Instituto Superior de Engenharia do Porto

Uma Abordagem de Alto Nível para a Verificação de Conteúdos na Web

Liliana Isabel Gouveia Alexandre

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Área de Especialização em Tecnologias do Conhecimento e da Decisão

Orientador: Doutor Jorge Manuel Neves Coelho

Júri:

Presidente: Doutora Maria de Fátima Coutinho Rodrigues

Vogais: Doutor Nuno Alexandre Pinto da Silva
Doutor Jorge Manuel Neves Coelho

Porto, Junho de 2011

Dedico este projecto aos meus pais e namorado

Agradecimentos

Em especial gostaria de agradecer, em primeiro lugar, aos meus pais, **Amália Gouveia** e **Victor Alexandre**, que sempre me apoiaram em todas as decisões tomadas, e por todo o carinho e compreensão que sempre demonstraram ter.

Um muito obrigado ao meu primo, irmão e amigo, **Tiago Silva**, que esteve sempre a meu lado nos momentos mais difíceis.

Aos meus tios **Rosário Sousa** e **Manuel Almeida** pelos tios fantásticos que sempre demonstraram ser e por todo o apoio na realização de mais uma etapa. Agradeço ainda aos meus primos, **Sandra Almeida** e **Emanuel Almeida** pelos fantásticos amigos que têm sido ao longo da vida.

Ao meu orientador, **Dr. Jorge Coelho**, agradeço sobretudo pelo apoio, conselhos e pronta disponibilidade que sempre demonstrou ter durante a elaboração deste projecto. Agradeço ainda todo o material bibliográfico por ele disponibilizado e revisão desta dissertação.

Agradeço ainda à entidade **Lusitânia – Companhia de Seguros** em união com a **NSeguros**, por toda a compreensão demonstrada durante os dois anos em que decorreu o Mestrado e por todo o apoio prestado, especialmente durante o último ano.

A todos os meus colegas de trabalho, sem excepção, pelos conselhos, amizade, compreensão e ajuda.

Agradeço ainda aos meus companheiros de trabalhos académicos e amigos **Ricardo Jafe**, **Rui Silva** e **Pedro Duque** por todo o apoio na realização dos mesmos. O contributo deles foi fundamental para que eu chegasse até aqui.

Não poderia deixar de agradecer a todos os meus amigos que sempre se mostraram compreensivos com as minhas ausências e que de uma forma ou de outra contribuíram para que este projecto se realizasse nas melhores condições.

Agradeço sobretudo, ao meu namorado **Renato Cardoso**, por toda a compreensão e disponibilidade demonstrada, pelos conselhos prestados e, acima de tudo, pelo apoio, paciência e amizade ao longo da realização deste projecto.

A todos o meu muito obrigada!

Resumo

A manutenção do conteúdo web pode ser uma tarefa difícil, especialmente se considerarmos websites em que muitos utilizadores têm permissões para alterar o seu conteúdo. Um exemplo deste tipo de websites são os wikis. Se por um lado permitem rápida disseminação de conhecimento, por outro lado implicam um grande esforço para verificar a qualidade do seu conteúdo. Nesta tese analisamos diferentes abordagens à modelação de websites, especialmente para a verificação de conteúdo, onde contribuímos com uma extensão à ferramenta VeriFLog para a tornar mais adequada à verificação de conteúdos em websites colaborativos.

Palavras-Chave

XML, Web, Verificação, Programação em Lógica

Abstract

Imposing constraints in web site content can be a difficult task, especially when such content is edited by many different people. One example of such scenario Is open collaboration web sites such as wikis. There, many different users can add and edit content. In one hand we have a way to rapidly disseminate a considerable amount of knowledge but on the other hand we need a constant and big effort to ensure high quality standards. In the thesis we survey different approaches to model and capture website's properties and focus on content verification where we can contribute with an extension of the VeriFLog tool with new features to enhance its application to open collaboration web sites.

Keywords

XML, Web, Verification, Logic Programming

Índice

Índice de Figuras

Glossário

Browser – Programa que permite aceder e interagir com toda a informação disponível na Internet. Utiliza o protocolo HTPP para fazer pedidos de páginas Web ao servidor Web.

Framework – conjunto de bibliotecas de software, disponibilizadas através de uma API, que fornecem determinadas funcionalidades genéricas que podem ser reescritas ou especializadas através do código do utilizador, tornando-as assim mais específicas.

Internet – É um sistema global de redes de computadores interligadas e que usam o protocolo TCP/IP para disponibilizar informação aos utilizadores por todo o mundo.

Intranet – Rede de computador privada que usa o protocolo da Internet para a partilha de informação dentro de uma organização.

Servlet – componente do lado do servidor que gera dados HTML e XML para a camada de apresentação de uma aplicação Web

Lista de Abreviaturas

API – Application Programming Interface

ASP - Active Server Pages

CGI - Common Gateway Interface

CFG - Control Flow Graphs

DBMS - Data Base Management System

HTML - Hypertext Markup Language

JSP - JavaServer Pages

LTS – Labelled transition system

PHP - PHP: Hypertext Preprocessor

SDL - Specification and Description Language

TCP/IP – Transmission Control Protocol/Internet Protocol

URL - Uniform Resource Locator

XML - Extensible Markup Language

XSD - XML Schema Definition

WebCFG – Web Control Flow Graphs

1. Introdução

As aplicações Web têm-se tornado cada vez mais complexas, sendo hoje em dia, muito mais interactivas e dinâmicas. Deixaram de ser apenas sites onde a informação era disponibilizada de modo estático para passarem a ser ferramentas onde é possível a realização de outro tipo de operações, como o comércio electrónico, e a visualização de novos conteúdos ao nível do entretenimento. O modo como são construídas e disponibilizadas também se modificou substancialmente. Suportam, agora, mais do que simples pesquisa e navegação, sendo capazes de executar operações que afectam o seu conteúdo e o seu estado de navegação. Além disso, são capazes de integrar as suas operações com as funcionalidades disponibilizadas pelos próprios browsers. O número de *hyperlinks* utilizados, na sua construção, é também muito maior e os conteúdos disponibilizados são mais gráficos e dinâmicos. Incluem um conjunto de linguagens de programação muito mais diversificado do que as iniciais e podem ser construídas por vários utilizadores que se encontram geograficamente afastados. O facto do número de acessos às aplicações ter aumentado largamente nos últimos anos, implicou que a arquitectura das mesmas fosse adaptada para que a sua performance não fosse comprometida.

Todos estes factores contribuem para que a complexidade das aplicações Web seja cada vez maior, tornando a sua manutenção numa tarefa bastante difícil, morosa e muitas vezes aborrecida. A modelação surge, neste contexto, como uma ferramenta essencial para lidar com a complexidade destes sistemas, fornecendo uma ajuda preciosa na manutenção dos mesmos. Na literatura disponível, podem ser encontrados vários modelos propostos para a validação e verificação das aplicações Web. Apesar de alguns desses modelos se apresentarem como uma abordagem inovadora, existem outros que são adaptados de técnicas de modelação já existentes para outros tipos de software. A modelação pretende apresentar uma visão abstracta da aplicação Web, ajudando os *designers* na fase de desenvolvimento da aplicação, através da definição de requisitos, na fase de testes, posterior à implementação e ainda em fases posteriores para verificação.

As primeiras técnicas de modelação centravam-se, essencialmente, no *design* das aplicações Web. No entanto, a modelação pode ser realizada ao nível da navegação (validação das ligações existentes nas páginas), ao nível da interacção (pressupõe todas as operações entre o utilizador, a aplicação e o browser) e ainda ao nível do conteúdo (validar se as páginas estão correctas e completas).

O trabalho aqui apresentado centra-se, na modelação ao nível do conteúdo, não deixando de abordar algumas das técnicas de modelação para validação e verificação da navegação e interacção, com o objectivo de ser mais abrangente e capturar possíveis pontos de ligação.

Quando uma aplicação Web é mantida por diversas pessoas e é necessário controlar os conteúdos existentes, torna-se bastante difícil executar esta tarefa manualmente. Nos últimos anos têm surgido diversas abordagens para a verificação e validação de conteúdos Web, sendo algumas delas baseadas em programação em lógica e na sua capacidade para tratar o conhecimento. Embora se analise outros modelos ao nível do conteúdo, o presente documento centra-se, principalmente, na abordagem proposta em [Coelho J. e Florido M., 2006], por esta se basear na programação em lógica.

Ao nível do conteúdo a modelação é utilizada para a verificação e validação do conteúdo de uma aplicação Web, ou seja, pretende-se que a técnica de modelação aplicada, seja capaz de verificar e validar se a informação, contida numa página, aparece na totalidade ou se há alguma que está em falta ou errada, e caso seja verdade, definir regras que executem determinadas operações sobre esse conteúdo específico. Além disso, é de esperar, também, que os modelos possam inferir nova informação a partir do conteúdo disponibilizado na aplicação.

A programação em lógica consiste no uso da lógica de primeira ordem [1]. Trata-se de uma linguagem puramente declarativa, uma vez que descreve os relacionamentos lógicos necessários para resolver um dado problema, com elevada capacidade para tratamento de conhecimento. A programação em lógica com restrições combina os aspectos declarativos da programação em lógica com a eficiência dos métodos de resolução de problemas que recorrem a restrições. Uma das vantagens mais significativa desta linguagem é a de oferecer um menor tempo de desenvolvimento de programas e uma eficiência mais elevada, quando comparada com linguagens imperativas.

É com base nesta linguagem que em [Coelho J. e Florido M., 2004] e [Coelho J. e Florido M., 2007b] é proposta uma linguagem para o processamento de documentos XML [6].

O XML é uma metalinguagem útil para a descrição de linguagens específicas de domínio para documentos estruturados e é, hoje em dia, considerado o formato padrão para a representação de informação produzida por diferentes aplicações.

Com base nestes conceitos e como ponto de partida para a criação da ferramenta, proposta neste documento, foram analisadas a linguagem e *framework* apresentadas em [Coelho J. e Florido M., 2004] e [Coelho J. e Florido M., 2006a], respectivamente. A ferramenta, denominada VeriFLog, utiliza a linguagem XCentric para disponibilizar as várias funcionalidades que permitem a validação e verificação de conteúdos para a Web.

A linguagem XCentric é uma linguagem de programação em lógica que implementa restrições, especializada no processamento de documentos XML. Esta linguagem permite a utilização de termos com símbolos de função com aridade flexível, implementando um mecanismo para a unificação não *standard* desses termos através da utilização de variáveis de sequência. O principal objectivo desta linguagem é disponibilizar uma ferramenta que torne o processamento de XML mais simples. Os elementos que compõem um documento de XML podem ser traduzidos para a notação interna do XCentric, em que o functor principal da função é igual à tag raiz do documento e esta pode conter zero ou mais argumentos. A sintaxe desta linguagem é, em muito, semelhante à do Prolog, com a diferença que implementa a restrição $=*=$.

O VeriFLog trata-se de uma aplicação, que tem por base a linguagem XCentric. Esta framework permite a tradução dos documentos XML em termos, possibilitando a validação sintáctica, a verificação semântica e a inferência de novos dados a partir dos existentes. A ideia fundamental é a de disponibilização de uma interface para dados semi-estruturados, capaz de descrever regras que validem e verifiquem o conteúdo de páginas Web. As principais regras disponibilizadas pela framework são: *delete* (remover conteúdo no caso de determinada restrição se verificar); *replace* (substituição do conteúdo existente por um novo se se verificar a restrição definida); *fail* (usada quando existem erros nas páginas demasiado graves para serem resolvidos automaticamente).

O presente trabalho tem como principal objectivo a implementação de uma ferramenta gráfica que estenda as principais funcionalidades oferecidas pelo VeriFLog, tornando-se mais amigável para o utilizador e também mais completa. As restrições básicas possíveis de definir visualmente na aplicação são: verificar a existência de determinado conteúdo num elemento, validar se o elemento contém um URL válido, testar se o conteúdo do elemento é vazio e ainda aplicar a negação a cada uma delas. Para além dessas funcionalidades, foram acrescentadas outras que, tornam a aplicação mais robusta, sendo exemplo disso, a possibilidade de introdução de regras manuais (definição de outras restrições não disponíveis graficamente), onde se

pressupõe o conhecimento da programação em lógica; a eliminação, edição e reutilização de regras previamente configuradas.

Este documento encontra-se organizado em cinco capítulos. O primeiro capítulo corresponde à Introdução no qual é possível obter uma visão generalizada do trabalho realizado e quais as motivações que levaram à sua execução.

O segundo capítulo apresenta o estado da arte no que diz respeito às abordagens propostas e metodologias utilizadas na área em que o presente trabalho se enquadra. Serão enumeradas e descritas as propostas de vários autores, com especial relevo para aquelas que validam e verificam o conteúdo de aplicações Web.

No terceiro capítulo serão descritas a linguagem XCentric e a framework VeriFLog como metodologias para a verificação e validação do conteúdo de páginas Web, fazendo referência para o papel fundamental que desempenharam ao servir de base para a implementação e realização deste trabalho.

O quarto capítulo descreve a aplicação desenvolvida, apresentando exemplos de todas as funcionalidades que disponibiliza. Além disso, é explicada a motivação por detrás de cada uma das suas funcionalidades e quais as tecnologias utilizadas na sua implementação, bem como o modo como estas se interligam.

Finalmente, no capítulo da conclusão, serão enumeradas todas as conclusões decorrentes do trabalho realizado e ainda uma referência ao trabalho futuro que ainda é possível desenvolver.

Este trabalho foi parcialmente apresentado em [Alexandre L. e Coelho J., 2011a] e em [Alexandre L. e Coelho J., 2011b].

2. Estado da Arte

As aplicações Web têm-se tornado cada vez mais complexas. Esta complexidade deve-se a vários factores, tais como, o aumento da interacção entre o utilizador e as aplicações, o uso de servidores distribuídos, a preferência por aplicações dinâmicas que utilizam um número elevado de ligações entre as várias páginas. A modelação é a ferramenta que nos pode ajudar a compreender melhor estes sistemas complexos. Os modelos serão uma ajuda fundamental na fase de *design* da aplicação *Web,* ao fornecer diferentes níveis de detalhe, e na fase de testes, posterior ao desenvolvimento da aplicação, tornando-se essenciais para a validação e verificação da mesma.

A análise do trabalho desenvolvido por [Manar et al., 2009] foi o ponto de partida para o desenvolvimento do presente trabalho. Embora o estudo aborde métodos de modelação que abrangem o comportamento, navegação e conteúdo Web, centrar-nos-emos essencialmente, na análise de modelos que são actualmente usados para testar e verificar aplicações Web ao nível do conteúdo.

O estudo de [Manar et al., 2009] centra-se na análise de modelos subjacentes à verificação e testes de aplicações Web. Foca-se principalmente em métodos que propõem modelos para capturar diferentes propriedades relacionadas com a estrutura (navegação), comportamento e conteúdo das aplicações e se essas propriedades são estáticas, dinâmicas ou interactivas. Este estudo pretende analisar o estado da arte dos modelos para a verificação e testes de aplicações Web, bem como, mostrar o que já foi feito e o que ainda é necessário fazer para colmatar as lacunas encontradas. Nesta tese tentamos inovar um pouco num destes domínios.

Este capítulo encontra-se dividido em cinco secções. A primeira secção apresenta uma breve descrição das aplicações Web e dos seus componentes, a segunda secção enumera os principais desafios na análise e modelação para verificação e teste destas aplicações e que contribuíram para o aparecimento dos modelos identificados neste estudo, a terceira secção resume as propriedades que os modelos de aplicações Web devem possuir, bem como as principais características que cada modelo é capaz de capturar, a quarta secção corresponde a uma descrição dos modelos propostos por vários autores, agrupados por nível de modelação, focando, essencialmente, os modelos de conteúdo Web baseados em programação em lógica, uma vez que é com base neste domínio que o trabalho aqui proposto é desenvolvido. Por fim, na quinta secção é feita uma análise geral dos métodos de modelação e do que ainda falta fazer nesta área.

