizado na forma de um grafo. Não tem relação com XML, mas com bancos

de dados relacionais. Tenta ultrapassar os limites impostos pelas linguaguens relacionais às aplicações não convencionais.

6.1.1.2 XML-GL

Baseada na G-log, XML-GL (CERI et al., 1999) é utilizada para consultas a fontes de dados em XML. Utiliza o formalismo gráfico da linguagem para representar as *querys* e, de modo ortogonal, também representa o conteúdo de fontes de dados XML e sua estrutura definida por DTD, o que, segundo os autores, permite a construção de *querys* complexas. Conclui que o uso de linguagem gráfica para consulta de documentos XML na web é bastante interessante.

6.1.1.3 VXT

VXT (PIETRIGA et al., 2001) se preocupa em facilitar a especificação das transformações e apresentação de dados XML utilizando um modelo gráfico próprio, que utiliza uma representação de grafo mapeada em uma estrutura aninhada para representar os DTDs e, também, um mecanismo de zoom para a navegação dos elementos da árvore do modelo DOM, conforme mostrado na Figura 6.3.

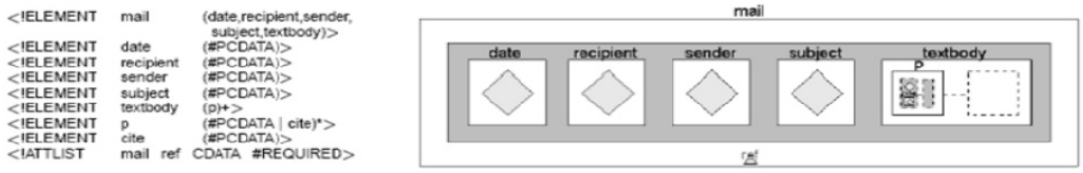


Figura 6.3: Dados XML Representados em VXT

A linguagem gráfica e seu uso são bastante complexos, não sendo de fácil entendimento por usuários. Mas o seu mecanismo de zoom, utilizado para esconder a complexidade dos dados e transformações, é uma idéia que pode ser interessante, se associada à abordagem de documento proposta por Xing.

6.1.2 Linguagens Baseadas No Paradigma de Documento

6.1.2.1 Xing

Xing (ERWIG, 2000), (ERWIG, 2003) é uma linguagem de consulta à fonte de dados XML que representa estes usando um paradigma de documentos, reproduzindo com exatidão os documentos encontrados no dia-a-dia, como cartas, faxes, descrição de produtos e todos os tipos de formulários. Os documentos são representados com o aninhamento dos seus elementos.

Propositalmente, o autor criou a linguagem independente da existência de DTDs para os dados a serem consultados, justificando que os dados na web geralmente não estão em conformidade com um DTD e que a existência de DTD obrigaria o usuário a ter conhecimentos do conceito de esquema conceitual.

Por decisão de projeto, Xing é uma linguagem destinada a ser simples e, portanto, de fácil entendimento por usuários leigos. De fato, ao examinar os exemplos e compará-los

com os outros modelos, parece a mais simples forma de representação gráfica entre as estudadas, principalmente devido à relação direta entre o modelo de documento e a percepção do usuário do que é um documento de texto.

Na Figura 6.4 o autor apresenta um exemplo de dados representando uma bibliografia (bib):

```
<bib>
   <livro ano="1998">
      <titulo>Concrete Mathematics</titulo>
      <autor>Graham</autor>
      <autor>Knuth</autor>
      <autor>Patashnik</autor>
   </livro>
   <artigo ano="1998">
      <titulo>Linear Probing and Graphs</titulo>
      <autor>Graham</autor>
      <journal>Algorithmica</journal>
   </artigo>
</bib>
```

Figura 6.4:XML Para Exemplos do Xing

Os dados XML apresentados na Figura 6.4 são representado em Xing conforme a Figura 6.5:

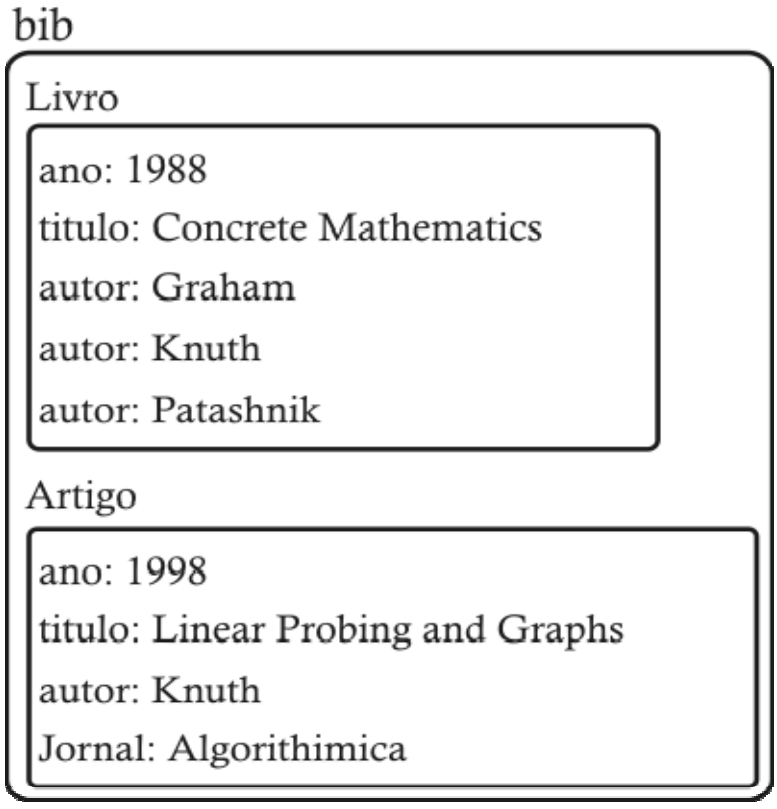


Figura 6.5:Representação Gráfica de Dados em Xing

Uma consulta que deseja obter todas as publicações cujo autor é Knuth é mostrada na Figura 6.6 e o resultado da consulta é apresentado na Figura 6.7.

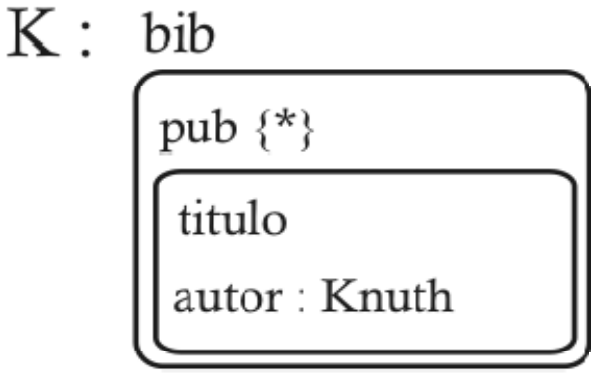


Figura 6.6: Consulta a livros cujo autor é “Knuth”

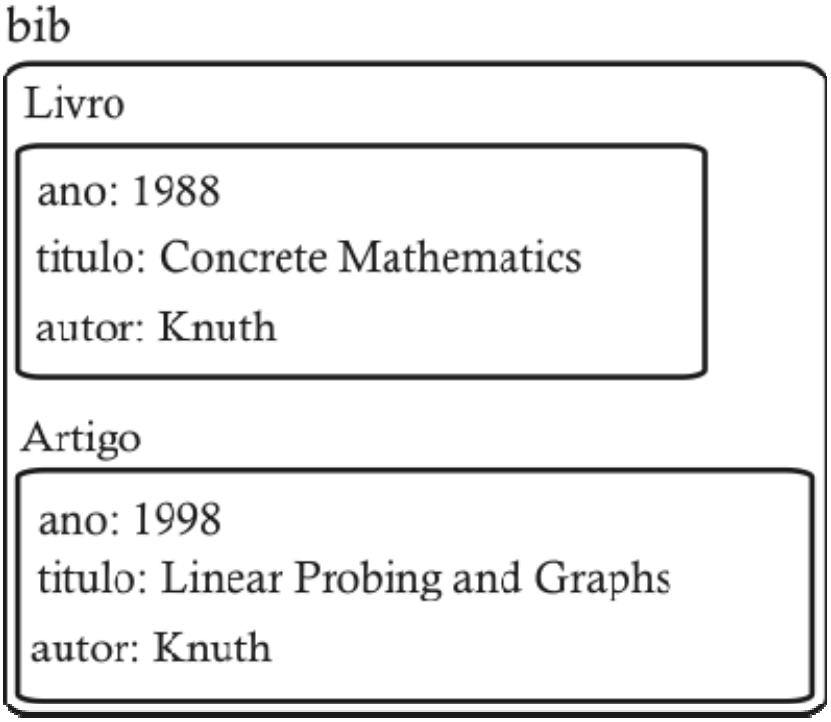


Figura 6.7: Resultado da Consulta a Livros

6.1.2.2 FoXQ

FoXQ (ABRAHAM, 2003) é uma linguagem de consulta a dados XLM, idêntica a Xing, com implementação de um protótipo em Java.

6.1.2.3 CCSL e VCCSL

CCSL (Content Container Style Language) e VCCSL (Visual Content Container Style Language) (CANFORA et al., 2002) abordam o problema da representação de transformações e estilo de apresentação de dados XML e se propõe a realizar este trabalho agregando em uma única linguagem as linguagens XSLT, XPath e XSL-FO.

VCCSL é a representação gráfica (visual) da linguagem CCSL. O objetivo explícito desta linguagem é permitir que o usuário defina os aspectos de apresentação e estilo de dados XML sem necessitar de conhecimentos de XSLT.

Por englobar três padrões complexos (XSLT, XPath e XSL-FO), CCSL é também uma linguagem complexa, porém a sua representação gráfica tenta minimizar o esforço necessário para a utilização destas tecnologias através de uma abordagem que agrega a formatação (XSLT e XSL-FO) com a lógica para extração dos dados em um paradigma de conteiner e conteúdo.

Apesar de visualmente parecer-se com a abordagem de documento usado por Xing, VCCSL é relacionada a uma árvore que representa a estrutura dos dados XML. O resultado obtido com CCSL é a geração de folha de estilo e formatação (XSLT e XSL-FO) para serem processadas por um *parser* XML com suporte a XSLT e XSL-FO.

(V)CCSL não lidam com a dimensão estrutural dos dados XML (DTD e XMLSchema) ficando somente preocupadas com os aspectos de apresentação.

A Figura 6.8 mostra exemplo de representação visual em VCCSL e a Figura 6.9 mostra o código XSL correspondente.