2.1 Aplicações Web

Uma aplicação Web é um programa de computador que pode ser acedido a partir de um cliente, como por exemplo, um browser através de uma rede como a Internet ou de uma Intranet. As aplicações Web têm uma estrutura de três camadas (three-tiered), representadas na Figura 1. O *Web Browser* representa a primeira camada; o *Web Server* que implementa CGI, PHP, *Java Servelets* ou Páginas ASP, em conjunto com os servidores de aplicação que interagem com a base de dados e com outros objectos Web formam a camada intermédia e finalmente a base de dados em conjunto com o sistema DBMS formam a terceira camada.

As aplicações Web geram páginas Web, estáticas ou dinâmicas, que são compostas por diferentes tipos de informação, tais como, texto, imagens e formulários. As páginas estáticas residem num servidor Web e contêm apenas HTML, código cliente executável (exemplo: JavaScript) e são alimentadas por um servidor Web. As páginas dinâmicas são geradas a partir da execução de um conjunto de scripts e componentes que se encontram no servidor e são compostas por HTML, código fonte e código executável e são servidas pelo servidor aplicacional.

O trabalho apresentado por [Manar et al., 2009] descreve métodos capazes de capturar diferentes propriedades relacionadas com a estrutura, comportamento e conteúdo das aplicações Web, e com o facto de essas propriedades serem estáticas, dinâmicas ou interactivas.

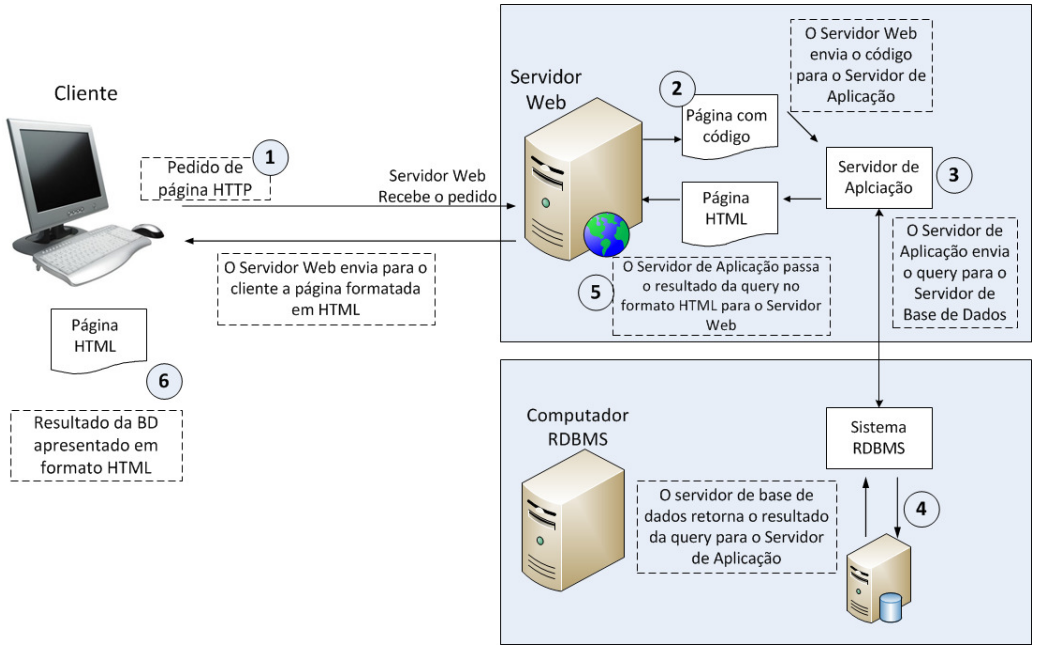

Figura 1 – Componentes de uma aplicação Web (adaptada de [Manar et al., 2009])

2.2 Desafios na análise e modelação de Aplicações Web

Actualmente, temos assistido a uma grande e rápida evolução das aplicações Web. Novas tecnologias, linguagens e modelos de programação contribuem para o aumento da interactividade e usabilidade destas aplicações. Esta evolução comporta, consequentemente, um aumento da complexidade, que traz novos e interessantes desafios para a modelação, análise, teste e verificação deste tipo de software. Alguns desses desafios são:

- As aplicações Web são muito diversificadas e cada vez mais complexas. Este tipo de aplicações interage com diversos componentes que, correm em diferentes plataformas de hardware e software. Podem ser escritas em diferentes linguagens de programação e basear-se em abordagens de programação diferentes. O cliente inclui browsers, HTML, linguagens script embutidas e applets e o servidor inclui CGI, JSPs, Java Servlets e tecnologias .NET. Ambos interagem com componentes que podem ser encontrados nos servidores Web ou noutros servidores. A integração de todos estes componentes pode tornar a análise da verificação e teste das aplicações baste difícil.
- Comportamento dinâmico das aplicações Web.
- As aplicações Web podem ser acedidas de qualquer parte do mundo e por um número variado de utilizadores. Importa que a informação disponibilizada seja a mesma para os diferentes utilizadores.
- Usualmente o output que é observado e analisado consiste apenas nos documentos HTML que resultam de pedidos provenientes do utilizador. No entanto, existem outros tipos de output numa aplicação Web, que também merecem ser alvo de análise, como por exemplo as alterações ao estado do servidor e da base de dados ou as mensagens enviadas entre aplicações Web e outros serviços. Torna-se um desafio efectuar uma análise das aplicações Web que tenha em consideração todo este tipo de informação e não apenas o que é mostrado ao utilizador.

2.3 Categorização dos modelos para aplicações Web

As aplicações Web podem ser classificadas segundo três níveis de modelação: navegação Web, conteúdo Web e comportamento Web. Nesta secção será apresentada uma categorização das principais propriedades que uma aplicação Web

deverá ter, de acordo com cada nível de modelação, e serão listadas os tipos de características capturadas por cada um desses modelos.

2.3.1 Propriedades

No que diz respeito à modelação ao nível da navegação são avaliadas as propriedades estática, dinâmica e de interacção da aplicação web. As propriedades de navegação estática descrevem o funcionamento dos *links* estáticos, ou seja, consiste na verificação de links que se perderam, acessibilidade e estimativa do custo de navegação, como por exemplo, a análise do caminho mais longo. As propriedades de navegação dinâmica descrevem todos os aspectos que tornam a navegação dinâmica, por exemplo, o mesmo *link* pode redireccionar para páginas diferentes dependendo da informação de entrada, que pode ser dada pelo utilizador, ou obtida através do sistema. As propriedades de navegação de interacção estão relacionadas com a navegação realizada pelo utilizador e que se encontra fora do controlo da aplicação, como é o caso da interacção do utilizador com o browser.

Os modelos centrados no conteúdo web validam propriedades de conteúdo estático e dinâmico. As propriedades de conteúdo estático verificam a consistência da página web original em relação à sua sintaxe e semântica, englobando a verificação de que uma página se encontra completa, ou seja, o modelo deve ser capaz de validar se uma determinada página Web contém a informação pretendida, se os *links* entre as várias páginas existem e também se a própria página existe, e ainda verificam se uma página se encontra correcta, o que significa avaliar se a informação contida numa página web é válida de acordo com os requisitos da aplicação. As propriedades de conteúdo dinâmico avaliam a consistência da sintaxe e da semântica do conteúdo dinâmico de uma página Web, ou seja, a validação do conteúdo gerado dinamicamente através da execução de scripts existentes no servidor da aplicação. É também aqui que devem ser validados todos os conteúdos que sejam gerados em tempo real.

A modelação ao nível do comportamento Web engloba a categorização das propriedades de segurança e de processamento de instrução. Para as primeiras são avaliados os controlos de mecanismo de acesso ao conteúdo web, por exemplo, os utilizadores não autorizados não devem ter acesso a certo tipo de informação se tal for assim definido. As propriedades de processamento de instrução descrevem todas as execuções efectuadas no lado do cliente e no lado do servidor que possam alterar o estado da aplicação. Um modelo deste nível deve ser capaz de identificar onde está a ocorrer a execução, se no lado do cliente, se no lado do servidor.

2.3.2 Características

As principais características, possíveis de capturar pelos métodos de modelação de aplicações web são as estáticas, dinâmicas e de interacção. As características estáticas incluem, como o próprio nome indica, as propriedades estáticas de uma aplicação Web, como por exemplo *links* que comunicam entre duas páginas HTML. Quando um utilizador faz um pedido ao servidor através de um botão ou *link* estático, o servidor responde ao pedido retornando ao cliente a página pedida. Esta característica relaciona-se com as propriedades de navegação estática e conteúdo estático das páginas Web.

As características dinâmicas descrevem todas as propriedades de conteúdo dinâmico e *links* dinâmicos. *Links* Dinâmicos correspondem à conexão entre a página HTML e o código que deve ser executado no servidor de modo a gerar a informação pedida e retorná-la para o cliente. Este processamento é feito com base na informação fornecida pelo utilizador, através do preenchimento de formulários, por exemplo, ou com base na informação existente no sistema, como por exemplo a data do servidor ou a interacção do servidor com outros recursos, como é o caso das bases de dados. Esta característica envolve as propriedades relacionadas com a navegação dinâmica, conteúdo dinâmico, segurança e instrução de processamento.

As características de interacção definem a influência que o browser tem no comportamento da navegação de uma aplicação Web. A interacção do utilizador com o browser pode alterar, interactivamente, os caminhos de navegação. Esta característica engloba as propriedades correspondentes à interacção do utilizador com o browser (navegação interactiva).

2.4 Modelação de aplicações Web

Nesta secção apresenta-se uma descrição dos métodos de modelação de aplicações Web com especial atenção para os métodos centrados no conteúdo Web, por serem estes os que serviram de base ao desenvolvimento e implementação do presente trabalho.

2.4.1 Métodos de modelação no comportamento da interacção

Estes modelos pretendem modelar todas as operações realizadas pelo utilizador. Estas operações dizem respeito à interacção do utilizador com a aplicação e com o browser, constituindo um ponto bastante importante de análise e ao mesmo tempo muito problemático. O utilizador pode, por exemplo, utilizar o botão de *back* no

browser para retroceder para um ponto de interacção anterior. Este tipo de interacção com o browser, como não é reportado para a aplicação Web pode provocar comportamentos inesperados da própria aplicação.

Alguns autores propuseram métodos, utilizando diferentes notações, capazes de modelar esta interacção. [Di Lucca e Di Penta, 2003] apresentaram um método de modelação que modela o browser como um grafo de estados que, representa as operações realizadas pelo utilizador e as transições entre essas operações, identificando se a operação foi feita no browser ou na aplicação.

Em [Graunke et al., 2003] é proposto um método que detecta a inconsistência dos dados inseridos pelo utilizador, através da identificação das alterações que ocorrem no servidor em tempo real. Este modelo pretende identificar dados errados que advêm da interacção do utilizador com o browser e com a própria aplicação, ou seja, identificação dados que são erradamente submetidos para o servidor, porque a interacção com os botões ou *links* da aplicação assim o permitiram e não propriamente porque o utilizador os inseriu incorrectamente.

Em [Licata and Krishnamurthi, 2005] é utilizado o modelo proposto pelos autores anteriores, reduzindo o número de interacções analisadas. A diferença existente entre os dois métodos reside no facto deste ser um método estático que garante a existência de todas as possibilidades de sequências de execução.

O modelo proposto em [Chen e Zhao, 2004] engloba todas as características dos modelos anteriores e adiciona a sua capacidade em representar uma pilha de histórico e o seu impacto na navegação, na cache local, e no refrescamento da página Web e em sessões de autenticação.

Em [Bordbar e Anastasakis, 2005] é criado um modelo abstracto que descreve as interacções entre o browser e a lógica de negócio.

Em suma, todos os modelos dos autores apresentados até agora são capazes de modelar as interacções das aplicações Web com o browser usando para tal modelos abstractos em diferentes notações. Todos os métodos analisados são capazes de modelar as operações de *backward* e de *forward* e alguns deles são ainda capazes de modelar outras características dos browsers como a pilha de histórico, as páginas em cache e as sessões de utilizador. Estes métodos integram manualmente o modelo de interacção com o modelo de navegação estática. Alguns dos métodos detectam bugs que existem entre o browser e a aplicação Web através de modelos de verificação implementados pelos próprios autores ou utilizando ferramentas de teste e modelação existentes. No entanto, nenhum dos modelos verifica a integração com aplicações

Web dinâmicas ou como é que as características dinâmicas afectam estes modelos de interacção.

2.4.2 Métodos de modelação do conteúdo de aplicações Web

Estes métodos pretendem verificar a semântica das aplicações Web, o que significa que estes métodos são responsáveis por verificar se uma determinada página Web se encontra completa e correcta, ou seja, são capazes de examinar e garantir que uma determinada página Web disponibiliza todo o conteúdo que deveria disponibilizar, de acordo com requisitos previamente definidos, que todos os *links* entre páginas existem e que a própria página também existe.

É notória a necessidade do aparecimento de mais métodos que forneçam este tipo de verificação, tanto para conteúdo Web estático como dinâmico.

A seguir serão apresentados os trabalhos propostos por alguns autores, sendo o trabalho proposto pelos segundos, aquele que serviu de base e motivação para a implementação da ferramenta apresentada neste documento.

Em [Alpuente et al., 2006] e [Alpuente et al., 2007] é proposto um método que verifica o conteúdo estático de uma aplicação Web de acordo com as propriedades semântica e a sintáctica através do uso da reescrita parcial. Esta tecnologia é usada, tanto para especificar condições de integridade, como para formalizar uma técnica de verificação que consegue identificar os requisitos que a aplicação Web não foi capaz de cumprir. Além disso, é ainda possível a reparação de erros através da identificação de páginas inexistentes ou de informação incompleta. Os autores apresentam, então, uma linguagem de especificação formal que, permite a definição de condições na estrutura e no conteúdo das aplicações Web de modo simples e conciso. Por exemplo, é possível a definição de condições que obrigam a que determinada informação esteja disponível numa dada página Web, que os *links* entre as páginas existam e funcionem correctamente ou que a própria página Web existe e está disponível. Através da análise dos requisitos não cumpridos pela aplicação Web, o método proposto, é capaz de identificar a informação em falta que é necessária para corrigir a página Web. O uso da reescrita parcial permite ainda a identificação de padrões que ocorrem em documentos semi-estruturados, como é o caso das páginas Web representadas por documentos HMTL e XML, é obtida através árvores de simulação [Henzinger et al., 1995].

Em [Coelho e Florido, 2006] foi apresentado um modelo capaz de verificar e reparar a semântica e a sintaxe do conteúdo de páginas Web baseadas em XML. Para tal usam

uma extensão ao Prolog denominada XCentric [Coelho e Florido, 2004] e [Coelho J. e Florido M., 2007b] . Trata-se de uma linguagem de programação em lógica para o processamento de XML, baseada na unificação de termos com funções de aridade flexível. Os autores implementaram uma *framework* designada VeriFLog [Coelho e Florido, 2006] utilizando a linguagem XCentric. Esta framework disponibiliza um conjunto de regras que permite a reparação e verificação que permitem avaliar o conteúdo de páginas Web. Permite também a tradução de documentos semi-estruturados, como é o caso do XML, para um conjunto de termos lógicos que correspondem ao conteúdo do documento original. Após a tradução do documento as regras de verificação e reparação são aplicadas e o documento é novamente traduzido para a sua estrutura inicial. Tanto a linguagem XCentric como a framework VeriFLog serão detalhadas no capítulo seguinte do presente trabalho.

2.4.3 Métodos de modelação da navegação em aplicações Web

Os modelos apresentados nesta secção usam diferentes notações e pretendem modelar todas as operações de navegação, possíveis de executar numa aplicação Web, por exemplo, um link que nos redirecciona para outra página.

Grande parte dos autores usa a notação UML [France R. et al., 1998] para a criação de métodos de modelação da navegação estática e dinâmica das aplicações Web. Em [Conallen, 1999] é proposta uma extensão à notação UML para representar os conteúdos estáticos e dinâmicos através de estereótipos. O modelo proposto por [Tonella e Ricca, 2000] baseia-se essencialmente na interacção do utilizador para extrair um modelo que representa as operações de navegação estática das aplicações Web sendo um modelo centrado na análise. [Bellettini et al., 2004] apresentam um modelo muito semelhante ao anterior diferindo no número de interacções necessárias, por parte do utilizador, para a criação desse modelo que, ao contrário do anterior, permite fazer uma análise estática e dinâmica da página.

Para além desta notação existem autores que usam grafos para a modelação da navegação. O uso de grafos não implica a conversão dos modelos para modelos formais, ao contrário do que acontece com os que se baseiam em UML. Um exemplo de modelos deste tipo, é o proposto por [Alfaro et al., 2001a] com a ferramenta MCWEB [Alfaro et al., 2001b], onde as várias páginas Web são representadas pelos nodos do grafo e os *links*, âncoras e frames pelas ligações entre esses nodos.

Existem ainda outros modelos propostos, baseados em diferentes notações, [Castelluccia et al., 2006], [Di Sciascio et al., 2005] e [Winckler e Palanque, 2003], bem como métodos de modelação híbridos, que não serão descritos no presente trabalho.

2.5 Análise geral dos métodos de modelação

Com base no estudo de [Manar et al., 2007] é possível reter algumas ideias e resultados relativamente aos métodos de modelação utilizados na análise do comportamento, conteúdo e navegação Web.

Os métodos de modelação do comportamento das aplicações Web são capazes de modelar a interacção das aplicações Web com o browser através de métodos abstractos representados em diferentes notações. Todos os métodos apresentados nesta categoria são capazes de modelar as operações básicas do browser (backward e forward) e alguns deles são também capazes de modelar características do browser como a pilha de histórico, a cache das páginas e as sessões de utilizador. Nenhum dos modelos apresentados aqui descrevem a integração com aplicações Web dinâmicas ou como é que as características dinâmicas afectam os modelos de interacção.

Os autores que apresentam métodos de modelação de conteúdo centram as suas propostas de modelos nos que são capazes de verificar o conteúdo estático das aplicações Web. Até à data de realização do trabalho apresentado por [Manar et al., 2007], nenhum dos modelos conseguia verificar o conteúdo dinâmico no sentido de validar se uma página se encontra correcta e completa. Com o aumento do dinamismo das aplicações Web é muito importante que esta lacuna seja preenchida.

Nos métodos de modelação da navegação, são usados modelos baseados em UML, grafos de estados e ainda em SDL para representar a navegação de uma aplicação Web. Para que os modelos baseados em UML possam ser usados para teste e verificação é necessário que os mesmos sejam traduzidos em modelos formais. Uma alternativa a estes modelos é a utilização de modelos baseados em grafos que, por serem formais, podem ser directamente usados para teste e verificação. Todos os modelos baseados em UML são capazes de capturar as características de navegação e de representar as características de uma aplicação Web.

O modelo ideal será algo que seja capaz de validar e verificar a aplicação web a todos os níveis de modelação. Apesar deste modelo ainda não existir, pelo menos até a presente data, talvez possa ser conseguido através da integração de alguns dos métodos analisados em [Manar et al., 2007].

3. Verificação de Conteúdos na Web

A manutenção de sites de grande dimensão pode ser uma tarefa difícil e também muito morosa e cansativa, especialmente se essa manutenção for executada por diferentes utilizadores. Aqui apresenta-se uma abordagem que seja capaz de efectuar a verificação e correcção de conteúdos web, tornando a sua manutenção mais simples e rápida.

Neste capítulo serão descritas a linguagem XCentric e a framework VeriFLog, para validação sintáctica e verificação semântica do conteúdo de páginas Web e nas quais se baseia o presente trabalho.

Este capítulo encontra-se divido em duas secções. A primeira descreve a linguagem XCentric [Coelho e Florido, 2004] e [Coelho J. e Florido M., 2007b], que se apresenta como uma linguagem de elevada expressividade através de um modelo de unificação baseado em funções de aridade flexível e variáveis de sequência e na segunda secção será detalhada a framework VeriFLog [Coelho e Florido, 2006b] baseada na linguagem XCentric que permite, de uma forma simples, a consulta de informação em documentos XML e HTML e ainda a escrita der regras de verificação de conteúdo.

3.1 XCentric: Linguagem de programação em lógica com restrições para processamento de XML

O XML é um formato para a representação de dados estruturados sob a forma de árvore que pode conter um número finito e arbitrário de nós folha. Existem dois tipos de conteúdo básicos num documento XML, os elementos e o texto simples. Um elemento consiste numa tag de início e outra de fim que deve conter qualquer sequência de outro conteúdo, como por exemplo $< Nome > Jorge\ Coelho </Nome >$. Além disso, cada elemento pode ainda conter atributos definidos por nome/valor $< Nome\ Importante = \text{"High"} > Jorge\ Coelho </Nome >$. A Figura 2 apresenta um exemplo de um documento de XML que descreve uma lista de docentes de um determinado departamento.

XCentric é uma linguagem baseada no Prolog com unificação de termos com funções de aridade flexível e variáveis de sequência. É uma abordagem que disponibiliza um modelo declarativo de alto nível para consulta de conteúdo em páginas Web. No XCentric os documentos XML são transformados para termos com símbolos de função de aridade flexível, o que significa dizer que o termo pode conter zero ou mais argumentos. Os atributos dos elementos XML, nesta linguagem, são traduzidos para

uma lista de pares, o que implica que, as operações a realizar sobre eles obriga a uma pesquisa na lista de atributos desse elemento.

O excerto de XML representado na Figura 2, que traduz uma determinada página Web com informação sobre docentes, poderá ser representado pelo termo da Figura 3, onde o functor da função corresponde à tag do XML. Se pretendermos consultar determinados dados ($email\ (E), office(O)\ e\ name(N)$) nesta página Web, a consulta a implementar é a representada pela Figura 4. As soluções, obtidas por backtracking, são apresentadas na Figura 5. O XML apresentado pode ser facilmente validado por um DTD que define sequências do tipo, $name(N)$, $office(O)$, $email(E)$, onde a informação sobre o $email$ é opcional. Como na informação sobre o docente "Mário Florido" não é apresentado o $email$, este docente não será listado nas soluções da consulta (Figura 5).

```
< teachers >
        < name > Jorge Coelho </name >
        < office > 403 </office >
        < email >
                jcoelho@isep.ipp.pt
        </email >
        < name > Mario Florido </name >
        < office > 202 </office >
        < name > Alan jones </name >
        < office > 214 </office >
        < email > ajones@ncc.up.pt </email >
</teachers >
```

Figura 2 – Documento de XML

```
teachers(name(Jorge Coelho),
        office(403),
        email("jcoelho@isep.ipp.pt"),
        name("Mario Florido"),
        office("202")
        name("Alan Jones"),
        office("204")
        email("ajones@ncc.up.pt"))
```

Figura 3 – Termo com funções de aridade flexível

```
?−teachers(_, name(N), office(O), email(E), _) =*=
        teachers(name(Jorge Coelho),
                office(403),
                email("jcoelho@isep.ipp.pt"),
                name("Mario Florido"),
                office("202")
                name("Alan Jones"),
                office("204")
                  email("ajones@ncc.up.pt"))
```

Figura 4 – Consulta aos dados da página Web

```
N = "Jorge Coelho"
O = "403"
E = "jcoelho@isep.ipp.pt"?;

N = "Alan Jones"
O = "214"
E = ajones@ncc.up.pt?
```

Figura 5 – Soluções para a uma consulta aos dados da página Web

O principal objectivo da linguagem XCentric é tornar o processamento de XML mais simples e intuitivo para programadores que utilizem o Prolog. Isto porque, cada vez mais, o XML é o formato utilizado para a representação de dados em áreas de

aplicação da programação em lógica, como são os casos das bases de dados e *Data Mining*.

3.1.1 Termos com símbolos de aridade flexível e variáveis de sequência

A ideia implícita no XCentric é estender o Prolog com termos com símbolos de aridade flexível e variáveis de sequência. Nas secções seguintes serão descritas a sintaxe e a semântica desta linguagem, apresentando definições e exemplos com base em [Coelho J. e Florido M., 2004].

Definição 3.1.1.1 – uma sequência t é definida da seguinte forma

- ε é uma sequência vazia
- t_1, t é uma sequência se t_1é um termo e t é uma sequência

Exemplo 3.1.1.1 – Dados os termos $f(a), b\ e\ X$, então $t = f(a), b, X$ é uma sequência

Duas árvores são iguais se, e apenas se, os seus functores raiz forem os mesmos e as suas sub-árvores correspondentes forem iguais.

Em [Coelho J. e Florido M., 2004] é proposto um alfabeto constituído por um conjunto de variáveis standard, um conjunto de variáveis de sequência, um conjunto de constantes, um conjunto de símbolos de funções com aridade fixa e um conjunto de símbolos de funções com aridade flexível.

Definição 3.1.1.2 – O conjunto de termos do alfabeto é o mais pequeno conjunto de termos que satisfaça as seguintes condições:

1. As constantes, variáveis standard e variáveis de sequência são termos.
2. Se f é uma função de aridade flexível e $t_1, \ldots, t_n (n \geq 0)$ são termos, então $f(t_1, \ldots, t_n)$ é um termo.
3. Se f é uma função de aridade fixa, com aridade $n, n \geq 0$ e $t_1, \ldots, t_n$ são termos, tal que para todo $1 \leq i \leq n$, t_i não contém variáveis de sequência como subtermos, então $f(t_1, \ldots, t_n)$ é um termo.

Termos do tipo $f(\mathrm{t}_1, \ldots, \mathrm{t}_\mathrm{n})$ em que f é um símbolo de função e $t_1, \ldots, t_n$ são termos, são chamados de termos compostos.

Definição 3.1.1.3 – Se t_1 e t_2 são termos, então $t_1 = t_2$ (unificação em Prolog) e $t_1 =*= t_2$ (unificação de termos com som símbolos de aridade flexível) são restrições.

Uma restrição $t_1 =*= t_2$ ou $t_1 = t_2$ tem solução se, e apenas se, existir uma atribuição de sequências ou termos instanciados, respectivamente, a variáveis cujo resultado da restrição seja verdadeiro, ou seja, após essa atribuição os termos tornam-se iguais.

Em [Coelho J. e Florido M., 2004] assume-se que o domínio de interpretação das variáveis é determinado pelo contexto em que as mesmas ocorrem. Isto quer dizer que, variáveis que surjam em restrições do tipo $t_1 =*= t_2$ são interpretadas no domínio de sequências de árvores, caso contrário são vistas com variáveis standard do Prolog.

Em [Coelho J. e Florido M., 2004] a proposta apresentada baseia-se no algoritmo proposto por [Kutsia T., 2002], em que, a partir das noções fundamentais da teoria da unificação, é possível fazer a substituição de variáveis de sequência por sequências de termos. Tendo por base esta noção de substituição, substituições mais gerais e unificador, [Kutsia T., 2002] define uma nova noção de conjunto mínimo completo de unificadores de uma equação E, $(MCU(E))$ como um conjunto mínimo de substituições respeitando o conjunto de variáveis da equação E:

1. Todas as substituições em $MCU(E)$ são um unificador de E.
2. Para qualquer unificador $\tau \in E$, existe um $\theta \in MCU(E)$, tal que θ é mais geral do que τ
3. Para todo o $\theta, \tau \in MCU(E)$, se θ é mais geral do que τ então $\theta = \tau$

Exemplo 3.1.1.2 – Dada a variável de sequência $\mathrm{X}, \mathrm{f(a, X, c, d)}$ é um termo de aridade flexível. X pode ser instanciado para os termos $\mathrm{f(a, a, a, c, d)}$ ou $\mathrm{f(a, c, d)}$, em que $\mathrm{X = a, a}$ e $\mathrm{X = \varepsilon}$, respectivamente.

Exemplo 3.1.1.3 – O conjunto mínimo completo de unificadores para a equação $f(g(a,X), g(Y,c)) =*= f(U, g(b,V))$ é representado por:

$$\{\{U = g(a,X), Y = b, V = c\}, \{X = \varepsilon, U = g(a), Y = b, V = c\}, \{U = g(a,X), Y = b, Y, V = Y, c\}, \{X = \varepsilon, U = g(a), Y = b, Y, V = Y, c\}\}$$

Os programas em XCentric têm uma sintaxe semelhante à do Prolog com a adição de uma nova restrição $=*=$.

3.1.1.1 Resolução de Restrições

As restrições do tipo $\mathrm{t_1 =*= t_2}$ são resolvidas através de uma unificação não standard que calcula o correspondente conjunto mínimo completo de unificadores. Esta unificação não standard é baseado no algoritmo de [Kutsia T., 2002].

Exemplo 3.1.1.4 – Dados os termos $a(X, b, Y)$ e $a(a, b, b, b)$, em que X e Y são variáveis de sequência, a unificação $a(X, b, Y) =*= a(a, b, b, b)$ devolve as seguintes soluções:

1. $X = a$ e $Y = b, b$
2. $X = a, b$ e $Y = b$
3. $X = a, b, b$ e $Y = \varepsilon$

Exemplo 3.1.1.5 – Dados os termos $a(b,X)$ e $a(Y,d)$ em que X e Y são variáveis de sequência, a unificação $a(b,X) =*= a(Y,d)$ devolve duas soluções possíveis:

1. $X = d\ e\ Y = b$
2. $X = N, d\ e\ Y = b, N,$ em que N é uma nova variável de sequência

3.1.2 Processamento de XML no XCentric

Em [Coelho J. e Florido M., 2004] é apresentado um conjunto auxiliar de predicados que permite o processamento de ficheiros XML. Na secção seguinte serão apresentados alguns exemplos utilizando os predicados do XCentric $xml2pro(XMLInput, XmlTerm)$, em que $XMLInput$ representa o documento XML original e $XmlTerm$ irá armazenar o termo correspondente a esse documento, e $pro2xml(XmlTerm, XMLOutput)$, em que $XMLOutput$ representa o documento XML final depois de processado, responsáveis por converterem ficheiros XML em termos e vice-versa, respectivamente, e ainda o predicado $newdoc(Root, Args, Doc)$ em que Doc é um termo com o functor $Root$ e os argumentos $Args$.

3.1.2.1 XML como termos com símbolos de funções de aridade flexível

No XCentric um documento XML é traduzido para um termo com função de aridade flexível. Este termo contém um functor principal, igual a tag raiz, e zero ou mais argumentos. Os atributos dos elementos XML são em [Coelho J. e Florido M., 2004], traduzidos para uma lista de pares $(nome, Valor)$.

Se considerarmos o documento XML representado pela Figura 6, que descreve um livro de endereços, o seu termo equivalente será o apresentado na Figura 7.

```
< addressbook >
        < record >
                < name > John </name >
                < address > New York </address >
                < email > john.ny@mailserver.com </email >
                </record >
                < record >
                < name > Sofia </name >
                < address > Rio de Janeiro </address >
                < phone > 123456789 </phone >
                < email > sofia.brasil@mail.br </email >
        </record >
</addressbook >
```

Figura 6 – XML representativo de um livro de endereços

$$
\begin{aligned}
&addressbook(record(name('Jonh'),\\
&\qquad\qquad address('New\ York'),\\
&\qquad\qquad email('john.ny@mailserver.com')),\\
&\qquad record(name('Sofia\ Ribeiro'),\\
&\qquad\qquad address('Rio\ de\ Janeiro'),\\
&\qquad\qquad phone('123456789'),\\
&\qquad\qquad email('sofia.brasil@mail.br')))
\end{aligned}
$$

Figura 7 – Tradução do XML na notação interna do XCentric

Exemplo 3.1.2.1 – Supondo que Doc representa o documento XML anterior, pretende-se obter a consulta de todas as pessoas que habitam em "Nova York". Esta consulta pode ser obtida através da seguinte restrição:

$$Doc =*= addressbook(\_, record(name(N), adress('New\ York'), \_), \_).$$

As soluções para esta consulta são obtidas por *backtraking*. Para o excerto de XML anterior a solução obtida seria $N = John$.