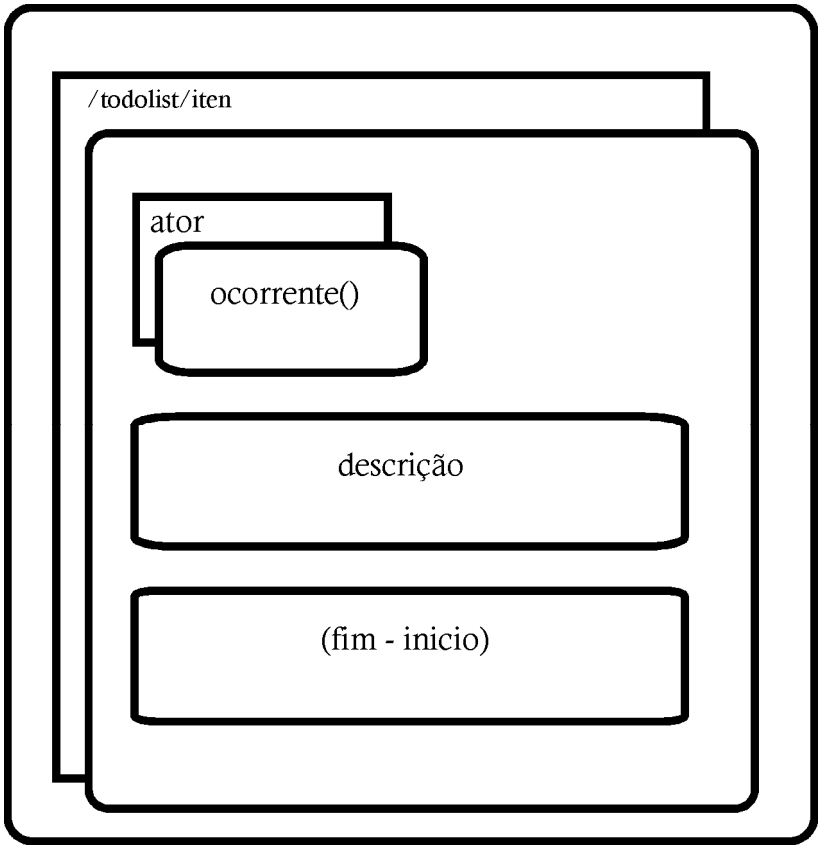


Figura 6.8: Exemplo Gráfico em VCCSL

```
<xsl : template  match="/">
  <xsl : for-each
          select="/todolist/iten">
   <xsl : for-each
           select= "actor">
   <fo:block>
      <xsl : value -of
              select= "current()" />
     </fo:block>
  </xsl : for-each>
  <fo:block>
     <xsl: value-of
         select= "description"/>
  </fo: block>
  <fo:block>
      <xsl: value-of
           select="end - start"/>
     </fo:block>
  </xsl:for-each>
</xsl:template>
```

Figura 6.9: XSL equivalente ao exemplo gráfico

7 GRAPHSCHEMA

7.1 Introdução

GraphSchema é uma linguagem visual compreensível ao usuário final que tem por objetivo a vizualização e criação de modelos conceituais de documentos de texto.

Diferente de outras abordagens, a proposta do GraphSchema é permitir que um usuário especialista no domínio jurídico possa criar modelos de contratos sem recorrer a auxílio técnico para o uso de uma linguagem de descrição de ontologias. Para a representação interna dos modelos, GraphSchema utiliza a Service Modeling Language (SML), uma linguagem resultante da união entre XMLSchema e Schematron, de origem recente, e que ainda não foi estudada como suporte a modelagem de documentos de texto.

Sendo uma linguagem e não um padrão, GraphSchema se caracteriza por não impôr um modelo de estrutura para os documentos, nem um vocabulário padronizado de termos e conceitos, como fazem a maior parte das tecnologias estudadas no capítulo 5. Estas concentram seu esforço na criação de estruturas e vocabulários padrão, ou mesmo, na criação de conceitos de relacionamentos que deverão ser obrigatoriamente usados para a descrição dos documentos.

Deste modo, GraphSchema apresenta uma proposta única no sentido de aumentar o poder do usuário, para que este utilize seu conhecimento de acordo com seu melhor julgamento do que é adequado ou necessário para a modelagem dos documentos, nos quais está interessado.

7.2 Definição

GraphSchema é baseada na representação visual denominada modelo de documento (*document model*) introduzido pela linguagem Xing (ERWIG, 2000), (ERWIG, 2003). Xing, no entanto, é projetada como uma linguagem de consulta a fonte de dados XML. O modelo de documento preconizado por Xing representa dados XML com o aninhamento dos seus elementos e mostrando o resultado final como se fosse a folha de um documento.

Por decisão de projeto, Xing é uma linguagem destinada a ser simples e de fácil entendimento por usuários leigos devido à relação direta entre o modelo de documento e a percepção do usuário do que é um documento de texto, com seus itens componentes e

inter-relacionamentos de conceitos, principalmente na questão da composição de itens a partir de subitens.

7.2.1 Overview

A abordagem adotada no GraphSchema não define um vocabulário, nem restringe o usuário a utilizar um vocabulário previamente definido. Com isto, GraphSchema pode ser usada para o desenvolvimento de modelos de documentos utilizando qualquer vocabulário padrão existente. Por exemplo, o vocabulário definido pelo LegalXML pode ser integralmente utilizado nos modelos, permitindo que o usuário defina modelos de documentos utilizando este padrão.

GraphSchema é uma representação gráfica da linguagem SML e esta, por sua vez, é uma extensão do XML Schema. Portanto, a estrutura básica é o Item. Os blocos construtores são representados conforme a Figura 7.1.

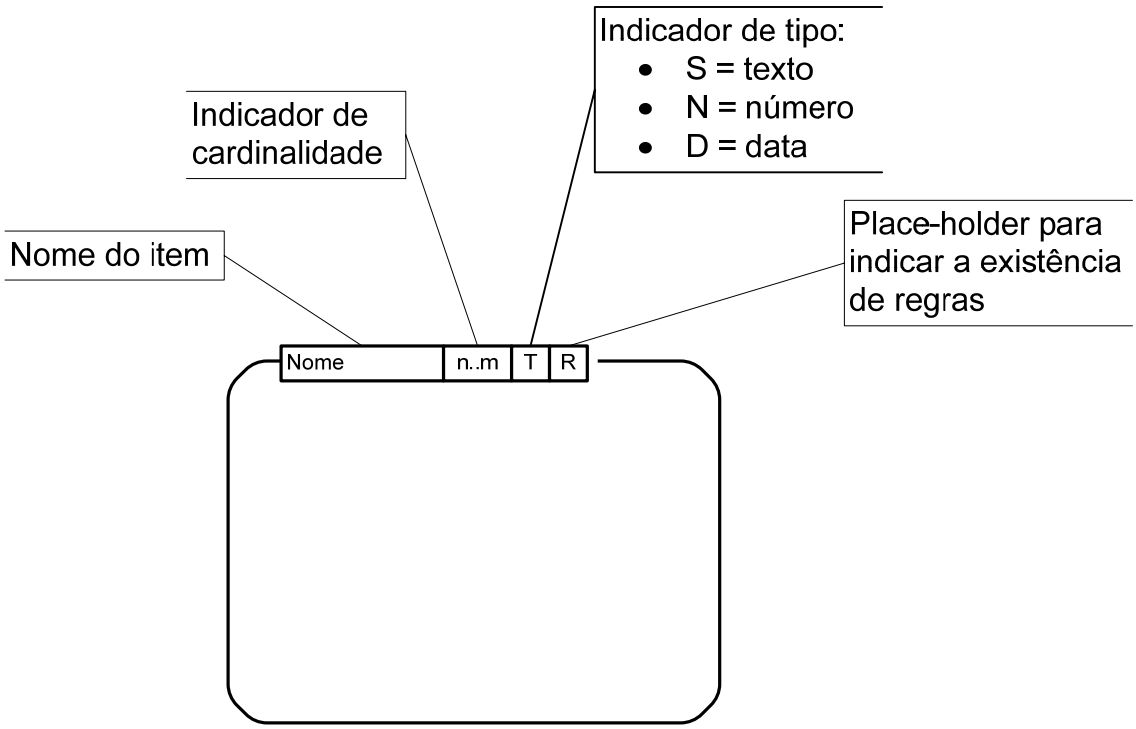


Figura 7.1: Representação de um item em GraphSchema

Como no Xing, o elemento individual é identificado por um retângulo que, além de seu nome, mostra a cardinalidade mínima e máxima, o tipo de dado primitivo, bem como a existência de *Rules* para este item. E, correspondendo a um esquema SML, no elemento poderá ser embutido outros elementos para criar elementos compostos.

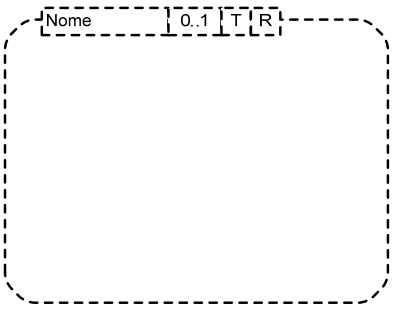
Além de ser mostrada explicitamente, a cardinalidade também é representada graficamente com o uso de linhas tracejadas para elementos opcionais e sombra para os itens com cardinalidade maior que um.

O retângulo "R" é somente um *placeholder* para localizar as regras associadas a este elemento. Se houver regras associadas a ele, aparecerá a letra "R" e, caso não existam,

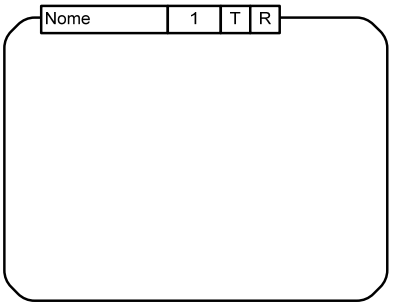
este retângulo estará vazio. Deste ponto, sairão linhas ligando o elemento aos outros com os quais tem regras de restrição.

Por convenção, será utilizada a denominação “representação gráfica” para as restrições que são apresentadas ao usuário utilizando a representação gráfica dos itens. Já para as restrições que não são diretamente visualizadas no gráfico, será usada a denominação “representação por expressão não gráfica” ou simplesmente “representação por expressão” ou ainda “representação não gráfica”.

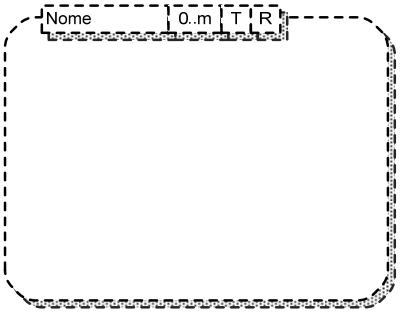
As possibilidades de criação de itens são representadas na Figura 7.2.



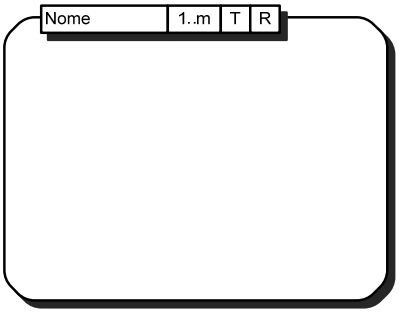
Item Opcional único
- linhas pontilhadas sem sombra
- Cardinalidade 0 ou 1



Item Obrigatório Único
- Gráfico: linha continua sem sombra
- Cardinalidade: 1



Item Opcional Múltiplo
- Gráfico: linhas pontilhadas com sombra pontilhada
- Cardinalidade: maior ou igual a 0 e menor ou igual a m



Item Obrigatório Múltiplo
- Gráfico: linha continua com sombra cheia
- Cardinalidade maior ou igual a 1 e menor ou igual a m

Figura 7.2: Variações Possíveis nos Itens

O grupo legalrdf.org (McCLURE, 2006) resume os tipos de declarações que podem ser encontradas em documentos, em especial em documentos legais:

1. Declaração de Existência: declara a existência de coisas, sejam elas entidades reais ou conceitos. A declaração de algo, neste caso, é suficiente para assumir a sua existência do ponto de vista do documento, independente da sua existência no mundo real;
2. Declaração de Definição: uma declaração de definição, além de afirmar que algo existe, também representa a definição deste conceito, entidade, ou evento, incluindo ou não o aspecto temporal, isto é, passado, presente, futuro. Para conceitos e objetos: existiu, existe, vai existir. Para eventos: ocorreu, ocorre, ocorrerá... e assim por diante.
3. Declaração Factual: declarações factuais representam informações que são sabidamente verdadeiras (ou falsas). Podem complementar declarações definicionais e existenciais, incluindo informação sobre propriedades dos objetos e conceitos e, também, relacionamentos entre eles.
4. Declaração Condicional: variação da declaração factual, neste caso afirma se algo é verdadeiro ou falso no modo condicional, associando ao fato uma expressão ou condição que, ao ser avaliada, irá definir o fato como verdadeiro ou falso.