Exemplo 3.1.2.2 – No livro de endereços, representado pelo XML anterior, existe uma tag que descreve o telefone (*phone*). O objectivo deste exemplo é a criação de um novo documento sem este elemento. Para tal temos de ler todos os registos e ignorar o elemento phone respectivo se este existir. Esta tarefa pode ser efectuada através do código apresentado na Figura 8.

$$
\begin{aligned}
&translate: -xml2pro('adressbook.xml',' adressbook.dtd', Xml),\\
&\qquad process(Xml, NewXml),\\
&\qquad pro2Xml(NewXml,' addressbook2.xml').\\
&\\
&process(A, NewA): -\\
&\qquad findall(Record, records\_without\_phone(A, Record), LRecords),\\
&\\
&\qquad newdoc(addressbook, LRecords, NewA).\\
&\\
&records\_without\_phone(A1, A2): -\\
&\qquad A1\ =*=\\
&addressbook(\_, record(name(N), address(A), \_, email(E)), \_),\\
&\qquad A2\ =\ record(name(N), address(A), email(E)).
\end{aligned}
$$

Figura 8 – Programa para remover o elemento $phone$

Neste exemplo, o predicado $translate/0$ traduz o ficheiro "$adressbook.xml$" num termo XCentric, que vai ser processado pelo predicado $process/2$, que dá origem a um novo termo XCentric e a um novo documento XML "$addressbook2.xml$" com o mesmo DTD. Este novo documento XML já não terá o elemento *phone* na sua estrutura.

Exemplos 3.1.2.3 – Neste exemplo são apresentados dois documentos XML ("$bookstore1.xml$" e "$bookstore2.xml$") que descrevem os livros existentes em duas livrarias. Ambos os documentos possuem o mesmo DTD e podem conter livros semelhantes. Um exemplo destes XML pode ser descrito pela Figura 9.

```
<?xml version = "1.0" encoding = "UTF − 8"?>
< catalog >
        < book number = "1" >
                < name > Art of Computer Programming </name >
                < author > Donald Knuth </author >
                < price > 140 </price >
                < year > 1998 </year >
        </book >
        ...
        < book number = "500" >
                < name >
                        Haskell: The Craft of Functional
                                Programming (2nd Edition)
                </name >
                < author > Simon Thompson </author >
                < price > 41 </price >
                < year > 1999 </year >
        </book >
</catalog >
```

Figura 9 – Documento XML representativo de uma livraria

O código apresentado a seguir descreve uma consulta sobre quais os livros que são mais baratos na livraria 1 (Figura 10).

```
best_prices(B): −
        xml2pro('bookstore1.xml','bookstore1.dtd',T1),
        xml2pro('bookstore2.xml','bookstore2.dtd',T2),
        process(T1,T2,B).

process(Books1,Books2,[N,A]): −
        Books1 =∗
        = catalog(_,book(name(N),author(A),
        price(P1),year(Y)),_),
        Books2 =∗
        = catalog(_,book(name(N),author(A),
        price(P2),year(Y)),_),
        atom2number(P1,P1f),
        atom2number(P2,P2f),
        P1f < P2f.
```

Figura 10 – Programa para obter os livros mais baratos na livraria 1

O predicado $best\_prices/1$ retorna, por backtracking todos os livros que são mais baratos na livraria 1 por comparação com os livros da livraria 2.

Para obter todos os livros de um determinado autor, o autor de um livro específico ou a combinação autor/livro, podemos implementar o código apresentado na Figura 11.

$$
\begin{aligned}
&books\_from(Author, Book) :- \\
&\quad xml2pro('bookstore1.xml', 'bookstore1.dtd', Xml), \\
&\quad process2(Xml, Author, Book). \\
&process2(Xml, Author, Book) :- \\
&\quad Xml =*= \\
&\quad\quad catalog(\_, book((name(Book), author(Author), \_)), \_).
\end{aligned}
$$

Figura 11 – Programa para obter os livros por autor

Neste exemplo o predicado $books\_from/2$ retorna, por *backtracking* todos os pares Autor/Livro da livraria 1.

Os exemplos apresentados nesta secção permitem-nos confirmar a natureza declarativa da sintaxe presente no XCentric.

3.1.3 Implementação

A implementação desta linguagem é composta por três módulos:

1. Tradução de documentos XML para termos
2. Módulo de resolução de restrições
3. Tradução do termo XCentric resultante para um documento XML

Esta implementação é baseada num conjunto de componentes, implementados usando bibliotecas para o SWI-Prolog, para processamento de XML em Prolog. A resolução de restrições é baseada no algoritmo Kutsia [Kutsia T., 2002].

3.1.3.1 Algoritmo de Unificação

O algoritmo de unificação consiste em duas fases principais, Projecção e Transformação. A primeira fase, Projecção, consiste na remoção de algumas variáveis de sequência, de modo a ser possível obter soluções em que essas variáveis são instanciadas por uma sequência vazia. A segunda fase, Transformação, consiste num conjunto de regras onde a unificação não standard é traduzida para unificação standard do Prolog.

Definição 3.1.3.1 – Dados os termos T_1 e T_2 e considerando V um conjunto de variáveis de T_1 e T_2 e A um subconjunto de V. A projecção elimina todas as variáveis de A em T_1 e T_2.

Exemplo 3.1.3.1 – Sendo $T_1 = f(b, Y, f(X))$ e $T_2 = f(X, f(b, Y))$, $V = \{X, Y\}$. A é um elemento do conjunto $\{\{\ \}, \{X\}, \{Y\}, \{X, Y\}\}$. Na fase de Projecção podemos obter os seguintes casos, correspondentes a $A = \{\ \}, A = \{X\}, A = \{Y\}$ *e* $A = \{X, Y\}$:

- $T_1 = f(b, Y, f(X)), T_2 = f(X, f(b, Y))$
- $T_1 = f(b, Y, f), T_2 = f\big(f(b, Y)\big)$
- $T_1 = f\big(b, f(X)\big), T_2 = f\big(X, f(b)\big)$
- $T_1 = f(b, f), T_2 = f(f(b))$

Na versão deste algoritmo apresentada em [Coelho J. e Florido M., 2004] é usado um tipo especial destes termos a que se chamou termos de sequência para representar sequencias de argumentos.

Definição 3.1.3.2 – Um termo de sequência $\bar{t}$ é definido da seguinte forma:

- $empty$ é um termo de sequência
- $seq(t, \bar{s})$ é um termo de sequência se t é um termo e $\bar{s}$ é um termo de sequência

Definição 3.1.3.3 – Para o termo de sequência $\bar{t}$, o comprimento de $\bar{t}$ é:

$$length(empty) = 0$$

$$length\big(seq(A, B)\big) = 1 + length(B)$$

Definição 3.1.3.4 – Um termo de sequência na forma normal é definido como:

- $empty$ está na forma normal
- $seq(t_1, t_2)$ está na forma normal se t_1 não é da forma $seq(t_3, t_4)$ e t_2 está na forma normal

Exemplo 3.1.3.2 – Para o símbolo de função f , a variável X e as constantes a e b, o seguinte termo representa um termo de sequência na forma normal.

$$seq\left(f\big(seq(a, empty)\big), seq\big(b, seq(X, empty)\big)\right)$$

Os termos de sequência são representados por listas e os termos de sequência na forma normal são listas simples. Esta notação é introduzida de modo diferente em [Coelho J. e Florido M., 2004] porque nesta versão da implementação do algoritmo de unificação, os termos de sequência funcionam como um elemento chave e é importante distingui-los das listas existentes no Prolog. Aqui um termo $f(t_1, t_2, \dots, t_n)$, onde f tem aridade flexível, é representado internamente por $f(seq\big(t_1, seq(t_2, \dots, seq(t_n, empty) \dots)\big)$, o que significa que os argumentos de funções

de aridade flexível são sempre representados como elementos de um termo de sequência.

Definição 3.1.3.5 – Para os termos $\bar{t_1}$ e $\bar{t_2}$, a concatenação de termos de sequência é definida como $\bar{t_1} + + \bar{t_2}$. Este operador define-se da seguinte forma:

$$empty + + \bar{t} = \bar{t}$$
$$seq(t_1, \bar{t_2}) + + \bar{t_3} = seq(t_1, \bar{t_2} + + \bar{t_3})$$

Definição 3.1.3.6 – Para um termo de sequência sua normalização é:

$$normalize(empty) = emtpy$$
$$normalize(t) = seq(t, empty), se\ t\ \text{é}\ uma\ constante\ ou\ variável$$
$$normalize(t) = seq\big(f\big(normalize(t_1)\big), empty\big), se\ t = f(t_1)$$
$$normalize\big(seq(t_1, \bar{t})\big) = normalize(t_1) + + normalize(\bar{t})$$

Em [Coelho J. e Florido M., 2004] as regras de transformação são definidas através do sistema de reescrita, representado pela Figura 12. As letras maiúsculas representam as variáveis de sequência, as letras minúsculas representam os termos e as letras acentuadas representam os termos de sequência. Estas regras representam o algoritmo de Kutsia que utiliza a unificação padrão existente no Prolog. As regras 6, 7, 8 e 9 são não determinísticas, por exemplo a regra 6 diz que para resolver $seq(X, \bar{t}) = *= seq(s\_1, \bar{s})$ podemos resolver através de $\bar{t} = *= \bar{s}$, em que $X = s_1$, ou então através da função de normalização $normalize(seq(X\_1, \bar{t})) = *= normalize(\bar{s})$, em que $X = seq(s\_1, seq(X\_1, empty))$. No final as soluções obtidas pelo algoritmo são normalizadas através da função $normalize$. Quando nenhuma das regras se aplica o algoritmo falha. No algoritmo de Kutsia é mostrado que o algoritmo termina se é observada a existência de ciclos e se cada variável de sequência não ocorre mais do que duas vezes num problema de unificação. Na versão do algoritmo, apresentada em [Coelho J. e Florido M., 2004], à semelhança do algoritmo de Kutsia, também existe a limitação das variáveis de sequência que ocorrem mais do que duas vezes num problema de unificação, no entanto a verificação de existência de ciclos não foi implementada por ser utilizado o backtracking do Prolog para obter todas as soluções.

Sucesso

(1) $t =*= s \Rightarrow Verdade, se\ t == s$

(2) $X =*= t \Rightarrow X = t\ se\ X\ não\ ocorre\ em\ t$

(3) $t = X \Rightarrow X = t\ se\ X\ não\ ocorre\ em\ t$

Eliminação

(4) $f(\bar{t}) =*= f(\bar{s}) \Rightarrow \bar{t} =*= \bar{s}$

(5) $seq(t_1, \bar{t}) =*= seq(s_1, \bar{s}) \Rightarrow t_1 =*= s_1,$
$normalize(\bar{t}) =*= normalize(\bar{s})$

(6) $seq(X, \bar{t}) =*= seq(s_1, \bar{s}) \Rightarrow X = s_1, se\ X\ não\ ocorre\ em\ s_1,$
$normalize(\bar{t}) =*= normalize(\bar{s})$
$\Rightarrow X = seq(s_1, seq(X_1, \varepsilon)),$
$se\ X\ não\ ocorre\ em\ s_1$
$normalize(seq(X_1, \bar{t})) =*= normalize(\bar{s}),$
$em\ que\ X_1 é\ uma\ nova\ variável$

(7) $seq(t_1, \bar{t}) =*= seq(X, \bar{s}) \Rightarrow X = t_1, se\ X\ não\ ocorre\ em\ t_1,$
$normalize(\bar{t}) =*= normalize(\bar{s}),$
$\Rightarrow X = seq(t_1, seq(X_1, \varepsilon)),$
$se\ X\ não\ ocorre\ em\ t_1$
$normalize(\bar{t}) =*= normalize(seq(X_1, \bar{s}))$
$em\ que\ X_1 é\ uma\ nova\ variável$

(8) $seq(X, \bar{t}) =*= seq(Y, \bar{s}) \Rightarrow X = Y$
$\bar{t} =*= \bar{s}.$
$\Rightarrow X = seq(Y, seq(X_1, \varepsilon)),$
$normalize(seq(X_1, \bar{t})) =*= normalize(\bar{s}),$
$em\ que\ X_1 é\ uma\ nova\ variável\ e\ X, Y\ são\ distintos$
$\Rightarrow Y = seq(Y, seq(Y_1, \varepsilon)),$
$normalize(\bar{t}) =*= normalize(seq(X_1, \bar{s}))$
$em\ que\ Y_1 é\ uma\ nova\ variável\ e\ X, Y\ são\ distintos$

Separar

(9) $seq(t_1, \bar{t}) =*= seq(s_1, \bar{s}) \Rightarrow Se\ t_1 =*= s_1 \Rightarrow r_1 =*= q_1, então$
$normalize(seq(r_1, \bar{t})) =*= normalize(seq(q_1, \bar{s}))$
.
.
.
$\Rightarrow Se\ t_1 =*= s_1 \Rightarrow r_w =*= q_w,$
$normalize(seq(r_w, \bar{t_n})) =*= normalize(seq(q_w, \bar{s})),$
$em\ que\ t_1\ e\ s_1\ são\ termos\ compostos$

Figura 12 – Regras de Transformação (adaptado de [Coelho J. e Florido M., 2004])

A seguir serão apresentados, com base em [Coelho J. e Florido M., 2004], alguns exemplos de aplicação do algoritmo implementado.

Exemplo 3.1.3.3 – Para $t = f(X, b, Y)$ e $s = f(c, c, b, b, b, b)$ a fase de projecção obtém os seguintes casos de transformação:

- $f(X, b, Y) =*= f(c, c, b, b, b, b)$
- $f(b, Y) =*= f(c, c, b, b, b, b)$
- $f(X, b) =*= f(c, c, b, b, b, b)$
- $f(b) =*= f(c, c, b, b, b, b)$

Através das regras de transformação podemos ver que a primeira e terceira regras são as únicas que vão ser bem sucedidas.

Para $\mathrm{f(X, b, Y)} =*= \mathrm{f(c, c, b, b, b, b)}$ temos as seguintes soluções:

- $X = c, c\ e\ Y = b, b, b$
- $X = c, c, b\ e\ Y = b, b$
- $X = c, c, b, b\ e\ Y = b$

Para $\mathrm{f(X, b)} =*= \mathrm{f(c, c, b, b, b, b)}$ temos as seguintes soluções:

- $X = c, c, b, b, b, b$
- $Y = \varepsilon$

Exemplo 3.1.3.4 – Para $\bar{t} = \mathrm{f(b, X, f(X))}$ e $\bar{s} = \mathrm{f(X, f(b, Y))}$. Depois da fase de projecção são obtidas as seguintes equações:

- $f(b, Y, f(X)) =*= f(X, f(b, Y))$
- $f(b, Y, f) =*= f(f(b, Y))$
- $f(b, f(X)) =*= f(X, f(b))$
- $f(b, f) =*= f(f(b))$

Para o primeiro e terceiro casos as soluções obtidas são, respectivamente:

- $X = b, Y$
- $X = b, Y = \varepsilon$

Exemplo 3.1.3.5 – Em alguns casos pode ser obtido um conjunto infinito de soluções para a unificação de dois termos. Por exemplo, para os termos $\mathrm{f(X, a)} =*= \mathrm{f(a, X)}$ as soluções possível são:

- $X = a$
- $X = a, a$
- $X = a, a, a$
- $X = a, a, a, a$
- ...

Para este exemplo o algoritmo Kutsia falhava logo após a verificação de que o problema de unificação se estava a repetir. A primeira solução é $\mathrm{X} = \mathrm{a}$ e a segunda solução é $\mathrm{X} = \mathrm{a}, \mathrm{X}_1$, o que dá origem a um novo problema, que é exactamente igual ao problema original: $\mathrm{X}_1, a =*= a, \mathrm{X}_1$.

A seguir serão apresentados exemplos que ilustram a utilização de termos de sequência para o processamento de XML.

Exemplo 3.1.3.6 – Neste exemplo os caracteres <> representam o termo de sequência vazia e $< \mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{n} >$ representa um termo de sequência na sua forma normal, $seq(t\_1, seq(t\_2, \ldots, seq(t\_n, empty) \ldots)$, de modo a simplificar a representação das sequências. Aqui serão usados dois ficheiros XML: o ficheiro text.xml de acordo com o DTD da Figura 13 e o ficheiro bib.xml com o DTD da Figura 14.

$$< ! ELEMENT\ ref\ (\#PCDATA) >$$
$$< ! ELEMENT\ b\ (\#PCDATA) >$$
$$< ! ELEMENT\ text\ (\#PCDATA \mid ref \mid b) *>$$

Figura 13 – DTD para o ficheiro text.xml ([Coelho J. e Florido M., 2004])

$$< ! ELEMENT\ bib\ (author, name) >$$
$$< ! ELEMENT\ bibliography\ (bib+) >$$
$$< ! ELEMENT\ author\ (\#PCDATA) >$$
$$< ! ELEMENT\ name\ (\#PCDATA) >$$

Figura 14 – DTD para o ficheiro bib.xml ([Coelho J. e Florido M., 2004])

No ficheiro $\mathrm{text.xml}$ os elementos representados por $< ref >$ contêm o nome dos autores que surge no documento bib.xml como conteúdo do elemento $< author >$. O objectivo deste exemplo é a geração de dois novos documentos, o ficheiro $\mathrm{text2.xml}$, em que o elemento $< ref >$ é substituído por um novo elemento $< i >$ cujo conteúdo é um número que representa um índice para esta referência ordenada por autor, e o ficheiro $\mathrm{bib2.xml}$" que irá conter todas as referências que aparecem no ficheiro $\mathrm{text.xml}$ ordenadas por autor, com um novo elemento $< label >$ que corresponde ao número que aparece no ficheiro $\mathrm{text2.xml}$. O programa que executa todas estas operações é representado na Figura 15.

```
run: −
        xml2pro('text.xml','text.dtd',T),
        process(T).
process(text(T)): −
        process2(T,T2,[ ],Refs),
        xml2pro('bib.xml','bib.dtd',B),
        add_bib(Refs,B,BibXML,1),
        newdoc(bibliography,BibXML,NewBIB),
        pro2xml(text(T2),'text2.xml'),
        pro2xml(NewBIB,'bib2.xml').
process2(<>,<>,L,L).
process2(A,B,L,RRefs): −
        A =*= <X,ref(R),Y>,
        B =*= <X,i(NewVar),Y2>,!,
        insert_sorted(R,NewVar,L,Refs),
        process2(Y,Y2,Refs,RRefs).
process2(S,S,L,L).
add_bib([],_,[],_).
add_bib([(A,AI)|Refs],B,[bib(label(AI),author(A),name(N))|XML],I): −
        B =*= bibliography(_,bib(author(A),name(N)),_),
        I1 is I + 1,
        atom_chars(I,AI),
        add_bib(Refs,B,XML,I1).
```

Figura 15 – Programa para comparação dos dados entre dois ficheiros XML

No programa, o predicado $process2$ traduz o texto original num novo texto onde os elementos $< ref >$ são substituídos pelos novos elementos $< i >$ e o conteúdo por uma variável. Nesta altura o conteúdo de $< ref >$ e a nova variável em $< i >$ são inseridos numa lista ordenada pelo conteúdo de $< ref >$, dando origem a uma lista ordenada de autores. Finalmente, o predicado add_bib faz uma consulta do ficheiro bib.xml, retornando as referências encontradas no texto e substituindo o conteúdo das variáveis pelo conteúdo final.

Se o ficheiro text.xml for representado pelo XML da Figura 16 e o ficheiro bib.xml representado pelo XML da Figura 17, depois da execução do programa anterior os ficheiros de output gerados serão os das figuras 18 e 19.

```
<text>
        Mainstream languages for <b>XML</b> processing such as XSLT
        <ref>W3C</ref>, XDuce <ref>Hosoya</ref>, CDuce <ref>
                Frish Casagna and Benzaken
        </ref> and Xtatic <ref>Pierce</ref> rely
        on the notion of trees with an arbitrary number of leaf nodes
        to abstract <b>XML</b> documents.
</text>
```

Figura 16 – XML representativo do ficheiro text.xml

```
< bibliography >
        < bib >
                < author > Coelho and Florido </author >
                < name > Type − based XML Processing in Logic Programming  </name >
        </bib >
        < bib >
                < author > W3C </author >
                < name > XSL Transformations (XSLT). http://www.w3.org/TR/xslt/</name >
        </bib >
        < bib >
                < author > Hosoya </author >
                < name > XDuce: A Typed XML processing language </name >
        </bib >
        < bib >
                < author > Frish Casagna and Benzaken </author >
                < name > CDuce an XML − centric general − purpose language </name >
        </bib >
        < bib >
                < author > Pierce </author >
                < name > Xtatic. http://www.cis.upenn.edu/~bcpierce/xtatic/</name >
        </bib >
</bibliography >
```

Figura 17 – XML representativo do ficheiro bib.xml

```
< text >
        Mainstream languages for  < b > XML </b >  processing such as XSLT
        < i > 4 </i >, XDuce  < i > 2 </i >, CDuce  < i > 1 </i >
        and Xtatic  < i > 3 </i >  rely on the notion of trees with an arbitrary
        number of leaf nodes to abstract  < b > XML </b >  documents.
</text >
```

Figura 18 – XML representativo do ficheiro text2.xml

```
< bibliography >
        < bib >
        < label > 1 </label >< author >  Frish Casagna and Benzaken </author >
        < name > CDuce an XML − centric general − purpose language </name >
        </bib >
        < bib >
        < label > 2 </label > < author > Hosoya </author >
        < name >  XDuce: A Typed XML processing language </name >
        </bib >
        < bib >
        < label > 3 </label > < author > Pierce </author >
        < name > Xtatic. http://www.cis.upenn.edu/~bcpierce/xtatic/</name >
        </bib >
        < bib >
        < label > 4 </label >< author > W3C </author >
        < name > XSL Transformations (XSLT). http://www.w3.org/TR/xslt/</name >
        </bib >
</bibliography >
```

Figura 19 – XML representativo do ficheiro bib2.xml

Através deste exemplo podemos constatar as vantagens do uso do XCentric no processamento de XML. Através do uso de variáveis de lógica o problema apresentado pode ser resolvido processando o ficheiro text.xml apenas uma vez.

Existem várias linguagens para o processamento de XML e que também utilizam símbolos de função com aridade flexível, mas que se diferenciam, em alguns aspectos da linguagem XCentric. Uma comparação entre elas pode ser analisada no trabalho apresentado em [Coelho J. e Florido M., 2004]. Para além disso, é ainda possível obter uma comparação entre o algoritmo proposto em [Kutsia T., 2002] e o proposto em [Coelho J. e Florido M., 2004].

3.2 VeriFLog

O principal objectivo deste framework [Coelho J. e Florido M., 2006b] é fornecer uma interface para dados semi-estruturados e validação sintáctica utilizando a linguagem XCentric para a verificação semântica do conteúdo de páginas Web. A verificação semântica consiste na verificação de que o conteúdo de uma página Web está correcto de acordo com determinados critérios.

Esta framework permite ainda, ao programador, fazer uso de todas as potencialidades da linguagem XCentric, para inferir conhecimento a partir das próprias páginas Web, o que significa dizer que, com base na informação disponibilizada numa determinada página, podem ser gerados novos dados para serem visualizados. Pode também ser usada como uma ferramenta para alguém responsável pela manutenção de páginas Web, de modo a valida-las e corrigi-las de acordo com determinadas restrições impostas.

Nesta secção serão apresentados alguns exemplos de funcionamento desta framework e das três principais regras, que são possíveis de implementar nesta ferramenta; será descrito o comportamento do XCentric como linguagem intermédia e ainda algum trabalho relacionado com o apresentado em [Coelho J. e Florido M., 2006b].

3.2.1 Restrições em Páginas Web

Na framework descrita em [Coelho J. e Florido M., 2006b] é definido que uma regra aplicada a uma página Web de entrada, origina uma nova página Web com as alterações necessárias de modo a obedecer a todas as restrições que foram impostas. Além disso, a consistência de regras é também assegurada, ou seja, a definição de uma regra não pode violar outra regra que já tenha sido definida anteriormente.

Com base nestas definições vão ser apresentados exemplos das três principais regras que a framework permite aplicar:

- $delete(S, WpageI, WpageO, L)$ – A partir de uma página WpageI, esta regra permite remover a sequência S, com base nas restrições especificadas em L, originando uma nova página WpageO.
- $replace(S1, S2, WpageI, WpageO, L)$ - Substitui a sequência S1 pela sequência S2, na página WpageI, se forem verificadas as restrições definidas em L, dando origem a uma nova página WpageO.
- $failure(WebpageI, L, Mesg)$ – Esta regra é utilizada quando o erro existente na página é demasiado grave para ser corrigido automaticamente. Se o erro se verificar é mostrada uma mensagem Mesg ao utilizador.

Exemplos 3.1.2.3 – Tendo por base o documento XML apresentado na Figura 20 que, representa uma página de artigos com referências bibliográficas, pretende-se remover todas as referências $< ref > \cdots </ref >$ que surjam no texto ($Content$) mas que não estejam referenciadas na secção bibliográfica. Este exemplo demonstra a validação sobre os atributos de um determinado elemento, sendo que neste caso pretendemos validar o atributo $number$ do elemento $< bibentry >$. O código que permite a validação e remoção do elemento ref pode ser definido da seguinte forma:

$$delete(ref(R), Wiki1, Wiki2, [not(deep(bibentry([(number, R)], \_), Wiki1))]).$$

A aplicação desta regra irá gerar um novo documento XML armazenado na variável $Wiki2$ (depois de ser transformado, o documento XML será idêntico ao representado na Figura 21) em que a referência com o número 3 ($< r e f > 3 </ r e f >$) deixa de aparecer no conteúdo do texto, uma vez que a mesma não está disponível na secção bibliográfica ($bibentry$). A pesquisa na secção bibliográfica é feita percorrendo a lista de atributos ($[(number, R)]$) deste elemento.

```
< WikiArticle >....
        < Content >
                XCentric < ref > 3 </ref > is an extension of Prolog with unification of terms
                of terms of flexible arity which enables a simpler and high level querying
                and processing of XML data.
        </Content >...
        < References >
                < bibentry number = "1" >
                Jorge Coelho and Mario Florido . XCentric: Logic Programming for XML Processing . 9th ACM
                International Workshop on Web Informat ion and Data Management . ACM Press, 2007.
                </bibentry >
        < bibentry number = "2" > SWI − Prolog, http://www.swi − prolog.org/</bibentry >
        </References >
</WikiArticle >
```

Figura 20 – Documento XML – Página de artigos com referências bibliográficas

```
<? xml version = "1.0" encoding = "utf − 8"? >
< WikiArticle >
        ....
        < Content >
                XCentric is an extension of Prolog with
                unification of terms of flexible arity which enables a simpler
                and high level querying and processing of XML data.
        </Content >
        ...
        < References >
                < bibentry number = "1" >
                        Jorge Coelho and Mario Florido .
                        XCentric: Logic Programming for XML Processing .9th ACM
                        International Workshop on Web Informat ion and Data
                        Management . ACM Press, 2007.
                </bibentry >
                < bibentry number = "2" >
                        SWI − Prolog, http://www.swi − prolog.org/
                </bibentry >
        </References >
</WikiArticle >
```

Figura 21 – Documento XML resultado da aplicação da regra

Exemplo 3.2.1.2 – Com base no XML da Figura 22, que representa um catálogo de livros pretende-se substituir todos os preços que não sejam numéricos por zero. Esta regra é traduzida pelo seguinte código:

$$replace(price(X), price(0), W1, W2, [not(number(X))]).$$

W1 e W2 representam, respectivamente, as páginas Web de entrada e saída e a variável X armazena o conteúdo de $price$ que vai ser validado.

```
< catalog >
        ...
        < book number = "500" >
                < name >
                        Haskell: The Craft of Functional Programming
                        (2nd Edition)
                </name >
                < author > Simon Thompson </author >
                < price > 41 </price >
                < year > 1999 </year >
        </book >
        ...
</catalog >
```

Figura 22 – XML: Catálogo de livros

Exemplo 3.2.1.3 – O XML da Figura 23 representa a página de um determinado professor no site institucional da universidade em que lecciona. Se existir uma restrição que obriga a que o email que é mostrado seja o da universidade e que o

mesmo seja válido, e essa restrição não estiver a ser cumprida, então pretende-se que a página não seja processada. A regra a implementar nesta situação é regra de $Fail$ e pode ser definida pelo código seguinte:

$$failure(T1, [deep(email([], X), T1), \\ not(sub\_string(X, \_, \_, \_, '@isep.ipp.pt'))], 'Valid\ email\ not\ found!').$$

Como no documento anterior o email ($< email > amf </email >$) não segue a restrição imposta, o utilizador será avisado dessa situação com uma mensagem de erro $'Valid\ email\ not\ found!'$.

```
< teacher >
        < name > Mario </name >
        < phone > +351 123456789 </phone >
        < email > amf </email >
        < research >
                < pub >
                < author > Sandra Alves </author >
                < author > Mario Florido </author >
                < title > Weak Linearization of the Lambda − Calculus. </title >
                < booktitle ></booktitle >
                < year > 2010 </year >
                < publisher > Elsevier Science </publisher >
                </pub >
                ...
        </research >
        < teaching >< course > Compilers </course ></teaching >
        < curriculum ></curriculum >
</teacher >
```

Figura 23 – Documento XML: Página de um professor

3.2.2 Verificação estática de regras

Em [Coelho J. e Florido M., 2006b], é proposta uma extensão à linguagem XCentric com tipos, fornecendo um meio simples de verificação, consulta e processamento dos documentos. Como os tipos implementados nesta linguagem não são objecto de estudo neste trabalho, será apenas apresentado um exemplo simples da sua utilização.

Utilizando o XML da Figura 22, e tendo em conta que este é validado com base no DTD apresentado na Figura 24.

```
<? xml version = "1.0" encoding = "UTF − 8"? >
< ! ELEMENT bib (pub ∗) >
< ! ELEMENT pub (author+, title, booktitle, volume? , year? , publisher? ) >
< ! ELEMENT author (#PCDATA) >
< ! ELEMENT title (#PCDATA) >
< ! ELEMENT booktitle (#PCDATA) >
< ! ELEMENT volume (#PCDATA) >
```

$$< !ELEMENT\ year\ (\#PCDATA) >$$
$$< !ELEMENT\ publisher\ (\#PCDATA) >$$

Figura 24 – DTD para o documento XML do catálogo de livros

Pretende-se impor que se o ano da publicação for superior a 2010, a *string* "$to\ appear$" deve aparecer o título do livro, caso não apareça deve ser adicionada. Esta regra pode ser facilmente definida pelo código seguinte.

```
swc(URLPub, NewPub):-
        xml2pro(URLPub, XML),
        bind_type(XML, type_pub),
        r1(XML1, XML2),
        pro2xml(XML2, NewPub).

r1(X1, X2):-
        replace(pub(X, booktitle(BT1), Y),
                pub(X, booktitle(BT2), Y), X1, X2,
                [Y =*=< _, year(Y1), _ >,
                atom2number(Y1, N),
                N > 2010, not(sub_string('to appear', BT1)),
                concat(BT1,' (to appear)', BT2]).
```

Figura 25 – Programa para substituir o conteúdo do título

Como podemos ver, pela análise do código, o documento de XML, depois de traduzido para a notação interna da linguagem XCentric, é carregado para o tipo $type\_pub$ que representa o DTD da Figura 24 ($bind\_type(XML, type\_pub)$). Na altura da compilação é feita uma validação estática de modo a verificar se a regra implementada viola o DTD apresentado. Durante o tempo de execução os tipos voltam a ser usados para verificar se os novos valores, introduzidos pelas regras, não modificam o documento erradamente. É também nesta altura que, se existir mais do que uma regra, é validada a consistência das mesmas.

A verificação estática das regras é feita através da análise estática das regras de acção. Estas regras podem modificar o conteúdo do documento quando aplicam operações de $delete$ ou $replace$. Se existir um tipo que descreve a estrutura do documento, é possível validar se essas transformações não alteraram a estrutura.

Esta verificação estática ocorre durante o processo de compilação do programa. Se existir um tipo definido para um determinado documento a verificação de tipos actua de acordo com as seguintes instruções:

1. Verificar se o tipo definido foi associado ao documento XML através da instrução $bind\_type$.
2. Para qualquer instrução do tipo $delete(s1, XML, XML2, L)$ verificar se a falta do elemento apagado não viola o tipo declarado
3. Para qualquer instrução do tipo $replace(s1, s2, XML, XML2, L)$, se $type(s1) \neq type(s2)$ verificar se a substituição entre os elementos não torna o documento inválido com base no tipo declarado.

Por exemplo pretendermos definir as seguintes regras:

- $delete(name(X), XML1, XML2, [\,])$: o tipo definido estabelece que o elemento $name$ não pode ser removido do documento
- $replace(phone(P), email(\text{"}test@testmail.com\text{"}))$: o tipo definido estabelece que apesar de podermos ter vários emails, o elemento $phone$ não pode desaparecer do documento já que é obrigatório.