GraphSchema permite a declaração de existência de conceitos, entidades e fatos diretamente através da criação de itens no modelo utilizando sua representação gráfica.

Para a declaração de definições, fatos e condições torna-se necessário utilizar combinações de itens (representação gráfica) que serão complementados por regras (representação por expressão).

7.2.2 Elementos

GraphSchema mantém a simplicidade da linguagem Xing. Para isto, define um conjunto mínimo de elementos a partir dos quais serão compostos os modelos de documento. Os modelos de documento (*Document Model*) fazem parte de uma biblioteca (*Model Library*). Cada *Document Model* será composto por itens (*Item*). *Item* é um conceito abstrato sendo materializado como modelos (*Item Model*) ou texto padrão pré-definido (*Default Text*). A Figura 7.3 apresenta os construtores disponíveis em GraphSchema.

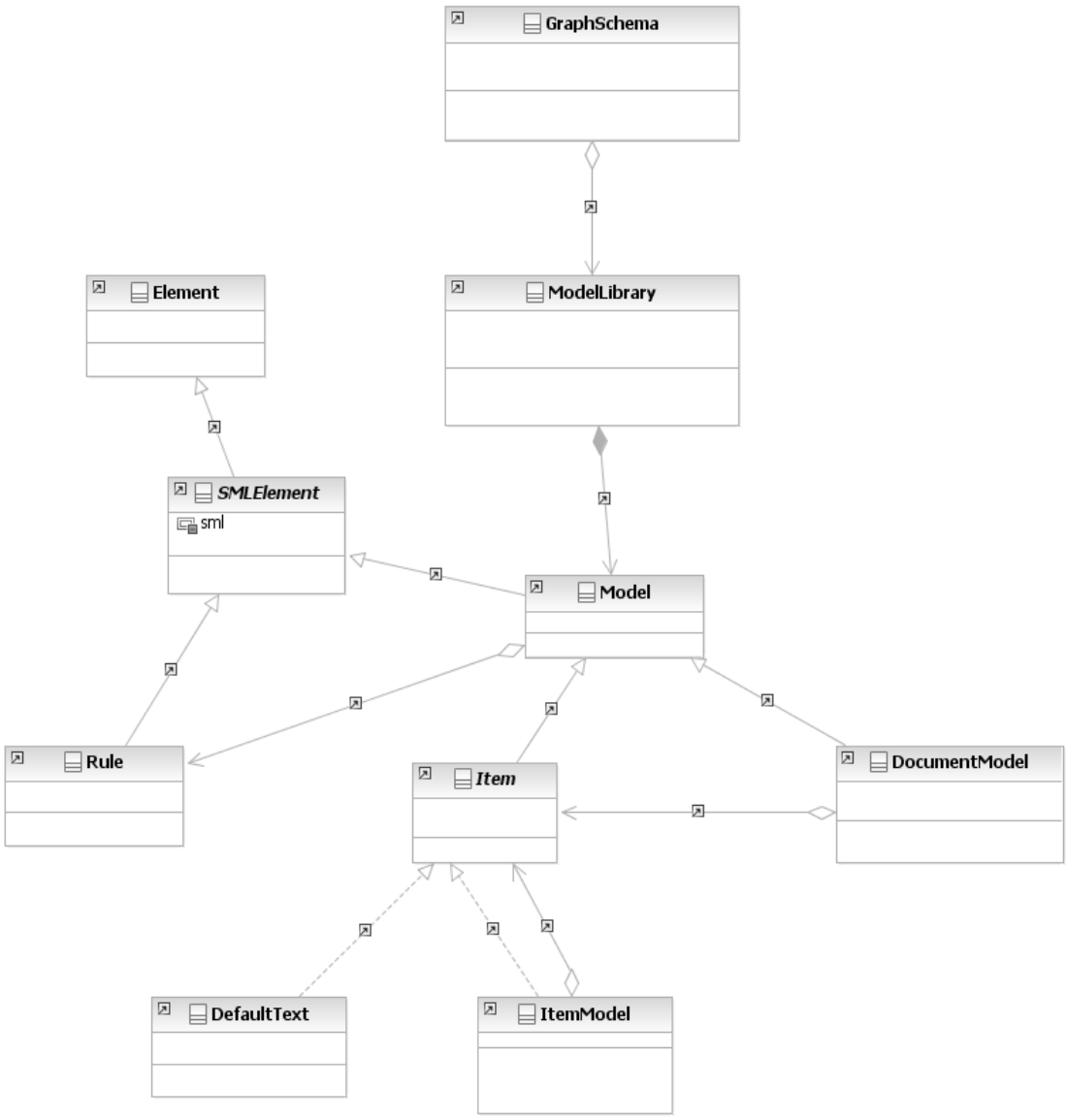


Figura 7.3: Construtores GraphSchema

7.2.2.1Model Library

Model library tem por função agrupar modelos relacionados ao mesmo domínio. Representa visualmente o conceito de *namespace* definido no XML.

Uma *model library* pode incluir modelos completos, *Document Models* e, também, modelos parciais, isto é, *Item Models*.

Os modelos de uma *model library* podem utilizar modelos de outras *librarys*.

7.2.2.2 Document Model

O *Document Model* é o modelo de um documento. Ele conterá todos os itens e regras que descrevem um tipo específico de documento.

Document Model é um SML schema que será usado como base para a criação de documentos propriamente ditos, isto é, será usado como modelo para edição do documento no *micro-level*.

7.2.2.3 Item Model

Item Model define itens componentes de *Document Model* – ele possui um nome e um valor.

O valor pode ser atômico ou composto, criando dois subtipos: *Item Model Atomic* e *Item Model Composite*.

O nome de um *Item model* identifica o item e seu conteúdo, isto é, seu valor. O *Item Model* também pode ser associado a *Rules*.

7.2.2.4 Item Model Atomic

Item Model Atomic é o bloco básico de construção de modelos. Outros componentes do modelo serão criados a partir da agregação dele.

Caracteriza-se por possuir um valor final, ou primitivo, identificado pelo nome do item. Também pode ser associado a *rules* e possuir um valor padrão ou *default*.

O valor de um *Item Model*:

- É um dado primitivo do tipo: texto, número ou data;
- Deve obedecer as regras impostas ao item, incluindo o tipo de dado;
- É informado pelo usuário na criação do texto *(micro-level)*;
- Para sua identificação, o nome pode ser qualificado com o nome da hierarquia de itens na qual ele está inserido;

7.2.2.5 Item Model Composite

Item Model Composite é criado a partir da agregação de *Itens Model Atomic*, e como estes, possui um valor.

O valor de um *Item Model Composite* incluirá um ou mais *Item*, isto é, *Item Model*, *Atomic*, *Composite*, ou *Default Text*, que é um caso especial de *item model atomic*.

Portanto o valor para um item model composto é uma referência a outro item de onde será obtido o dado primitivo.

O valor de um Item Model Composite é uma referência para:

- Outro item do mesmo *Item Model*
- Item de outro *Item Model* do mesmo *Document Model*. Neste caso tem-se uma referência intradocumento.
- Item de outro *Document Model,* da mesma ou de outras *Model Libraries*, tratando-se, neste caso, de uma referência interdocumento.

7.2.2.6 Default Text

Default Text é um *Item Model Atomic* contendo um texto pré-definido no modelo. O valor do texto pode ser marcado como mutável ou imutável.

7.2.2.7 Rules

GraphSchema permite ao usuário expressar declarações condicionais a partir da criação de regras para a validação da estrutura e conteúdo do documento. As *Rules* (regras), ou *constraints*, são mapeadas diretamente em expressões SML/Schematron.

Esta é uma característica que representa grande desafio para o GraphSchema, pois existe a intenção de disponibilizar ao usuário uma representação de fácil entendimento, que o possibilite a total liberdade na criação de regras para os modelos, de modo a refletir as necessidades impostas pelos diferentes tipos de modelos.

Considerando que a razão de utilizar uma representação gráfica é isolar o usuário leigo da representação final das *Rules*, é necessário lidar com a dificuldade intrínseca de definir esta representação gráfica para criação das *Rules*.

Visualmente, o usuário pode identificar Itens que possuem *Rules* pela existência da palavra *"Rules"* no *placeholder* específico na representação gráfica de um item. Também poderá visualizar a existência de linhas que ligam itens que possuem *Rules* definindo seu relacionamento.

Para a criação das regras, o usuário irá clicar no *placeholder* e serão apresentadas as opções existentes. Após selecionar o tipo de regra, será apresentada a lista dos elementos disponíveis, quando for o caso, para que ele selecione o elemento que está sendo relacionado com o que recebeu o "click" do mouse. O sistema então monta o código SML correspondente e desenha as linhas indicando a existêcia de regra. As opções de regras (restrições e relacionamento) são descritas a seguir.

As declarações condicionais podem ser classificadas em três categorias (McCLURE, 2006):

- Obrigações;
- Permissões não Obrigatórias;
- Proibições.

Com a utilização de regras complementando a representação gráfica, em GraphSchema é possível a modelagem de relacionamentos e restrições para um documento.

Para isto, as restrições foram divididas em dois tipos:

- Restrições unitárias aplicadas a um único item;
- Restrições de relacionamento, quando envolver mais de um item.

7.2.2.7.1 Restrições Unitárias

Restrições unitárias aplicam-se individualmente a um item, restringindo aspectos deste item tais como tipo de dado, valor ou quantidade.

Restrição de tipo de dados é feito diretamente na criação do item, onde seu valor é definido, seja primitivo ou composto.

Já os outros tipos de restrições unitárias são sempre expressos através de representação não gráfica.

As restrições unitárias em GraphSchema podem ser de três tipos: Escalar, Conjunto e Intervalo:

Tabela 7.1: Restrições Unitárias

Tipo de Restrição	**Categoria**	**Descrição**	**Modo de Representação**
Escalar	Obrigação	A.Valor é <*OPERADOR*> x	Regra
Conjunto	Obrigação	Valor do item tem que ser um entre um conjunto de valores definido pelo usuário	Regra
Intervalo	Obrigação	Valor do item está no intervalo [n, m], inclusivo	Regra

- A, B são Itens
- A.valor representa o valor primitivo do item
- x é um valor compatível com os tipos primitivos
- n, m são valores numéricos
- <*OPERADOR*> é um dos operadores: =, > , < , >= , <= , !=

As restrições podem ser agrupadas com a utilização de operadores lógicos AND/OR.

7.2.2.7.2 Restrições de Relacionamento

Restrições de relacionamento definem os limites a serem impostos, considerando o relacionamento entre itens do documento.

Para a criação de restrições de relacionamento, pode-se utilizar a representação gráfica diretamente ou associada à definição explícita de regras.

Utilizando a representação gráfica, podem ser definidas restrições de relacionamentos de composição e agregação, i.e., parte-todo. Já para outros tipos de relacionamentos, GraphSchema apresenta ao usuário uma interface para guiá-lo na criação das restrições.