No momento em que a verificação de tipos é efectuada, são retornados erros para o utilizador indicando que as regras impostas alteram a estrutura do documento de acordo com o tipo declarado.

O VeriFLog herda do XCentric grande parte das suas características. Um utilizador, experiente em XML, mas pouco à vontade com a programação em lógica pode ter algumas dificuldades na forma como deve efectuar a verificação.

Existem variados trabalhos que propõem uma abordagem, usando programação em lógia, para a aplicação de restrições na criação de páginas Web. Uma comparação entre estas abordagens e a que tem sido descrita nesta secção pode ser obtida através da consulta de [Coelho J. e Florido M., 2006b].

A contribuição de [Coelho J. e Florido M., 2006b] é uma framework para a verificação de páginas Web aplicando os conceitos da linguagem XCentric [Coelho J. e Florido M., 2004] que, como foi descrito nesta secção, é bastante apropriada para esse objectivo. Como conclusão do trabalho apresentado em [Coelho J. e Florido M., 2006b], os autores acreditam que a abordagem que propõem, baseada em programação em lógica, é a mais completa, na medida em que integra uma interface para dados semi-estruturados, validação sintáctica e verificação semântica e tipos.

4. Editor Visual de Regras

O editor visual de regras trata-se de uma aplicação desenvolvida com base na linguagem XCentric [Coelho J. e Florido M., 2004] e [Coelho J. e Florido M., 2007b] e na framework VeriFLog [Coelho J. e Florido M., 2006b]. Este capítulo encontra-se dividido em cinco secções. Na primeira descreve-se o que motivou o desenvolvimento da ferramenta, na segunda quais as principais funcionalidades que foram adicionadas, na terceira quais as linguagens utilizadas nessa implementação e o modo como as mesmas se interligam, na quarta secção será feita uma descrição geral do seu funcionamento e na última secção serão apresentados exemplos ilustrativos de como a aplicação pode ser utilizada.

4.1 Motivação

O facto de aplicações Web se terem tornado cada vez mais complexas, devido a um conjunto de factores já enumerados anteriormente neste documento, e da manutenção dos seus conteúdos ser, geralmente, realizada de forma manual, motivou o aparecimento da framework VeriFLog. No entanto, havia a necessidade de disponibilizar todas as funcionalidades desta ferramenta a um maior número de pessoas, não tão familiarizadas com a linguagem de programação em lógica. Deste modo, para além da ferramenta apresentada disponibilizar todas as funcionalidades já oferecidas pelo VeriFLog, também a estendeu de modo a torná-la melhor e mais robusta.

Muitas das aplicações Web a que hoje temos acesso podem ser construídas por diferentes pessoas em todo o mundo. São páginas colaborativas em que qualquer pessoa pode inserir os seus próprios conteúdos, como é o caso da conhecida *Wikipedia* [9]. A principal aplicação do editor visual de regras é a validação e verificação do conteúdo inserido neste tipo de páginas.

4.2 Principais Funcionalidades

Para além das funcionalidades extraídas do VeriFLog, e que na aplicação são disponibilizadas graficamente, foram ainda adicionadas outras que tornam a utilização deste editor mais robusta e completa:

- Representação gráfica simples de XML e XSD [7] através da utilização de controlos nativos do .NET (*tree view*) para representação de XML e XSD, evitando que o utilizador tenha de conhecer a estrutura dos ficheiros,

facilitando a selecção de nós do XML para serem validados e também a sua visualização

- Inferência de XSD a partir de instâncias de XML. A partir de um ficheiro XML seleccionado inferir o XSD respectivo, tornando a selecção de elementos para validação mais simples e intuitiva, na medida em que, apenas é mostrada a estrutura do documento e não o documento por inteiro.
- Validação de instâncias de XML com base em XSD. Para um dado ficheiro XML ou para uma directoria de ficheiros XML validar se estes respeitam a estrutura com base num XSD específico. Apenas os ficheiros que respeitam esta estrutura serão processados.
- Aplicações de regras a um único ficheiro de XML ou a uma directoria de ficheiros XML. Todas as regras configuradas poderão ser aplicadas a um único ficheiro, ou então a uma directoria que contenha vários ficheiros XML, tornando possível a validação de várias páginas Web, que possuam uma estrutura semelhante.
- Gravação e reutilização de regras. As regras configuradas são guardadas em disco podendo ser utilizadas mais tarde.
- Edição manual de regras. Possibilidade de validação de mais aspectos impossíveis de validar graficamente. Através das restrições que são viáveis de configurar nas janelas da aplicação não é possível definir todos os aspectos que pretendemos validar numa determinada página Web. Para colmatar esta lacuna é possível definir regras manuais. Neste caso o utilizador terá de possuir conhecimento na área da programação em lógica.
- Edição de regras já implementadas. Alteração de regras (manuais e visuais) previamente configuradas. Após uma configuração inicial das regras, o utilizador pode, se assim o pretender, alterar as configurações de uma regra que foi anteriormente definida.
- Eliminação de regras previamente aplicadas. Possibilidade de remover regras que já tenham sido aplicadas. Caso o utilizador entenda que determinada regra, previamente configurada, já não faz sentido, pode simplesmente remove-la da lista de regras a aplicar.

4.3 Implementação

Para a implementação desta ferramenta é utilizada a linguagem orientada a objectos C# [5], por apresentar um grau de simplicidade elevado. Como a framework VeriFLog se encontra implementada em Prolog, para que a aplicação possa beneficiar de todas

as suas potencialidades, é necessária a utilização de uma camada intermédia responsável pela comunicação entre as duas linguagens. Tal é conseguido através de uma biblioteca “Swi-cs-pl” [4] disponibilizada por terceiros através do SWI-Prolog [3], que é capaz de executar predicados e instruções Prolog a partir do C#.

Esta comunicação é feita essencialmente através da manipulação de *strings*, ou seja, todos os comandos Prolog que se pretendem executar são convertidos em *strings* no C#, para que depois possam ser entendidos pela biblioteca e consequentemente executados. Por exemplo, se quisermos adicionar um predicado à base de conhecimento, o comando Prolog para esta instrução será algo como o indicado na figura seguinte:

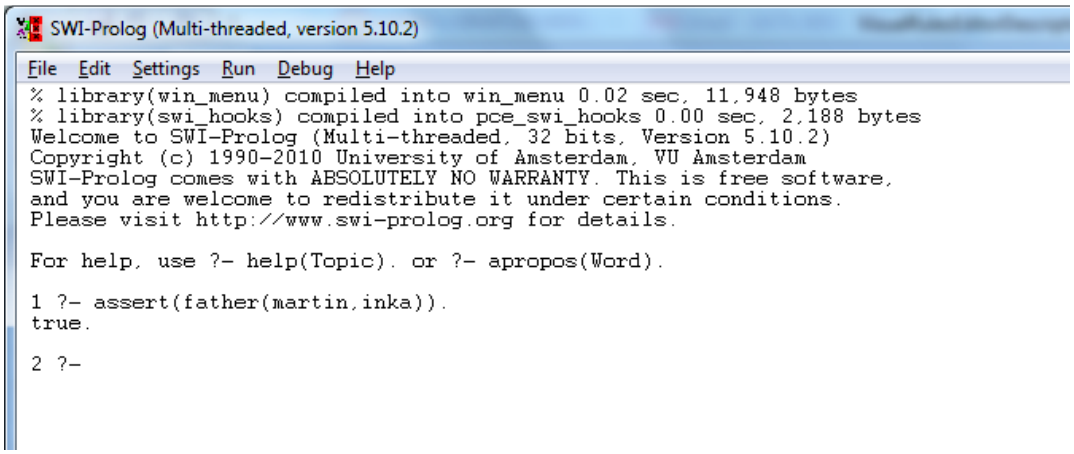

Figura 26 - Comando Prolog para adicionar um predicado à base de conhecimento

A mesma instrução, executada através da biblioteca intermédia “Swi-cs-pl”, terá de ser convertida para *string* para que possa ser passada como argumento para o construtor da classe: $PlQuery.PlCall("assert(father(martin,inka))")$;

A figura seguinte apresenta um exemplo de um programa que utiliza esta biblioteca para a comunicação do Prolog com o C#. Aqui é feita a construção da base de conhecimento onde são definidos os graus de parentesco entre várias pessoas. Depois é listada para o ecrã uma lista que indica o pai de cada pessoa e ainda uma lista de todos os filhos de uma única pessoa (Figura 28).

```
using System;
using SbsSW.SwiPlCs;
namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            if (!PlEngine.IsInitialized)
            {
                String[] param = { "-q" };
                PlEngine.Initialize(param);
                PlQuery.PlCall("assert(father(martin, inka))");
                PlQuery.PlCall("assert(father(uwe, gloria))");
                PlQuery.PlCall("assert(father(uwe, melanie))");
                PlQuery.PlCall("assert(father(uwe, ayala))");
                using (PlQuery q = new PlQuery("father(P, C),
                            atomic_list_concat([P,' is_father_of ',C], L)"))
                {
                    foreach (PlQueryVariables v in q.SolutionVariables)
                        Console.WriteLine(v["L"].ToString());

                    Console.WriteLine("all child's from uwe:");
                    q.Variables["P"].Unify("uwe");
                    foreach (PlQueryVariables v in q.SolutionVariables)
                        Console.WriteLine(v["C"].ToString());
                }
                PlEngine.PlCleanup();
                Console.WriteLine("finshed!");
            }
        }
    }
}
```

Figura 27 - Exemplo de programa C# com utilização da biblioteca Swi-cs-pl

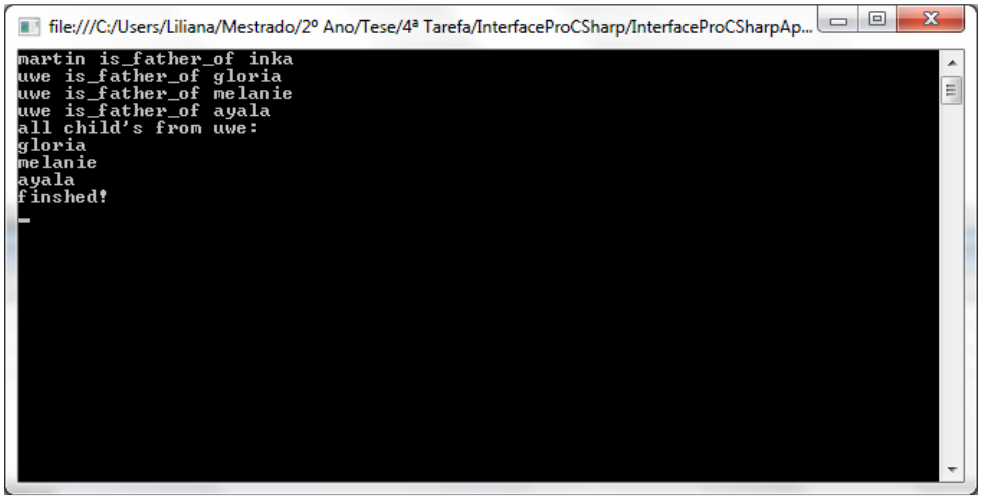

Figura 28 – Resultado da execução do programa C# utilizando a biblioteca Swi-cs-pl

Esta biblioteca permite ainda a execução de ficheiros .pl que contenham o código que se pretende executar. Isto quer dizer que podemos criar uma programa e guardá-lo

num ficheiro com a extensão pl e depois executa-lo a partir do C# sem haver a necessidade de construir o programa todo do lado do C#. Se, por exemplo, tivermos um ficheiro "family.pl" que contém todas as instruções do exemplo anterior, o programa C# para o executar pode ser descrito da seguinte forma:

```
using System;
using SbsSW.SwiPlCs;

namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            if (!PlEngine.IsInitialized)
            {
                String[] param = { "-q" };
                PlEngine.Initialize(param);
                PlQuery.PlCall("consult('family.pl')");
                using (PlQuery q = new PlQuery ("father(P, C),
                       atomic_list_concat([P,' is_father_of ',C], L)"))
                {
                foreach (PlQueryVariables v in q.SolutionVariables)
                Console.WriteLine(v["L"].ToString());
                Console.WriteLine("all child's from uwe:");
                q.Variables["P"].Unify("uwe");
                foreach (PlQueryVariables v in q.SolutionVariables)
                Console.WriteLine(v["C"].ToString());
                }
                PlEngine.PlCleanup();
            }
        }
    }
}
```

Figura 29 - Exemplo de programa C# para execução de ficheiros .pl

4.4 Descrição da Aplicação

O editor visual de regras é uma aplicação em janelas, onde o utilizador pode, graficamente, configurar todas as regras, baseadas em restrições, que pretende aplicar a um ou vários documentos XML representativos de páginas Web.

O ecrã inicial da aplicação, representado pela Figura 30, é o ponto de partida para a configuração e execução dessas regras.

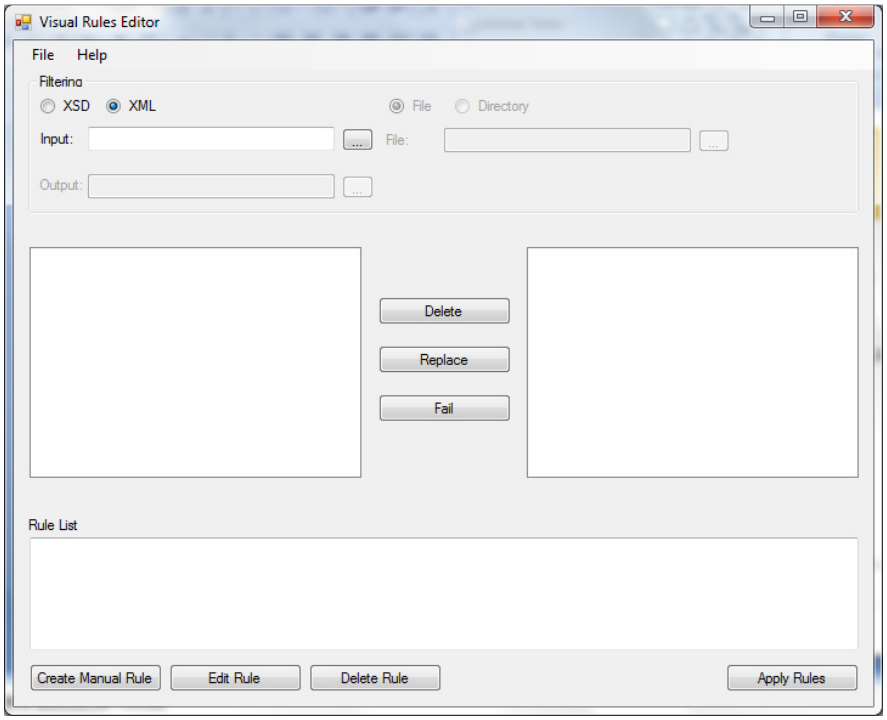

Figura 30 – Ecrã inicial da aplicação

Neste ecrã o utilizador pode seleccionar entre dois tipos de ficheiros, XSD ou XML. No caso de seleccionar a opção XML, a própria aplicação irá inferir o XSD referente a esse XML e carrega-lo no ecrã sob a forma de uma árvore, tornando a selecção dos elementos mais simples e rápida, já que na janela apenas será mostrada a estrutura do ficheiro e não o ficheiro completo. Se o utilizador seleccionar a opção XSD, terá também de especificar o ficheiro (*File*) ou a directoria de ficheiros (*Directory*) sobre os quais são aplicadas as regras. Nesta última opção, todos os ficheiros da directoria serão validados face ao XSD seleccionado e caso exista algum que não respeite essa estrutura não será considerado na aplicação final das regras.