A combinação de representação gráfica associada à criação de regras permite expressar os seguintes relacionamentos:

Tabela 7.2: Restrições de Relacionamento

Tipo de Restrição	Categoria	Descrição	Modo de representação
Composição	Obrigação	A é composto por n até m B	Gráfico
Agregação	Permissão	A pode incluir n até m B	Gráfico
Coexistência Restrita	Obrigação	Se A e B estão no modelo então A.valor<*OPERADOR*>B.valor	Regra
Coexistência Obrigatória	Obrigação	Se modelo tem A então tem que ter n a m B	Regra
Coexistência Opcional	Permissão	Se modelo tem A então pode ter n a m B	Regra
Coexistência Proibitiva	Obrigação	Se modelo tem A então não pode ter B	Regra
Existência Exclusiva	Obrigação	Modelo tem que ter A ou B	Regra

- A, B são Itens
- A.valor representa o valor primitivo do item
- n, m são valores numéricos
- <*OPERADOR*> é um dos operadores: =, > , < , >= , <= , !=

Na representação gráfica, as regras são apresentadas como parte de um elemento. Uma regra em GraphSchema não é diretamente apresentada como gráfico, apenas indiretamente.

7.3 Exemplo de Aplicação

Para demonstrar a modeladem de um documento, será apresentado um contrato de compra/venda de imóvel.

Cabe resaltar que o exemplo apresentado não é um documento real, isto é, um contrato completo. Isto porque o texto apresentado foi simplificado de modo a diminuir a complexidade do modelo e facilitar o entendimento do mesmo.

Na Figura 7.4 é apresentando o modelo de um contrato simplificado de compra e venda de imóvel. O contrato é apresentando primeiramente no modo gráfico, isto é, representado na linguagem gráfica GraphSchema. Depois é mostrado a representação SML correspondende a este modelo. Finalmente, na Figura 7.6 é apresentado o texto de um contrato formatado de acordo com o modelo.

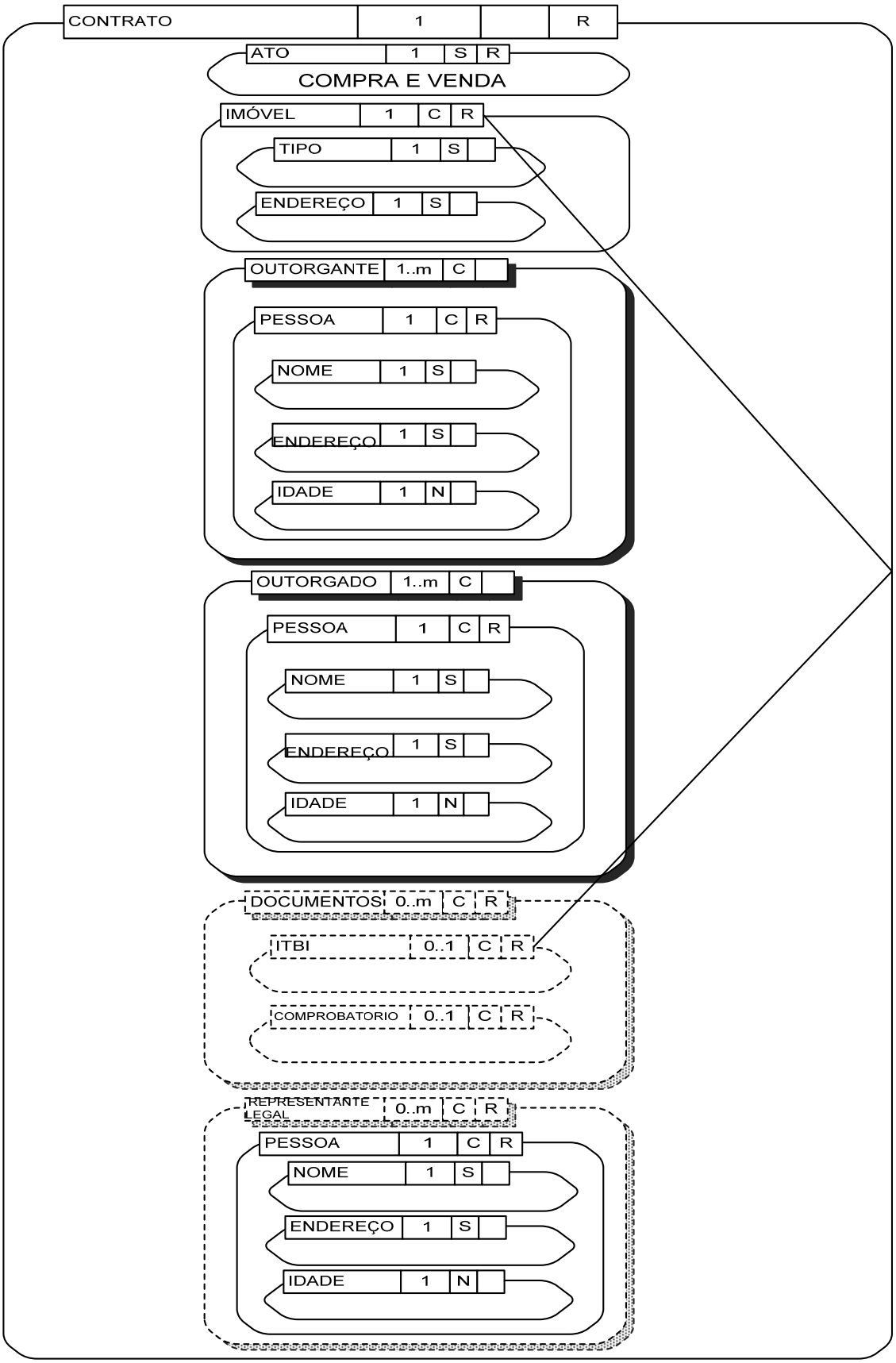


Figura 7.4: GraphSchema Para Um Contrato de Compra e Venda de Imóveis

No modelo estão presentes os seguintes elementos:

- Contrato: Document Model composto por Itens;
- Ato: Item Atomic, obrigatório, único (cardinalidade 1);
- Imovel: Item composite, obrigatório, único (cardinalidade 1), composto por dois Item Atomic um Endereço e Tipo, ambos obrigatórios;
- Endereço: Item Atomic, único (cardinalidade 1), obrigatorio, tipo Texto;
- Tipo: Item Atomic, único(cardinalidade 1), obrigatorio, tipo Texto;
- Outorgante: Item composite múltiplo (cardinalidade 1 a n), obrigatorio, composto por uma ou mais Pessoas;
- Outorgado: Item composite múltiplo (cardinalidade 1 a n), obrigatorio, composto por uma ou mais Pessoas;
- Representante Legal Item composite (cardinalidade 1 a n), opcional. múltiplo, composto por uma ou mais Pessoas;
- Pessoa Item composite, (cardinalidade 1), composto por Nome, Endereço e Idade, todos obrigatórios e com cardinalidade 1;
- Nome: Item Atomic tipo Texto;
- Idade: Item Atomic tipo Numero;
- Documentos: Item Composite opcional multiplo (cardinalidade 0 a n);
- ITBI: Item Atomic opcional multiplo (cardinalidade 0 a n);
- Comprobatório: Item Atomic opcional multiplo (cardinalidade 0 a n).

Além dos elementos identificados acima, também cabe resaltar a linha continua ligando Imovel a Documento/ITBI. A linha continua, conforme especificação do GraphSchema, representa uma regra associando estes dois Itens. Esta regra declara a obrigatoriedade de ITBI associada ao documento, isto é, dado que Imóvel é obrigatório deve ser apresentado o documento de pagamento do ITBI (imposto transmissão de bens imobiliarios). A representação da lógica desta regra é apresentada em SML, conforme código apresentado na Figura 7.5.

A representação SML, parcial, do modelo é:

```
<!-- .... codigo incompleto .... -->
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xmlns:sml="http://schemas.serviceml.org/sml/2006/07"
    xmlns:sch="http://purl.oclc.org/dsdl/schematron" >
<xsd:element  name="CONTRATO">
    <xs:complexType>
        <xs:sequence> <!-- a ordem dos elemento é obrigatoriamente esta -->
            <xs:element name="IMOVEL" minOccurs="1" maxOccurs="1"> <!-- so podemos ter um imovel no contrato -->
                <xs:complexType>
                    <xs:attribute name="id" type:="xs:string" />
                    <xs:complexType>
                        <xs:sequence> <!-- a ordem dos elemento é obrigatoriamente esta -->
                            <xs:element ref="tipo" minOccurs="1"  maxOccurs="1"/>
                            <xs:element ref="end" minOccurs="1"  maxOccurs="1"/>
                        </xs:sequence>
                    </xs:complexType>
                <xs:complexType>
            </xs:element>
            <!-- .... nao são mostrados os outros itens... -->
            <xs:element name="DOCUMENTOS" minOccurs="0" maxOccurs="unbounded"> <!-- documentos é opcional -->
                <xs:complexType>
                    <xs:attribute name="id" type:="xs:string" />
                    <xs:element ref="ITBI" minOccurs="0"  maxOccurs="unbounded"/>
                    <!-- EM XMLSchema, não teria como dizer que este documento é obrigarorio
                    no caso de ter imóvel, pois aqui é dito que ele pode ter 0 ou mais -->
                    <xs:element ref="outro_doc" minOccurs="0"  maxOccurs="unbounded"/>
                </xs:complexType>
            </xs:element>
        <xs:annotation>
            <xs:appinfo>
                <sch:schema> <!-- aqui a regra que associa o imovel ao documento de ITBI,
                forçando a existencia dele quando tem imóvel não é possível fazer isto somente com XMLSchema -->
                    <sch:title>ITBI Obrigatorio Para Imóvel</sch:title>
                    <sch:pattern name="Itbi Obrigatorio Para Imóvel" id="ItbiObrigatorioParaImóvel">
                        <sch:rule context="/sp:CONTRATO">
                            <sch:let name="TemImovel" value="count('/CONTRATO/IMOVEL/')" />
                            <sch:let name="TemItbi" value="count('/DOCUMENTOS/ITBI/')" />
                            <!-- um teste mais completo deveria ver se o documento corresponde ao imóvel -->
                            <sch:assert test="TemImovel > 0 AND TemItbi < 1" >
                                Imóvel exige apresentação do ITBI.
                            </sch:assert>
                        </sch:rule>
                    </sch:pattern>
                </sch:schema>
            </xs:appinfo>
        </xs:annotation>
    </xs:complexType>
</xs:element>
```

Figura 7.5: Código SML coorespondende ao Contrato de Compra/Venda de Imóveis

E finalmente, o texto de um contrato de compra e venda correspondente ao modelo apresentado na Figura 7.4 é o seguinte:

ATO

Disseram os outorgantes que vendem às outorgadas o imóvel descrito a seguir pelo preço de R$100.000,00 (Cem mil reais), com pagamento em dinheiro em uma única parcela. Dão às compradoras plena, geral e irrevogável quitação; transmitem todo o domínio, direito, ação e posse que tinham no imóvel, obrigando-se pela evicção.

IMÓVEIS

Apartamento localizado na Rua Borges de Medeiros, número 24 apto 2**.**

OUTORGANTES

JOSE DREITAS NITZ, 33 anos, residente e domiciliado nesta cidade Avenida Carazinho, 2040, 53.

ANDRE VASQUES VIANNA, 34 anos, residente e domiciliado nesta cidade Av. Baltazar de Oliveira Garcia, 3850, apto. 3.

OUTORGADOS

ANGELA NITZ, 28 anos, residente e domiciliado nesta cidade Rua Luis de Campos, 2801.