Depois da selecção anterior, o utilizador pode passar à configuração efectiva das regras. O ecrã de configuração das regras é acessível através dos botões situados no centro da janela principal da aplicação sendo carregado de acordo com o tipo de operação seleccionado ($Delete$, $Replace$ e $Fail$). Cada um desses botões corresponde a uma das funcionalidades disponíveis na framework:

- *Delete* – Remoção de determinados elementos do ficheiro de XML se o conteúdo dos mesmos não respeitar as restrições definidas pelo utilizador
- *Replace* – Substituição do conteúdo se este não passar nas validações impostas pelo utilizador
- *Fail* – Rejeição total do ficheiro de XML se o mesmo não respeitar certas imposições do utilizador

Para cada uma das regras anteriores é possível definir um conjunto de restrições utilizadas para efectuar as validações pretendidas. Na Figura 31 (ecrã de configuração das regras), no painel de restrições ($Constraints$), estão representadas todas as restrições possíveis de configurar para cada uma das regras. A restrição NOT é a única que não é exclusiva em relação às outras, já que se pode seleccionar em conjunto com uma outra ($Contains$, $Contains\ Valid\ URL$, $NULL$). No entanto, nenhuma destas últimas pode ser seleccionada em conjunto.

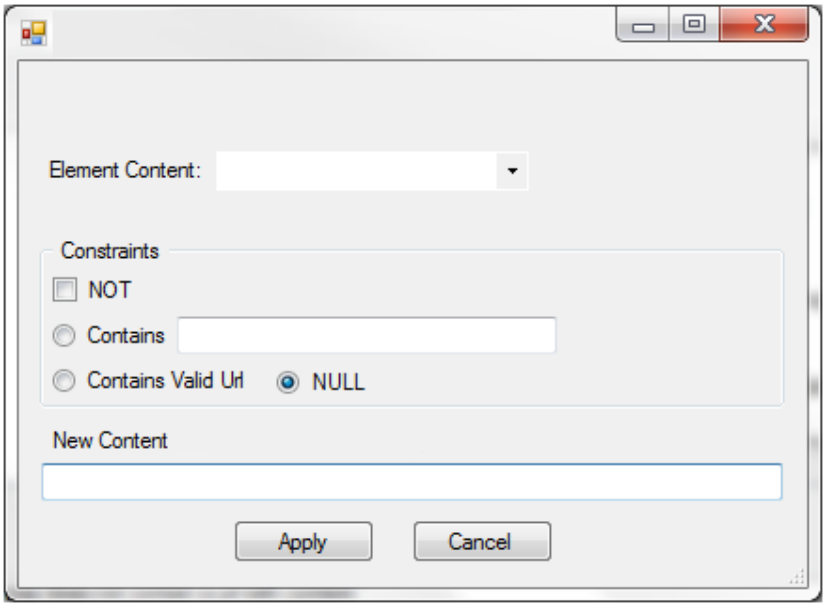

Figura 31 – Ecrã de configuração das regras

Cada uma destas funcionalidades será descrita em pormenor nas secções seguintes. Além destas funcionalidades, foram adicionadas outras que tornam a utilização desta aplicação mais robusta, como:

- Edição de regras manuais, assumindo-se, neste caso, que o utilizador está familiarizado com a linguagem Prolog. Através de uma janela, onde é possível inserir linhas de código, o utilizador pode configurar restrições que não é possível definir através dos componentes gráficos da aplicação.
- Alteração das regras que foram definidas visualmente e manualmente. A qualquer altura do processamento o utilizador pode optar por alterar as configurações das regras que já foram aplicadas ao documento.
- Gravação e Reutilização de regras. Todas as regras vão sendo gravadas em disco, sendo possível a sua utilização posterior.
- Eliminação de regras já aplicadas. Todas as regras que já foram aplicadas ao documento XML podem ser removidas. A remoção de uma regra implica a não aplicação dessa regra ao documento.

A selecção inicial terá influência no número de ficheiros output gerados depois da aplicação final das regras. Se a opção seleccionada for a opção XML, será apenas gerado um ficheiro XML de output, mas se opção seleccionada for a opção XSD, poderá ser gerado apenas um ($File$) ou vários ($Directory$) ficheiros de output.

4.5 Exemplos

Esta secção pretende mostrar o funcionamento da aplicação desenvolvida. Em cada uma das subsecções que a compõem, serão apresentados exemplos de cada uma das funcionalidades oferecidas pela ferramenta, bem como a descrição da sua implementação.

Para a apresentação dos exemplos será utilizada uma wikipage [8] que representa as tradicionais e bem conhecidas páginas amarelas em papel. Nesta *wikipage* qualquer pessoa pode contribuir inserindo a descrição do seu negócio.

Um XML que pode descrever a estrutura desta *wikipage* é o apresentado na Figura 32. Será com base neste XML que serão configuradas todas as regras de exemplo.

```
< yellowpage >
    < name > La Lanterna di Vittorio </name >
    < category > Cafe/ Wine bar </category >
    < addressdirections >
        < Address > 129 MacDougal St.</Address >
        < map >< label > Map it </label >< url ></url ></map >
    </addressdirections >
    < contact >
        < phone > (212) 529 − 5945 </phone >
        < fax > (212) 981 − 2731 </fax >
        < email > La.Lanterna verizon.net </email >
    </contact >
    < url > http://lalanternacaffe.com/ </url >
    < hours > Fri − Sat 10: 00am − 4: 00am Sun − Thu 10: 00am − 3: 00am </hours >
    < description > Cafe/ wine bar serving a wide selection of wines,… </description >
    < prices > </prices >
    < neighborhood > West Village </neighborhood >
    < gallery >
        < photo >
            < url > http://www.wikipages.com/images/3/38/Lalatterna.jpg < /url >
            < label > La Lanterna di Vittorio </label >
        </photo >
    </gallery >
    < reviews >
        http://nyc.metblogs.com/archives/2007/05/wine_in_the_gar.phtml
    </reviews >
</yellowpage >
```

Figura 32 – XML representativo da wikipage

4.5.1 Validação de XML com base no XSD

A validação de ficheiros XML, com base no XSD, é feita quando é seleccionada a opção XSD no lado esquerdo do ecrã (Figura 33). Nesta situação o utilizador terá ainda de escolher, no lado direito do ecrã, entre as opções File ou Directory. Em qualquer um dos casos será verificado se o XML respeita ou não a estrutura do XSD seleccionado.

Após a selecção do ficheiro XML ou da directoria de ficheiros XML, é apresentada uma janela que indica quais os ficheiros que respeitam a estrutura do XSD e quais os que não respeitam (Figura 34). Apenas os ficheiros que, sigam a estrutura do XSD seleccionado, serão processados aquando da aplicação das regras que o utilizador irá definir.

Imagine-se que o utilizador selecciona a opção File (selecção de um único ficheiro) e escolhe o ficheiro "$WikiArticle.xml$", representado pela Figura 35. Comparando este ficheiro com o XSD [7] é possível verificar que o XML não segue a estrutura do XSD e como tal surgirá na lista de ficheiros inválidos e não será processado (Figura 34).

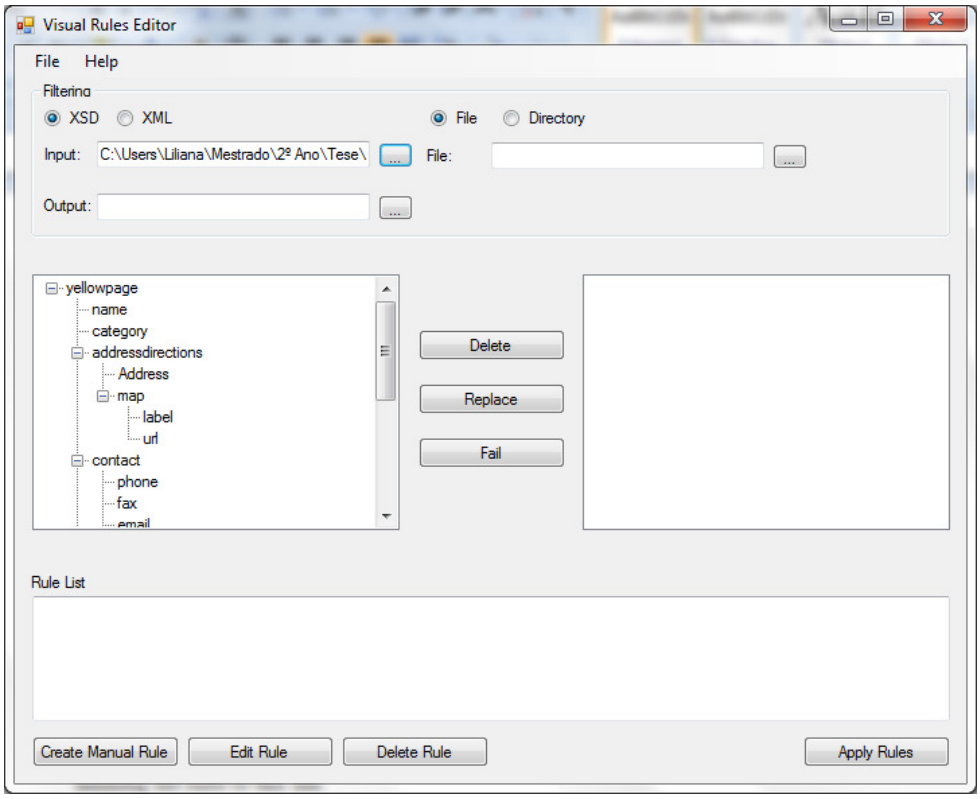

Figura 33 – Árvore com XSD carregado

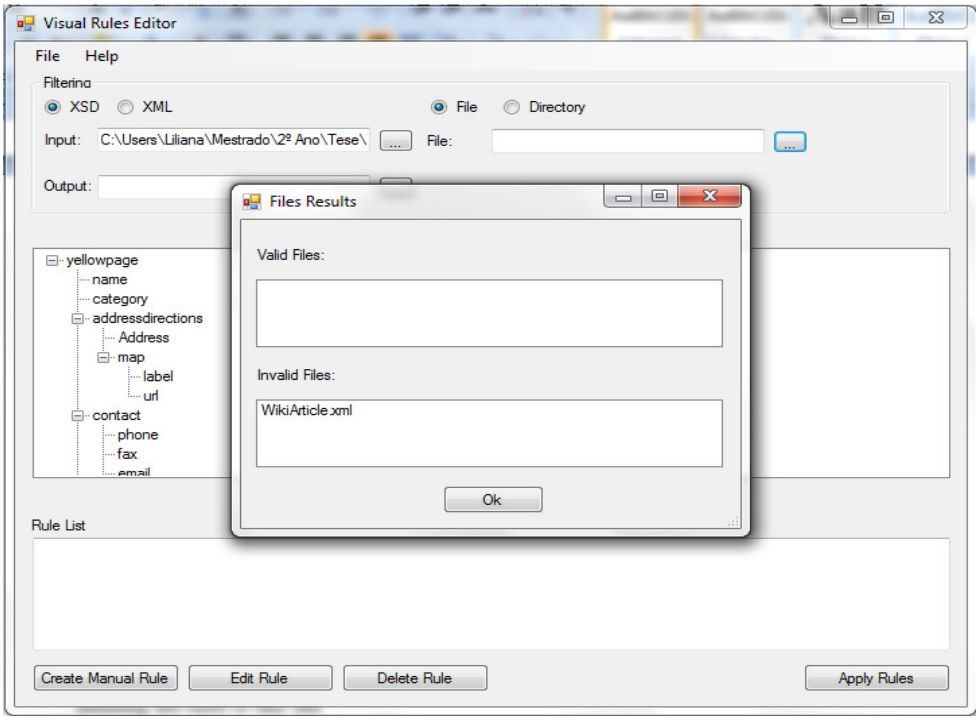

Figura 34 – Validação de ficheiros XML para um único ficheiro XML

```
< WikiArticle >
        < Content >
                XCentric < ref > 3 </ref > is an extension of Prolog with
              unification of terms of flexible arity which enables a simpler
              and high level querying and processing of XML data.
        </Content >
        < References >
                < bibentry  number = "1" > Ad Sales
                        Jorge Coelho and Mario Florido.
                        XCentric: Logic Programming for XML Processing. 9th ACM
                        International Workshop on Web Information and Data
                        Management. ACM Press, 2007.
                </bibentry >
                < bibentry  number = "2" >
                        SWI − Prolog, http://www.swi − prolog.org/
                </bibentry >
        </References >
</WikiArticle >
```

Figura 35 – Exemplo XML que não respeita o XSD

Se em vez do utilizador seleccionar a opção *File* seleccionar a opção *Directory*, a aplicação apresentará uma lista onde apresenta os ficheiros válidos (respeitam a estrutura do XSD) e inválidos (não respeitam a estrutura do XSD) que existem na directoria escolhida (Figura 36).

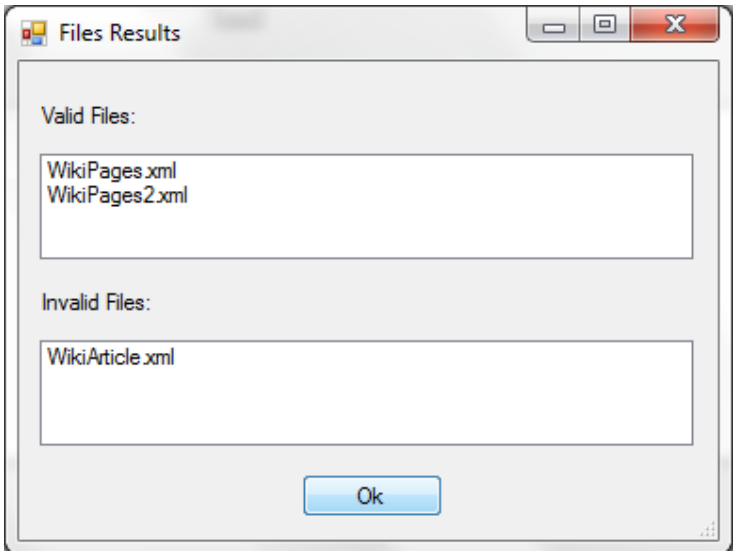

Figura 36 – Validação de ficheiros XML para uma directoria

4.5.1.1 Implementação

A implementação desta funcionalidade foi feita através de um método que recebe dois ficheiros como parâmetro, o ficheiro XML de input e o ficheiro XSD Figura 37. Através de classes de manutenção de XML existentes no .NET é construído um objecto que guarda todas as propriedades do XSD. Este objecto é depois carregado para um outro objecto que armazena o ficheiro XML e é efectuada uma leitura desse objecto. Se a leitura desse objecto for realizada até ao fim o método retorna resultado verdadeiro, caso contrário retorna resultado falso.

No caso de a validação ser feita sobre uma directoria de ficheiros, a validação é feita por ficheiro, sendo os ficheiros adicionados a uma lista de ficheiros que guarda o ficheiro e o resultado da validação (Inválido/Válido).

```
public static bool validateXmlByXsd(string prmXsdFile, string prmXmlFile)
{
      bool isValid = true;
      try
      {
            XmlReaderSettings settings = new XmlReaderSettings();
            settings.Schemas.Add(null, prmXsdFile);
            settings.ValidationType = ValidationType.Schema;
            XmlDocument document = new XmlDocument();
            document.Load(prmXmlFile);
            XmlReader rdr = XmlReader.Create(new
                        StringReader(document.InnerXml), settings);
            while (rdr.Read()) { }
      }
      catch { isValid = false;}
      return isValid;
}
```

Figura 37 – Método para a validação de XML com base num XSD

4.5.2 Funcionalidade *Delete*

Esta funcionalidade permite remover conteúdo de uma página Web e é utilizada quando pretendemos filtrar o conteúdo que é ou não mostrado na página Web ao utilizador, de acordo com restrições impostas.

Para demonstrar a aplicabilidade desta regra será apresentado um exemplo que remove da página Web uma secção completa por a mesma não estar correctamente preenchida.

Na *wikipage* existe uma secção (Map) onde o utilizador pode inserir um URL com um *link* para um mapa que mostra a localização do seu negócio.

Nesta secção o elemento *label* corresponde à legenda a atribuir ao *link* para o mapa e o elemento url guarda o *link* para o mapa com a localização. Imaginemos que o utilizador apenas inseriu o texto da $label$, mas esqueceu-se de introduzir a informação do url. Uma validação possível pode ser a imposição de que se o elemento *url* não estiver preenchido não faz sentido mostrar na página a secção *map*.

Para a definição desta regra através do editor de regras o utilizador tem em primeiro lugar de seleccionar qual o ficheiro XML que pretende validar, seleccionar na árvore, carregada com o XSD inferido a partir do XML, o elemento que pretende validar e por fim definir a regra. A Figura 38 mostra como se pode configurar a regra de *delete* através da janela de configuração.