JOANA MARTINS DA ROCHA, 21 anos, residente e domiciliada na cidade de Estância Nova/RS, na Rua João XXIII nº 28.

DOCUMENTOS APRESENTADOS

Comprovante de pagamento do ITBI relativo ao imóvel acima citado, sob protocolo número 12345 emitido em 23/03/2008.

Figura 7.6: Texto de Contrato de Compra/Venda de Imóvel

O texto do contrato apresentando é um contrato válido de acordo com o modelo GraphSchema apresentado anteriormente.

8 IMPLEMENTAÇÃO

Para demostrar a possibilidade de geração de SML a partir de uma representação gráfica, foi implementado um protótipo parcial que permite ao usuário a criação de modelos utilizando o mouse e elementos gráficos pré-definidos.

8.1 Projeto

8.1.1 Ferramentas Utilizadas

Para o desenvolvimento do protótipo foi escolhida a linguagem Python (www.python.org) e utilizada duas bibliotecas para auxiliar na criação da interface gráfica e geração do SML: wxPython e ElementTree.

A apresenta o modelo completo de objetos utilizado no protótipo. As classes marcadas com uma cruz não foram alteradas, sendo utilizadas na sua forma original da biblioteca de origem.

As bibliotecas utilizadas e alterações relevantes realizadas nas mesmas estão descritas nos itens a seguir.

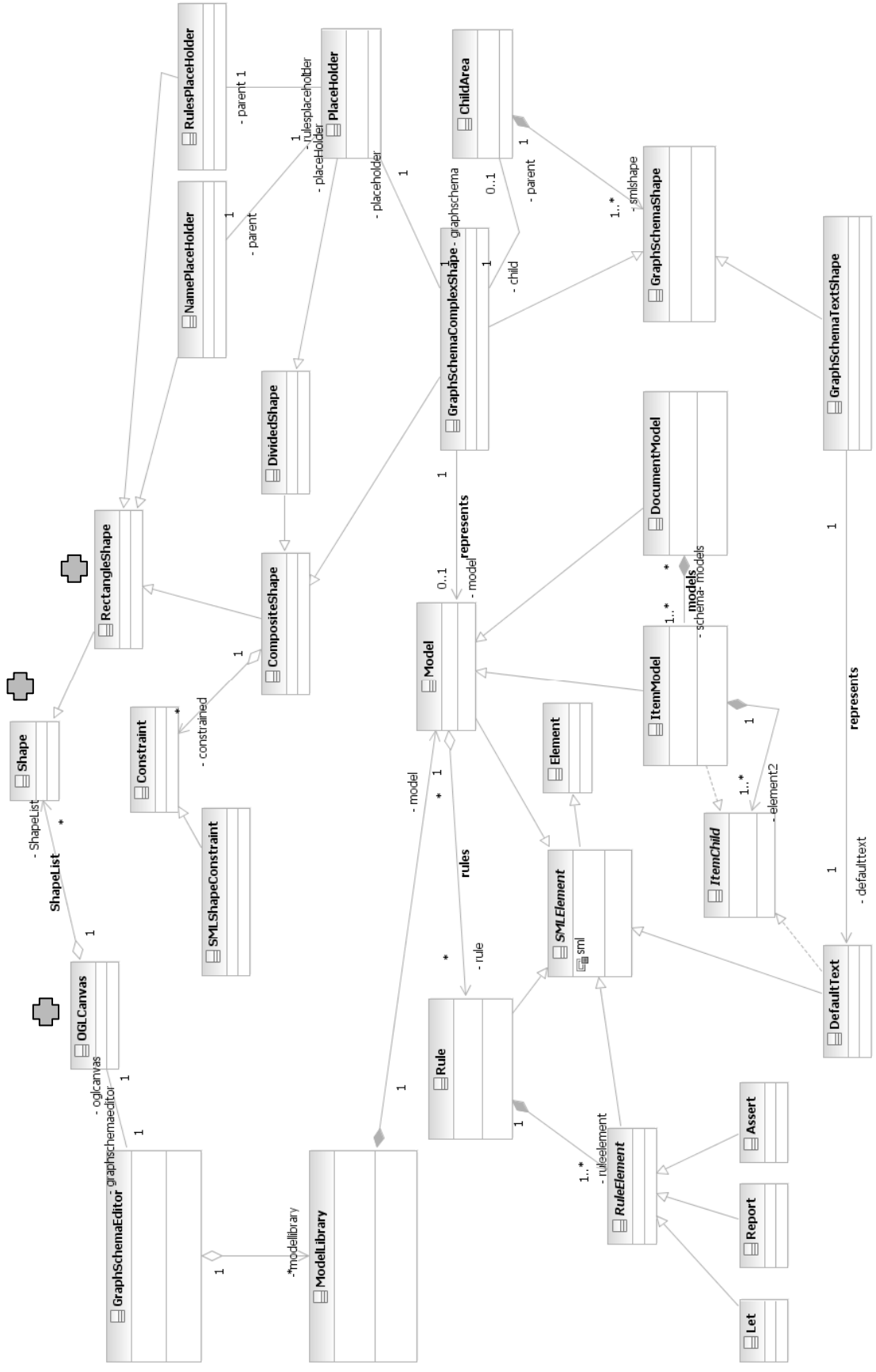


Figura 8.1: Modelo Completo das Classes Implementadas no GraphSchema

8.1.1.1 Biblioteca Para Interface Gráfica

Para a criação da interface com o usúario foi utilizada a biblioteca wxPython. wxPython (http://www.wxpython.org/) é uma *toolkit* para construção de GUI multiplataforma na linguagem Python. É implementado como um modelo de extensão Python, que encapsula a popular biblioteca multiplataforma wxWidgets, escrita em C++ e disponível para Windows, Linux, Unix, Mac OS entre outros.

A vantagem da utilização do wxPython, além da liberdade de execução multiplataforma, está na grande disponibilidade de componentes que podem ser usados para a criação da interface. No caso específico do GraphSchema, o módulo Object Graphics Library (OGL) foi o ponto de partida para a criação dos objetos, representando os elementos do GraphSchema, tendo sido alterado para atender as necessidades específicas do GraphSchema.

OGL utiliza o padrão *composite* para a representação de objetos gráficos. A base de todos os objetos é a classe *Shape*. Para o GraphSchema, foram modificados os módulos de *shapes* "*divided*" e "*composite*".

Para criar os cabeçalhos dos elementos (nome, cardinalidade, tipo, *placeholder* de regras), a classe *divided* foi alterada para incluir a criação de objetos com divisão vertical, já que a classe original somente disponibilizava divisão horizontal.

Esta classe sofreu grande alteração, praticamente recriada, passando de 400 para 600 linhas de código. Todos os métodos que apresentam HORIZONTAL_DIVISION foram criados para o GraphSchema.

```
class DividedShape(RectangleShape):
    """A DividedShape is a rectangle with a number of vertical or horizontal divisions.
    Each division may have its text formatted with independent characteristics,
    and the size of each division relative to the whole image may be specified.

    Derived from:
      RectangleShape
    """
    def __init__(self, w, h, divisionType=HORIZONTAL_DIVISION):
        RectangleShape.__init__(self, w, h)
        self.ClearRegions()
        self.divisionType = divisionType

    def OnDraw(self, dc):
        RectangleShape.OnDraw(self, dc)

    def OnDrawContents(self, dc):
        defaultProportion = self.DefaultProportion()

        if self.divisionType == HORIZONTAL_DIVISION:
            self.OnDrawContentsHorizontal(dc, defaultProportion)
        elif self.divisionType == VERTICAL_DIVISION:
            self.OnDrawContentsVertical(dc, defaultProportion)

    def HorizontalLimits(self):
        currentY = self._ypos - self._height / 2.0
        maxY = self._ypos + self._height / 2.0

        leftX = self._xpos - self._width / 2.0
        rightX = self._xpos + self._width / 2.0

        return currentY , maxY, leftX, rightX

    def OnDrawContentsHorizontal(self, dc, defaultProportion):
        currentY , maxY, leftX, rightX = self.HorizontalLimits()

        if self._pen:
            dc.SetPen(self._pen)

        dc.SetTextForeground(self._textColour)

        # For efficiency, don't do this under X - doesn't make
        # any visible difference for our purposes.
        if sys.platform[:3] == "win":
            dc.SetTextBackground(self._brush.GetColour())

        if self.GetDisableLabel():
            return

        xMargin = 2
        yMargin = 2

        dc.SetBackgroundMode(wx.TRANSPARENT)

        for region in self.GetRegions():
            dc.SetFont(region.GetFont())
            dc.SetTextForeground(region.GetActualColourObject())

            if region._regionProportionY < 0:
                proportion = defaultProportion
            else:
                proportion = region._regionProportionY

            y = currentY + self._height * proportion
            actualY = min(maxY, y)

            centreX = self._xpos
            centreY = currentY + (actualY - currentY) / 2.0

            DrawFormattedText(dc, region._formattedText, centreX, centreY, self._width -
2 * xMargin, actualY - currentY - 2 * yMargin, region._formatMode)

            if y <= maxY and region != self.GetRegions()[-1]:
                regionPen = region.GetActualPen()
                if regionPen:
                    dc.SetPen(regionPen)
                    dc.DrawLine(leftX, y, rightX, y)

            currentY = actualY
```

Os elementos do GraphSchema são *CompositeShapes*, isto é, foram construídos a partir de *shapes* primitivas disponíveis. O posicionamento dos objetos de uma *CompositeShapes* é controlado pela classe *Constraint*. Desta forma, ao mover uma *shape* de mais alto nível, a *constraint* se encarrega de redesenhar os filhos da *shape* composta.

Para as necessidades do GraphSchema , foram utilizadas *shapes* Retangulos. Mas foi necessária a alteração da classe *Constraint* para contemplar o controle de formatação seqüencial necessário para "encaixar" os elementos do GraphSchema corretamente. Neste caso foi criada uma *constraint* que organiza os subelementos em um modo seqüencial de linhas e colunas, que foi denominada CONSTRAINT_SEQUENCIAL_FLOW.

Para desenhar os elementos filhos, o funcionamento da *Constraint* CONSTRAINT_SEQUENCIAL_FLOW controla o posicionamento e redesenho dos elementos filhos permitindo, ou não, operações de modificação de tamanho do elemento (*resize*) ou posição dos filhos (*drag*). O estágio atual da implementação restringe as possibilidades para permitir o desenvolvimento do software em tempo adequado. Mais trabalho deve ser feito neste controle no futuro.

De acordo com o tipo de elemento, o funcionamento do CONSTRAINT_SEQUENCIAL_FLOW desenvolve-se da seguinte maneira:

1. Document Model:

Permite fazer o redimencionamento tanto horizontal como vertical e redesenha os filhos em ordem seqüencial. Isto é, vai posicionando os filhos em linhas, de acordo com a ordem em que aparecem, pulando para a próxima linha, se a linha corrente está cheia, ou se o filho não couber no espaço existente. O tamanho do elemento é redimensionado na vertical para acomodar todos os filhos, mas não na horizontal.

2. Item Model:

O desenho do *item model* depende da dimensão do elemento pai, já que normalmente um *item model* está embutido em outro item, seja ele um *document model* ou outro item. O algoritmo permite fazer redimencionamento aumentado no tamanho máximo horizontal do pai, isto é, limita a largura do filho à largura do pai. O redimensionamento na vertical não é limitado sendo que, se o filho for aumentado mais que o tamanho do pai, o algoritmo de *constraint* enviará mensagens para que o pai aumente seu tamanho vertical na mesma proporção.

3.Default Text:

A *constraint* para *default text* limita o texto à largura ao pai, isto é, o usuário vai digitando até chegar à linha delimitadora horizontal do elemento (à direita), então é realizada a quebra de linha. No caso de chegar ao limite vertical (linha na parte inferior do elemento), o algoritmo aumenta o pai automaticamente para acomodar o texto.

4. Rules:

O algoritmo de *constraint* desenha somente as linhas ligando os elementos relacionados através de regras, se houver.

O código (parcial) da implementação da *constraint* é listado a seguir:

```
    def Evaluate(self):
        """Evaluate this constraint and return TRUE if anything changed."""
        maxWidth, maxHeight = self._constrainingObject.GetBoundingBoxMax()
        minWidth, minHeight = self._constrainingObject.GetBoundingBoxMin()
        x = self._constrainingObject.GetX()
        y = self._constrainingObject.GetY()

        dc = wx.ClientDC(self._constrainingObject.GetCanvas())
        self._constrainingObject.GetCanvas().PrepareDC(dc)

        if self._constraintType == CONSTRAINT_SEQUENCIAL_FLOW :
            prevWidth = 0
            prevHeight = 0
            prevX = x - minWidth / 2.0
            topMargin = 10
            prevY = (y - minHeight / 2.0) + topMargin
            newY = y
            parentLeftEdge = x - (self._constrainingObject.GetWidth() / 2)
            parentRightEdge = x + (self._constrainingObject.GetWidth() / 2)
            parentBottonEdge = y + (self._constrainingObject.GetHeight() / 2)


            for constrainedObject in self._constrainedObjects:

                newX = prevX + (prevWidth / 2.0) + (constrainedObject.GetWidth() / 2)
                newY = prevY #+ (prevHeight / 2.0) + (constrainedObject.GetHeight() / 2)
                rightEdge = newX + (constrainedObject.GetWidth() / 2)
                bottonEdge = newY + (constrainedObject.GetHeight() / 2)

                if rightEdge > parentRightEdge:
                    newY = prevY + (prevHeight / 2.0) + (constrainedObject.GetHeight() /
2)
                    newX = parentLeftEdge +  (constrainedObject.GetWidth() / 2)
                    rightEdge = newX + (constrainedObject.GetWidth() / 2)
                    bottonEdge = newY + (constrainedObject.GetHeight() / 2)
                    if bottonEdge > parentBottonEdge:

self._constrainingObject.SetHeight(self._constrainingObject.GetHeight()+constrainedObjec
t.GetHeight())

                constrainedObject.Move(dc, newX, newY, False)
                prevWidth = constrainedObject.GetWidth()
                prevHeight = constrainedObject.GetHeight()
                prevX = newX #x - (prevWidth / 2.0)
                prevY = newY #y - (prevHeight / 2.0)

            return True

        elif self._constraintType == CONSTRAINT_CENTRED_VERTICALLY:
        ..... continua codigo original ....
```

8.1.1.2 Biblioteca para Manipulação de Árvore de Objetos

Já para auxiliar a geração e tratamento do SML, foi utilizada a biblioteca ElementTree (http://effbot.org/zone/element-index.htm), que realiza tratamento de contêiner de objetos em forma hierárquica de árvore e possui facilidades para lidar com elementos XML, tanto na criação da estrutura lógica como na geração do texto do XML. Element Tree disponibiliza módulos para a construção de estruturas em árvore e facilita a criação de elementos XML.

ElementTree é bastante eficiente e direta no tratamento dos objetos em árvore e torna a criação de XML uma tarefa relativamente simples, por exemplo, o código listado a seguir:

```
        self._element = Element("complexType", name = self._name)
        sequence = ElementTreeSubElement(self._element, "sequence")
```

gera o seguinte XML:

```
<xs:complexType>
    <xs:sequence>
    </xs:sequence>
</xs:complexType>
```

Como o objetivo deste protótipo é a criação de modelos e não sua interpretação, não foi utilizado um *parser* e validador SML nesta etapa do projeto.

8.2 Modelo de Objetos

A geração do SML é feita por elementos derivados da classe SML. Conforme pode ser visto no diagrama de classe na Figura 7.3, para cada tipo de elemento do GraphSchema corresponde um elemento SML, que irá conter a especificação do SML a ser gerado para aquele elemento.

Por exemplo, o código a seguir mostra a geração de SML para *ItemModel* e DefaultText.

```
class ItemModel(Item):
    def __init__(self, canvas, name, parent = None):
        self._name = name
        #self._gs_type = "ItemModel"
        self._parent = parent
        self._element = Element("complexType", name = self._name)
        sequence = ElementTreeSubElement(self._element, "sequence")

        self._represented_by = GraphSchema.GraphSchemaComplexShape(canvas,
self._name)

class DefaultText(Item):
    def __init__(self, canvas, name, parent = None):
        self._name = name
        #self._gs_type = "DefaultText" ,
        self._parent = parent
        self._element = Element("element",   name = self._name)

        self._represented_by = GraphSchema.GraphSchemaTextShape(self, canvas,
self._name)
```

Examinando o código, percebe-se a ligação entre o objeto SML e a sua representação gráfica. Por exemplo, um *item model* é representado graficamente por um GraphSchemaComplexShape:

```
self._represented_by = GraphSchema.GraphSchemaComplexShape(canvas, self._name)
```

Por sua vez, GraphSchemaComplexShape é implementado com o código apresentado a seguir.

```
class GraphSchemaComplexShape(ogl.CompositeShape, GraphSchemaShape):
    def __init__(self, canvas, name, childAreaWidth = DEFAULT_childAreaWidth
,
                                       childAreaHeight =
DEFAULT_childAreaHeight):
        super(GraphSchemaComplexShape,self).__init__()
        self.SetCanvas(canvas)
        self.SetId(name)
        #print "canvas %s" % canvas.__class__.__name__
        # calculate width for placeholder
        dc = wx.ClientDC(canvas)
        placeHolderMinWidth = dc.GetTextExtent(name)[0] + 20

        if placeHolderMinWidth   > DEFAULT_childAreaWidth:
            placeHolderMinWidth  = DEFAULT_childAreaWidth

        self._childArea = GraphSchemaChildAreaShape( canvas, name,
DEFAULT_childAreaWidth, DEFAULT_childAreaHeight )
        self._PlaceHolder = GraphSchemaPlaceHolderShape(canvas, name ,
placeHolderMinWidth , 20 )

        self._childArea.SetBrush(wx.WHITE_BRUSH)
        self._PlaceHolder.SetBrush(wx.WHITE_BRUSH)

        self.AddChild(self._childArea)
        self.AddChild(self._PlaceHolder)

        constraint = GraphSchemaConstraint(ogl.CONSTRAINT_MIDALIGNED_TOP,
self._childArea, [self._PlaceHolder])
        self.AddConstraint(constraint)
        self.Recompute()

        # If we don't do this, the shapes will be able to move on their
        # own, instead of moving the composite
        self._childArea.SetDraggable(False)
        self._PlaceHolder.SetDraggable(False)

        # If we don't do this the shape will take all left-clicks for itself
        #childArea.SetSensitivityFilter(0)
        self._PlaceHolder.SetSensitivityFilter(0)

    def AddChildItem(self, child, addAfter = None):
        """Adds a child item"""
        #child.SetWidth()
        #child.SetHeight()
        self._childArea.AddChild(child, addAfter)
```

9 CONCLUSÃO

Documentos de texto é um formato onipresente no armazenamento de informações. Estes, apesar de serem classificados como informação não estruturada, possuem estruturas semânticas implícitas que, se fossem preservadas, seriam de fundamental importância para a manutenção das memórias organizacionais em conformidade com o conceito de ”Enriquecimento de Documentos”.

Em especial, documentos de textos legais e comerciais, tais como, documentos jurídicos, contratos, acordos e outros, possuem regras que são de conhecimentos de especialistas do domínio ao qual se enquadra o documento. Apesar de este trabalho estar focado em contratos, praticamente todos os documentos citados são compostos a partir de itens (partes), de texto padrão, intercambiáveis e que possuem regras determinadas por lei, que orientam a sua combinação para criação do documento final. Em especial, no Brasil, há uma legislação complexa para criação de documentos jurídicos, mas esta situação é idêntica em praticamente todos os países e, em particular, em países com notariado latino, tais como, Portugal e Espanha. Este cenário aumenta a relevância da pesquisa realizada neste trabalho, pois sua aplicação possui o escopo central de suprir a necessidade de constantes elaborações de contratos nos sistemas jurídicos, hordiernamente, ainda embasados no Direito Romano.

A criação de documentos de texto, ou contratos, em muitos casos, não é realizada por estes especialistas, pois geralmente estes são raros ou demasiadamente comprometidos com outras atividades. Dessa forma, a tarefa de elaboração de contratos é delegada a profissionais menos experientes, cabendo ao especialista apenas a revisão dos contratos gerados. Isto cria um gargalo de produtividade para as entidades que geram tais documentos. Porém, mesmo se os especialistas criassem os contratos, estes não possuem conhecimento que os capacite a fazer uma modelagem da estrutura do documento usando as linguagens de estruturação de texto e/ou representação de ontologias, pois este conhecimento é restrito a alguns poucos profissionais da ciência da computação.

Este problema impede a disseminação em larga escala do uso de técnicas de construção automática de contratos *(drafting)* e também impede a geração de documentos enriquecidos com metadados.

O cenário apresentado acima exige uma solução que envolva a disponibilidade de um editor de textos, capaz de guiar usuários na criação dos contratos em uma técnica conhecida como *contract drafting*. Isto exige que os possíveis tipos de contrato sejam

previamente modelados, permitindo então a estruturação de documentos de texto e seu enriquecimento semântico.

A questão central desta pesquisa encontra-se no fato que o usuário de editor de texto é aquele que necessita mais da modelagem, mas não consegue desenvolvê-la, pois, como foi abordado, modelagem conceitual é difícil para leigos, além do domínio de linguagens de representação de conhecimento e estruturação de textos (XML).

A solução que foi denominada GraphSchema, separa a criação do texto da modelagem conceitual dos tipos de contratos, sendo esta última realizada por um especialista e preservada para uso dos usuários não especialistas, servindo, posteriormente, de guia para criação dos documentos.

GraphSchema consiste em uma ferramenta para usuário final realizar a criação de itens e modelagem do domínio do texto utilizando uma representação gráfica da linguagem SML. Os modelos criados serão posteriormente utilizados por um editor guiado por SML para criação de documentos marcados. Devido a sua simplicidade, a utilização de SML é apresentada como uma alternativa a outras linguagens de descrição de ontologia.

9.1 Principais Contribuições

GraphSchema inova por focar-se na geração de modelos de contratos e de documentos de texto diretamente pelos usuários finais. Para isto, preserva a visão de documento que o usuário possui e isola o mesmo da linguagem utilizada para a representação interna do modelo.

Ao visualizar o modelo gráfico e utilizando o paradigma “arrastar/soltar”, intuitivamente o usuário consegue realizar a geração de modelos de documentos e a modelagem de relacionamentos entre itens com a utilização de regras complexas determinadas a seu critério. Ao introduzir o conceito de *Model Library*, o GraphSchema também possibilita a criação de modelos de itens isolados e que posteriormente poderão ser usados para composição de outros modelos, permitindo a reutilização de trabalhos anteriores e, conseqüentemente, aumentando a produtividade na geração de modelos.

Ao propor a utilização da SML como suporte para a definição do modelo conceitual de documentos de texto, GraphSchema reduz a complexidade do formato final do modelo, já que está linguagem é uma extensão do XMLSchema que utiliza Schematron para a especificação das regras sendo, portanto, de amplo conhecimento dos profissionais de informática – ao contrário das linguagens de descrição de ontologia baseadas em formalismos lógicos mais complexas, e menos conhecidas.

Por ser embasada no XMLSchema, a construção de ferramentas para tratamento dos modelos pode beneficiar-se da ampla gama de ferramentas e bibliotecas de software, disponíveis para o tratamento de XML e Schematron, o que muitas vezes não é possível para outras linguagens de descrição de ontologias.

Ao introduzir uma técnica para a criação de modelos de documentos sem exigir um profissional com conhecimento de linguagens de marcação de texto, i.e., usuários finais, possibilita a disseminação e ampla utilização de documentos estruturados, em múltiplos cenários:

- Servir como implementação de referência para o padrão eContracts, permitindo a um usuário leigo criar modelos de contratos em conformidade com este padrão, sem a necessidade de entrar nos detalhes técnicos;
- Viabilizar a edição de documentos marcados, *a priori,* com o uso de editores guiados por XML;
- Para criação automática de documentos usando XSLT, ou alguma técnica equivalente;
- No contexto de *"Document Enrichment"*, ou enriquecimento de documentos (MOTTA et al., 2000) e *"Indexation Conceptuelle"*, ou Indexação Conceitual (PRIÉ, 2000);
- O editor de texto transforma-se em gerador de dados de entrada para bancos de dados de informação não estruturada e integra-se diretamente à criação de memórias organizacionais.

Em suma, o GraphSchema servirá de base de comparação para trabalhos futuros que utilizem linguagens mais complexas que SML para a geração de ontologias diretamente por usuários finais.

9.1.1 Principais Dificuldades Encontradas na Implementação do Software

9.1.1.1 Identificação de elementos

A identificação dos elementos é sem dúvida um problema a ser melhor estudado. Isto deve-se ao fato de ter sido originalmente projetada para usar o nome do elemento, porém como temos a possibilidade de ter o mesmo elemento repetido no documento, isto é, mais de um item do mesmo tipo, no protótipo foi utilizado a geração automática de um identificador numérico único para cada item. E este aspecto deve ser considerado na montagem final do SML, tanto nas referências como na criação de regras.

9.1.1.2 Implementação

Durante o desenvolvimento do protótipo ficou claro que a definição e desenvolvimento de um software que apresente uma interface gráfica para o usuário demandará muito mais esforço que o originalmente previsto. Isto está associado às dificuldades intrínsecas da criação de uma ferramenta de manipulação gráfica que permita a definição de schemas SML diretamente por usuários finais.

9.2 Trabalhos futuros

A pesquisa na área de modeladem de documentos de texto legais, e geração automática de documentos de texto e contratos, é de fundamental importância para a automação das atividades jurídicas. Esta importante área de atividade humana, apesar de ser alvo de projetos desde os primórdios da edição de texto computadorizada, ainda encontra-se muito distante da realidade de outras áreas, como, por exemplo, na engenharia, onde a utilização de ferramentas de CAD não só é uma realidade, como se tornou indispensável para a competitividade das empresas.

Neste sentido, GraphSchema aborda uma questão fundamental para a disseminação do uso de modelos conceituais de documentos com os benefícios advindos desta

tecnologia, como, por exemplo, a geração automática de textos legais válidos e com marcação *a priori*, viabilizando a utilização de padrões existentes e, ao mesmo tempo, permitindo que o usuário assuma o controle de como estes padrões podem ser aplicados a sua realidade.

Mesmo concentrando-se na geração de modelos de contratos, GraphSchema aborda um tema tão amplo que ultrapassa o escopo de um trabalho de mestrado. O desenvolvimento de um software que permita ao usuário criar modelos através de uma linguagem gráfica, por si só, já é uma tarefa complexa. No entanto, a criação de modelos é apenas o passo inicial para a geração de documentos de texto legais. A utilização destes modelos em um editor de texto guidado por SML seria o próximo passo necessário para validar na prática a proposta do GraphSchema.

Especificamente em relação ao editor de modelos, ainda existem oportunidades de melhorias na configuração do editor, principalmente na possibilidade de configuração de regras não previstas nas situações apresentadas no projeto, isto é, a possibilidade de inserir novos tipos de regras unitárias, ou de relacionamento, sem a necessidade de modificação do código do editor ou, melhor ainda, que isto possa ser feito diretamente pelo usuário final, aumentando a independência deste usuário, o que é a base da proposta do GraphSchema.

A implementação do editor de modelos ainda encontra-se em uma fase inicial, sendo basicamente um protótipo que serve como prova de conceito e não adequado para seu público alvo, principalmente devido a sua meta de ser uma ferramenta voltada para usuários. Isto obriga que este software apresente um grau elevado de usabilidade. Portanto, a conclusão desta implementação e a sua otimização também é uma meta a ser perseguida.

Para complementar o estudo, também se faz necessária a avaliação prática das potencialidades do GraphSchema, isto é, testes com usuários e casos reais. Para isto, é necessária a existência de um editor de texto guiado por SML, para que possam ser realizadas avaliações completas de geração automática de contratos e não somente da criação de modelos.

Entre outras oportunidades de pesquisa, destaca-se a comparação de modelos SML de contratos, ou mais abrangentemente, de documentos de texto, com modelos representados em outras linguagens. Finalmente, ainda pode ser objeto de pesquisas futuras a avaliação da utilização da linguagem gráfica do GraphSchema, não somente para gerar modelos em SML, mas também para a geração de modelos em outras linguagens de descrição de esquemas e ontologias.

REFERÊNCIAS

ALLEN, L. E.; SAXON, C. S. Some Problems in Designing Expert Systems to Aid Legal Reasoning. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 1., 1987. **Proceedings…** New York: ACM, 1987. p. 94–103.

ARNOLD-MOORE, T. **Information Systems for Legislation.** 1998. Ph.D. Dissertation, Royal Melbourne Institute of Technology, Australia.

BAADER, F. et al. **The Description logic handbook** : theory, implementation, and applications. Cambridge: Cambridge University Press, 2004.

BARR, A.; FEIGENBAUM, E. A. (Ed.). **The Handbook of Artificial Inteligence**. Reading: Addison-Wesley, 1981. v. 1, p 141-222.

BATRES, E. J. Q. et al. Uso de Ontologias para a Extração de Informações em Atos Jurídicos em uma Instituição Pública. **Encontros Bibli: Revista Eletrônica de Biblioteconomia e Ciência da Informação**, Florianópolis, v. 19, n. 1, p. 73-88, 2005.

BELLORD, N. J. Information and Artificial Intelligence in the Lawyer's Office. In: CIAMPI, C. (Ed.). **Artificial Intelligence and Legal Information Systems**. Amsterdam: North-Holland Publishers, 1982. p. 241–249.

BIAGIOLI, C. et al. NIREditor, A XML Specific Environment for Legislative Drafting (Norme in Rete project). In: INTERNATIONAL WORKSHOP ON THE DEVELOPMENT OF STANDARDS FOR DESCRIBING LEGAL DOCUMENTS, 2003, Utrecht, The Netherlands. **Proceedings…** Disponível em: <http://www.lri.jur.uva.nl/standards2003/>. Acesso em: out. 2005.

BIAGIOLI, C. et al. A Legal Drafting Environment Based on Formal and Semantic XML Standards. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, ICAL, 10., 2005, Bologna. **Proceedings…** [S.l.: s.n.], 2005. p. 244-245.

BOER, A. et al. MetaLex: An XML Standard for Legal Documents. In: XML EUROPE CONFERENCE, 2003, London, UK. **Proceedings…** Disponível em: <http://www.idealliance.org/papers/dx_xmle03/papers/02-05-04/02-05-04.html> Acesso em: nov. 2005.

BOER, A. et al. Proposal for a Dutch Legal XML Standard. In: INTERNATIONAL CONFERENCE ON ELECTRONIC GOVERNMENT, EGOV, 1., 2002, Aix-en-Provence, France. **Electronic Government**: proceedings. Berlim: Springer, 2002. p. 3-12. (Lecture Notes in Computer Science, v. 2456).

BOLEY, H. **The Rule Markup Initiative**. 2002. Disponível em: <http://www.dfki.uni-kl.de/ruleml/> Acesso em: out. 2005.

BORBINHA, J.; FREIRE, N. **Metadados**. Portugal: Biblioteca Nacional e INESC. Disponível em: < http://metadados.bn.pt/>. Acesso em: nov. 2005.

BRADNER, S. **Key words for use in RFCs to Indicate Requirement Levels:** RFC 2119. [S.l.]: Internet Engineering Task Force, 1999.

BRANTING, L. K. Techniques for automated drafting of judicial documents. **International Journal of Law and Information Technology**, Oxford, v. 6, n. 2, p. 214-229, 1998.

BRANTING, L. K. et al. Automated Drafting of Self-Explaining Documents. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 6., 1997. **Proceedings…** New York: ACM, 1997. p. 72–82.

BRANTING, L. K. et al. Integrating Discourse and Domain Knowledge for Document Drafting. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 7., 1999. **Proceedings…** New York: ACM, 1999. p. 214–220.

BUNEMAN, P. Semistructured Data. In: SIGMOD INTERNATIONAL SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, PODS, 16., 1997, Tucson, Arizona. **Proceedings…** [S.l.: s.n.], 1997. p.117-121.

BUNEMAN, P. et al. Adding Structure to Unstructured Data. In: DATABASE THEORY, ICDT, 1997, Delphi, Grecee. **Proceedings…** [S.l.]: Springer, 1997. p 336-350.

DASKALOPULU, A.; SERGOT, M. J. A. Constraint-Driven System for Contract Assembly. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 5., 1995. **Proceedings…** New York: ACM , 1995. p.62–69.

DASKALOPULU, A.; SERGOT, M. J. The Representation of Legal Contracts. **AI & Society**, Berlin, v. 11, n. 1/2, p. 6-17, 1997.

DASKALOPULU, A. Legal Contract Drafting at the Micro-Level. In: LAW IN THE INFORMATION SOCIETY, 5., 1998. **Proceedings…** [S.l.: s.n.], 1998.

DEBAENE, S. et al. SOLON. Een computersysteem ter ondersteuning van de wetgevingsactiviteit van de Vlaamse regering. In: DEBAENE, S. ; BUGGENHOUT, B. van (Ed.). **Informatietechnologie en de kwaliteit van wetgeving**. Antwerpen: Intersentia, 2000. p. 79–120.

DORNELES, C. **Extração de dados semi-estruturados com base em uma ontologia.** 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

eContracts Version 1.0 Committee Specification , 27 April 2007, Document identifier: legalxml-econtracts-specification-1.0. Disponível em: <http://docs.oasis-open.org/legalxml-econtracts/legalxmlecontracts-specification-1.0.html>. Acesso em: nov. 2006.

EGUCHI, G.; LAURENCE, L. Rule-based XML. **Artificial Intelligence and Law,** The Netherlands, [S.l.], v. 10, n. 4, p. 283–294, 2002.

ERDMANN, M.; STUDER, R. Ontologies as Conceptual Models for XML Documents. In: WORKSHOP FOR KNOWLEDGE, ACQUISITION, MODELING AND MANAGEMENT, 12., 1999. **Proceedings…** [S.l.: s.n.], 1999.

ERDMANN, M.; DECKER, S. **Ontology-aware XML-Queries.** 2000. Disponível em: <http://xml.coverpages.org/erdmann-semantic-xql-webdb00.pdf>. Acesso em: dez. 2007

ERWIG, M. **A Visual Language for XML.** 2000. Disponível em: <http://citeseer.ist.psu.edu/erwig00visual.html>. Acesso em: dez. 2007.

ERWIG, M. Xing: A Visual XML Query Language. **Journal of Visual Languages and Computing,** [S.l.], v. 14, n. 1, 2003.

FLORESCU, D.; LEVY, A.; MENDELZON, A. Database Techniques for the World Wide Web: A Survey. **SIGMOD Record**, New York, v. 27, n. 3, p.59-74, Mar. 1997.

GIL, Y.; RATNAKAR, V. A Comparison of (Semantic) Markup Languages. In: INTERNATIONAL FLAIRS CONFERENCE, 15., 2002. **Proceedings…** [S.l.]: AAAI Press, 2002.

GOLDMAN, R.; MCHUGH, J.; WIDOM, J. From Semistructured Data to XML:Migrating the Lore Data Model and Query Language. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, WEBDB, 2., 1999. **Proceedings…** [S.l.: s.n.], 1999.

GÓMEZ-PÉREZ, A. et al. **Ontological Engineering: with examples from areas of knowledge management, e-commerce and the semantic web.** New York: Springer-Verlag, 2004.

GOTTSCHALK, P. Chapter X - Knowledge Management in Law Firms. In: **Knowledge Management Systems In Law Enforcement:** Technologies and Techniques. [S.l.]:Idea Group, 2007. p. 252-275.

GRUBER, T. R. **Toward Principles for the Design of Ontologies Used for Knowledge Sharing.** Standford: Knowledge Systems Laboratory, Stanford University, 1993. (Technical Report KSL 93-04).

HATTER, C. **Approaches to Modeling Legislative Texts in XML.** 2006. Disponível em: <http://2006.xmlconference.org/proceedings/110/presentation.pdf>. Acesso em: nov. 2006.

HEUSER, C. A. et al. Dados Semi-Estruturados. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS**,** 15., 2000, JoãoPessoa, PB. **Anais...** João Pessoa: [s.n.], 2000.

JASPER, R.; USCHOLD, M. **A Framework for Understanding and Classifying Ontology Applications.** Disponível em: <http://www.ietf.org/rfc/rfc2119.txt>. Acesso em: nov. 2006.

ISO/IEC. **ISO/IEC 19757-3:** Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation - Schematron. [S.l.], 2006. Disponível em: <http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip>. Acesso em: out. 2006.

KERRIGAN, S.; LAW, K. H. Logic-Based Regulation Compliance Assistance. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 9., 2003. **Proceedings…**. New York: ACM, 2003. p. 126–135.

KUDRASS, T. Coping with semantics in XML document management. In: OOPSLA WORK-SHOP ON BEHAVIORAL SEMANTICS - BACK TO BASICS, 10., **Proceedings…** [S.l.: s.n.], 2001. p . 150-161.

LAENDER, A. H. F. et al. A Brief Survey of Web Data Extraction Tools. **SIGMOD Record**, New York , v. 31, n. 2, 2002.

LASSILA, O.; MCGUINNESS, D. **The Role of Frame-Based Representation on the Semantic Web.** Standford: Knowledge Systems Laboratory, Stanford University, 2001. (Technical Report KSL-01-02).

LAURITSEN, M. Frontiers - Legal Drafting Systems. In: INTERNATIONAL CONFERENCE ON AI AND LAW, 11., 2007. **Proceedings...** [S.l.: s.n.], 2007.

LEGALDOCS: Legal Documents Online. Disponível em: <https://www.legaldocs.com/>. Acesso em: dez. 2006.

LEHTONEN, M. et al. A dynamic user interface for document assembly. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING**,** 2002. **Proceedings…** [S.l.: s.n.], 2002. p. 134-141.

MCCALLUM, A. Information Extraction: Distilling Structured Data from Unstructured Text. **Queue**, New York, v. 3, n. 9, p. 48-57, Nov. 2005.

MCCLURE, J. **Tagging Legal Text.** [S.l.]: Hypergrove Engineering, 2006. Disponível em: <http://www.hypergrove.com/legalrdf.org/LegalMarkup.html>. Acesso em: jan. 2007.

MERCATALI, P. Computer-Aided Methods and Tools for Legislative Drafting. In: DEBAENE, S.; BUGGENHOUT, B. Van (Ed.). **Informatietechnologie en de kwaliteit van wetgeving**. Antwerpen: Intersentia, 2000. p. 139–156.

MILDE, J. UXO: An XML-Based Extensible Annotation Editor. In: GLDV-SPRING MEETING, 2001. **Proceedings…** [S.l.: s.n.], 2001.

MINSKY, M. A framework for representing knowledge. In: WINSTON, P. H. (Ed.)**. The Psychology of Computer Vision.** New York: McGraw-Hill, 1975. p. 211-278.

MOENS, M.-F. Improving Access To Legal Information: How Drafting of Computer. In: LODDER, A. R.; OSKAMP, A. (Ed.). **Information Technology and Lawyers**: Advanced Technology in Legal Domain, from Challenges to Daily Routine. Dordrecht, The Netherlands: Springer, 2006. p. 119–136.

MOTTA, E. et al. Ontology-Driven Document Enrichment: Principles, Tools and Applications. **International Journal of Human-Computer Studies**, London, v. 52, p. 1071-1109, 2000.

MOUNTAIN, D. Disrupting Conventional Law Firm Business Models using Document Assembly. **International Journal of Law and Information Technology**, Oxford, v. 15, n. 2, 2007.

ISO/IEC. **ISO/IEC 21000-5:2003:** Information Technology – Multimedia Framework – Part 5: Rights Expression Language. Disponível em: <http://www.iso.ch/iso/en/CombinedQueryResult.CombinedQueryResult?queryString= 21000-5>. Acesso em: out. 2006.

NATALYA, N. Order from Chaos. **Queue**, New York, v. 3, n. 8, Oct. 2005. Disponível em:<http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=341&page=1>. Acesso em: set. 2006.

NEWELL, A. The knowledge level. **Artificial Intelligence**, Amsterdam, v. 18, p. 87-127, 1982.

O'LEARY, D. E. Knowledge Management Systems: Converting and Connecting. **IEEE Intelligent Systems**, Los Alamitos, v. 13, n. 3, p. 30-33, May/June 1998.

PEASE, A; NILES, I.; LI, J. The Suggested Upper Merged Ontology: a large ontology for the semantic web and its applications. In: WORKSHOP ON ONTOLOGIES AND THE SEMANTIC WEB, 2002, Edmonton, Canada. **Proceedings…** [S.l.: s.n.], 2002.

POULIN, D.; HUARD, G.; LAVOIE, A. The Other Formalization of Law: SGML Modelling and Tagging. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 6., 1997. **Proceedings…** New York: ACM, 1997. p. 82–88.

PRIÉ, Y. Sur la piste de l'indexation conceptuelle de documents. **Document Numérique**, Paris, v. 4, n. 162, p. 11-35, Dec. 2000.

RIVALDO, R. et al. SML Model-based Management. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, 10. , 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 761-764.

RIVALDO, R. et al. Improving Distributed Service Management Using Service Modeling Language (SML). In: IFIP/IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM , NOMS, 2008. **Proceedings…** [S.l.: s.n.], 2008.

RUBINO, R.; ROTOLO, A.; SARTOR, G. An OWL Ontology of Fundamental Legal Concepts. In: ANNUAL CONFERENCE ON LEGAL KNOWLEDGE AND INFORMATION SYSTEMS, JURIX, 19. , 2006. **Proceedings…** Amsterdam: IOS Press, 2006.

SARTOR, G. Fundamental legal concepts: A formal and teleological characterization. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND LAW, 14., 2006. **Proceedings…** New York: ACM , 2006. p. 101-142.

SML: Service Modeling Language (SML) Working Group. Disponível em: <http://www.w3.org/XML/SML/>. Acesso em: nov. 2006.

SML WD: Service Modeling Language, W3C Working Draft. Disponível em: <http://www.w3.org/TR/sml/>. Acesso em: nov. 2006.

SOWA, J. F. **Semantic Networks.** Disponível em: <http://www.jfsowa.com/pubs/semnet.htm>. Acesso em: 27 set. 2005.

SPERBERG-MCQUEEN, C. M. XML and Semi-Structured Data Semi-Structured Data. **QUEUE**, New York, v. 3, n. 8, Oct. 2005.

STAAB, S. et al. Knowledge Processes and Ontologies. **IEEE Intelligent Systems**, Los Alamitos, v. 16, n. 1, p.26-34, 2001.

SWETLAND. **Setting the Stage - An Introdution to Metadata.** Disponível em: <http://www.getty.edu/research/conducting_research/standards/intrometadata/2_articles/index.html>. Acesso em: out. 2006.

VANZIN, M.; ABEL, M. **Projeto IDOC**. 2001. Relatório de Pesquisa – Instituto de Informática, UFRGS, Porto Alegre.

VAN DER VET, P. E.; MARS N. J. I. Bottom-up Construction of Ontologies. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 10, n. 4, p. 513-526, 1998.

VIVANCOS-VICENTE, P.J. et al. An Approach for Multirelational Ontology Modelling. In: PACIFIC RIM INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, PRICAI, 8., 2004, Auckland, New Zeland. **Trends in Artificial Intelligence**: Proceedings. Berlim: Springer, 2004. p. 997-998. (Lecture Notes in Artificial Intelligence, v. 3157).

VOERMANS, W. **Sturen in de mist. Maar dan met radar**. 1995. Ph.D. Dissertation, Tilburg University.

WANG, X. et al. **The Contract Expression Language – CEL**. Disponível em: <http://www.contentguard.com/drmwhitepapers/The_CEL.pdf>. Acesso em: nov. 2006.

WELTY, C. **Towards a Semantics for the Web.** 2000. Disponível em: <http://citeseer.nj.nec.com/welty00towards.html>. Acesso em: dez. 2006.

WILKINSON, R. et al. **Document Computing:** Technologies for Managing Electronic Document Collections. Boston: Kluwer Academic Publishers, 1998.

WS-ADRESSING CORE: Web Services Addressing 1.0 - Core. Disponível em: <http://www.w3.org/TR/ws-addr-core>. Acesso em: nov. 2006.

XML: Extensible Markup Language (Xml) 1.0 (Fourth Edition). Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: out. 2006.

XML SCHEMA: The Xml Schema Working Group - W3C. Disponível em: <http://www.w3.org/XML/Schema>. Acesso em: out. 2006.

XML SCHEMA DATATYPES: XML Schema Part 2: Datatypes Second Edition. Disponível: <http://www.w3.org/TR/xmlschema-2>. Acesso em: nov. 2006.

XML SCHEMA STRUCTURES: XML Schema Part 1: Structures Second Edition. Disponível em: <http://www.w3.org/TR/xmlschema-1>. Acesso em: nov. 2006.

XMLNS() SCHEME: XPointer xmlns() Scheme. Disponível em: <http://www.w3.org/TR/xptr-xmlns/>. Acesso em: out. 2006.

XPATH: XML Path Language (XPath) Version 1.0. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em: out. 2006.

XPOINTER: XPointer Framework. Disponível em: <http://www.w3.org/TR/xptr-framework/>. Acesso em: nov. 2006.

XPOINTER() SCHEME: XPointer xpointer() Scheme. Disponível em: <http://www.w3.org/TR/xptr-xpointer/>. Acesso em: nov. 2006.

WXPYTHON. Disponível em: <http://www.wxpython.org>. Acesso em: out. 2006.