Depois de seleccionado o elemento a validar (map) o utilizador selecciona a opção de *"Delete"* para abrir a janela de configuração da regra. Aqui a *dropdown* " $Element\ Content$ " é carregada com o conteúdo do elemento seleccionado anteriormente. Como o que se pretende validar é o conteúdo do elemento url é este que deve ser seleccionado. Na secção de "$Constraints$" define-se o que pretendemos validar para o elemento em causa, ou seja, para esta situação queremos verificar se o conteúdo do url é vazio, por isso seleccionamos a opção $NULL$. Depois de aplicar a regra o editor carrega para a árvore do lado direito um exemplo do XML final, resultado dessa aplicação.

Neste exemplo o elemento *url* não se encontra preenchido, pelo que depois da aplicação da regra o elemento map já não deverá aparecer no ficheiro final, tal como é mostrado na Figura 39.

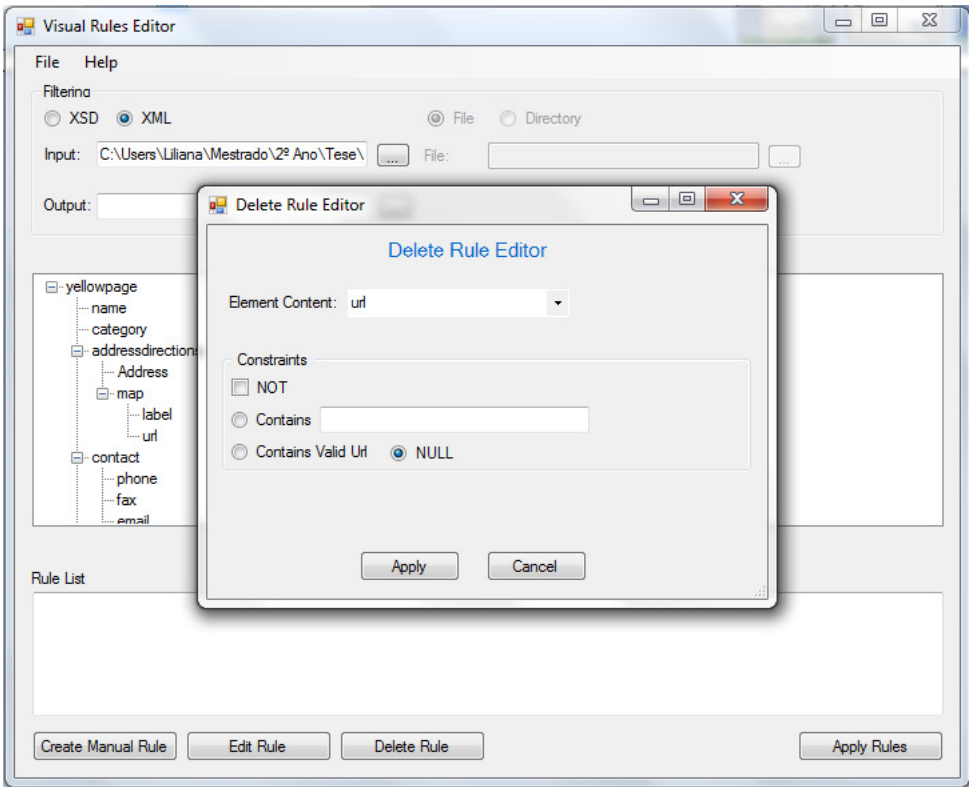

Figura 38 – Definição da regra Delete

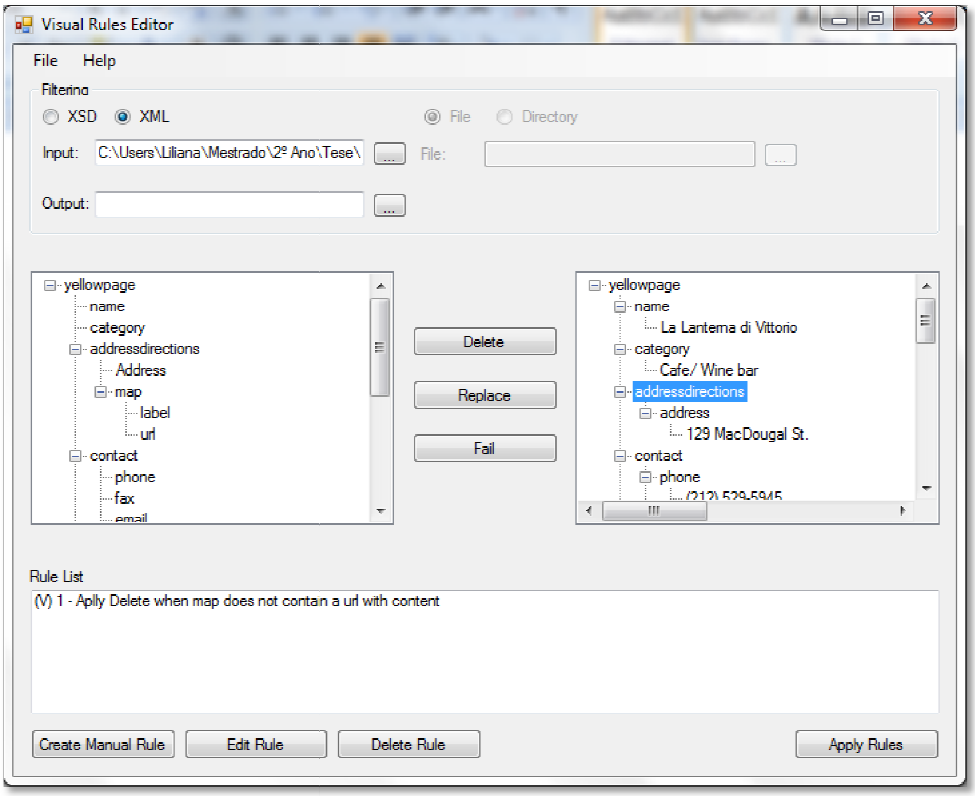

Figura 39 – Resultado da aplicação da regra Delete

4.5.2.1 Implementação

Ao nível da framework, a regra anteriormente definida visualmente pode ser implementada pelo seguinte código:

$$delete(map(Map), YP1, YP2, [deep(seq(url([\ \ ], U, empty), Map), U = \sim = empty)]).$$

Em que YP1 guarda o XML inicial e YP2 guarda o conteúdo do novo XML depois da aplicação da regra.

Do lado do C# a definição desta regra é feita através da manipulação de *strings,* o que quer dizer, que o código Prolog anterior terá de ser convertido numa *string* (Figura 40) para poder ser executado pela framework.

```
/*  prmMlParametros - objecto com as variáveis necessárias à construção
de uma regra
prmMlParametros.pSelectNode - Nó seleccionado para validação na árvore
prmMlParametros.pNodeToValidate - Elemento seleccionado na combobox de
elementos*/
_strPredicado = String.Format(@"delete({0}([],Map),YP1,YP2,
                        [deep(seq({1}([],U),empty),YP1),U=~=empty]),",
                        prmMlParametros.pSelectNode.ToLower(),
                        prmMlParametros.pNodeToValidate.ToLower());
_stbQuery.AppendFormat(@"xml2pro('{0}',YP1),",
                  prmMlParametros.pInputXmlFile.Replace("\\", "/"));
_stbQuery.AppendLine();
_stbQuery.Append(_strPredicado);
_stbQuery.AppendLine();
_stbQuery.AppendFormat(@"pro2xml(YP2,'{0}').",
                              xmlPathXmlOutput.Replace("\\", "/"));
_stbQuery.AppendLine();
_strQuery = _stbQuery.ToString();

createFile(_strFileName, _strQuery, _strPredicado,
                  prmMlParametros.pEdicao, prmMlParametros.pRuleIndex);

foreach (MlPredicados _objPredicado in _lstPredicados)
{
      if (_objPredicado.pFicheiroExecutar.Contains(_strConsult))
      {
            PlQuery.PlCall(String.Format(@"consult('{0}')",
                                    _objPredicado.pFicheiroExecutar));
            using (PlQuery query = new
                        PlQuery(_objPredicado.pNomePredicado))
            {
                  int i = 0;
                  while (query.NextSolution())
                  {
                        i++;
                        sucesso = true;
                   }
             }
        }
}
```

Figura 40 – Implementação do código em C#

A variável _strPredicado armazena o código da regra definido anteriormente. O objecto de strings _stbQuery é preenchido com o código completo para a execução da regra, o que significa que, para além da regra, tem ainda os predicados para a conversão do documento XML num termo ("xml2pro('{0}',YP1)") e vice-versa ("pro2xml(YP2,'{0}')"), em que $YP1$ e $YP2$ guardam os documentos XML de input e de output, respectivamente.

Depois da string construída, é criado o ficheiro temporário com o código criado e, utilizando os métodos disponíveis na classe Swi-cs-pl, é executado o código deste ficheiro. A instrução para que o código seja executado é a seguinte, em que, $\_objPredicado.pFicheiroExecutar$ é o caminho para o ficheiro que vai ser executado.

$$PlQuery.PlCall(String.Format(@"consult('\{0\}')", \_objPredicado.pFicheiroExecutar));$$

4.5.3 Funcionalidade Replace

A funcionalidade *Replace* permite a substituição do conteúdo existente na página por um novo conteúdo definido pelo utilizador, caso se verifique que determinada restrição acontece nessa página.

Com base na mesma *wikipage* pretendemos validar se os preços praticados pelo negócio, que está a ser inserido, foram ou não apresentados. No caso de estarem em falta aparecerá no ficheiro de *output* uma mensagem indicativa dessa situação ("$Prices\ Unavailable$") no conteúdo do elemento $prices$. A definição desta regra através do editor é feita através da selecção do elemento *prices* (é o que pretendemos validar) e da operação *Replace* (queremos substituir o conteúdo de um elemento por outro). No ecrã de edição da regra *Replace* (Figura 41), o elemento *prices*, por ser um nó folha, surge já seleccionado na *dropdown,* como tal apenas temos de clicar na restrição NULL para verificar se o elemento se encontra ou não preenchido. Se não estiver preenchido pretendemos que o conteúdo vazio seja substituído pela mensagem que definimos em "$New\ Content$".

No exemplo que estamos a usar, os preços deste negócio não foram inseridos pelo utilizador, logo o seu conteúdo será substituído pela mensagem definida anteriormente no ecrã de definição da regra (Figura 42). A regra é definida pelo termo seguinte:

$$replace(prices[\ \ ],P),prices([\ \ ],'PricesUnavailable'),YP1,YP2,$$
$$[deep(seq(prices([\ \ ],P),empty),(P = \sim = empty)]).$$

A variável P guarda o conteúdo do elemento $prices$ que vai ser validado ($P = \sim = empty$). Se no documento XML existir mais do que um elemento $prices$, a regra será

aplicada a todos porque neste caso estamos a estender a pesquisa ou transformação a todo o documento (variável).

Neste momento o ficheiro XML resultado, carregado na árvore do lado direito, contém já o resultado da aplicação das duas regras definidas, pelo que se compararmos o ficheiro inicial com o que temos após a definição das regras notam-se já algumas diferenças (Figura 43).

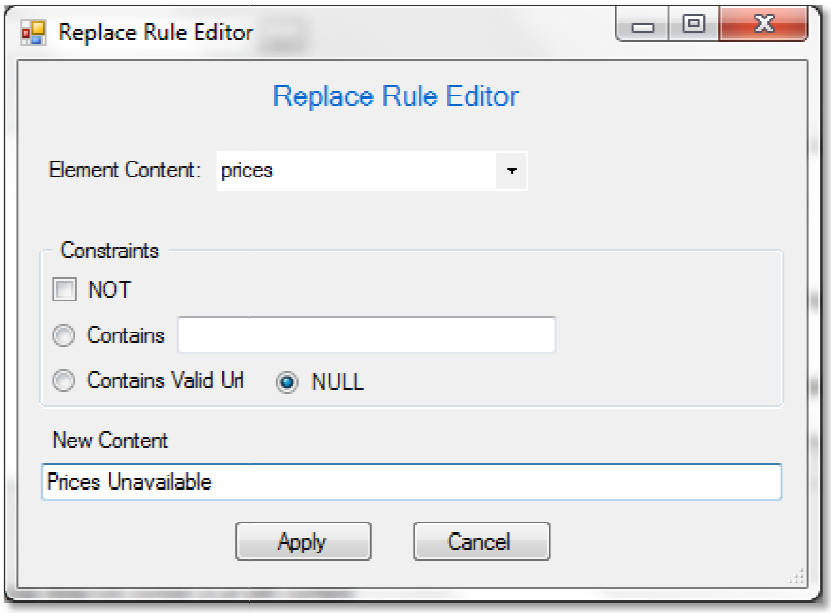

Figura 41 – Definição da regra Replace

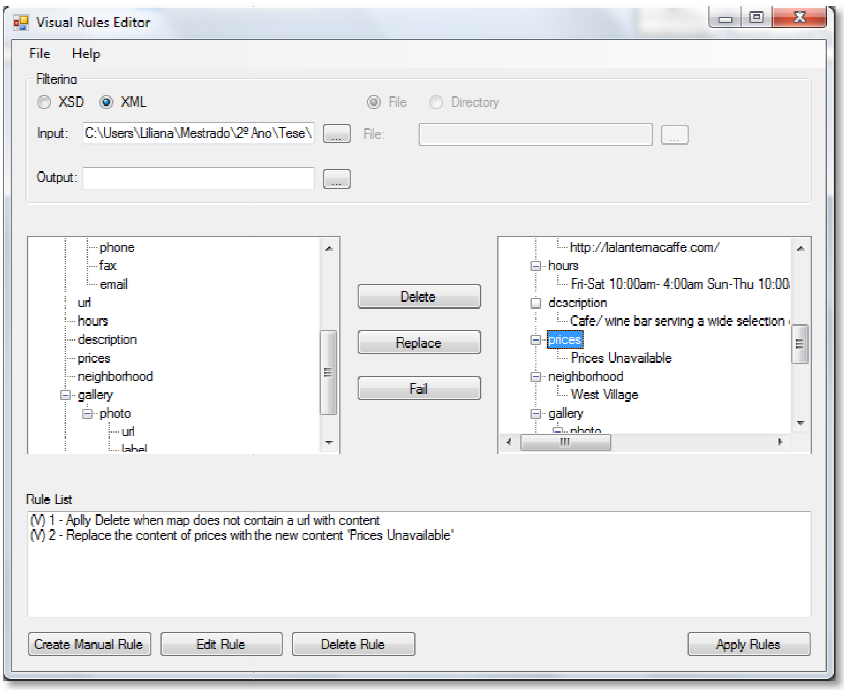

Figura 42 – Resultado da aplicação da regra Replace

```
< yellowpage >
        < name > La Lanterna di Vittorio </name >
        < category > Cafe/ Wine bar </category >
        < addressdirections >
                < Address > 129 MacDougal St.</Address >
        </addressdirections >
        < contact >
                < phone > (212) 529 − 5945 </phone >
                < fax > (212) 981 − 2731 </fax >
                < email > La.Lanterna verizon.net </email >
        </contact >
        < url > http://lalanternacaffe.com/ </url >
        < hours > Fri − Sat 10: 00am − 4: 00am Sun − Thu 10: 00am − 3: 00am </hours >
        < description > Cafe/ wine bar serving a wide selection of wines,… </description >
        < prices > Prices Unavailable </prices >
        < neighborhood > West Village </neighborhood >
        < gallery >
                < photo >
                        < url > http://www.wikipages.com/images/3/38/Lalatterna.jpg < /url >
                        < label > La Lanterna di Vittorio </label >
                </photo >
        </gallery >
        < reviews >
                http://nyc.metblogs.com/archives/2007/05/wine_in_the_gar.phtml
        </reviews >
</yellowpage >
```

Figura 43 – Ficheiro XML após aplicação das regras *Delete* e *Replace*

4.5.3.1 Implementação

A implementação da funcionalidade *Replace* é semelhante á implementação da funcionalidade anterior (Figura 40), sendo feita a manipulação de strings para a construção do predicado a enviar para o Prolog. A única diferença é o carregamento da variável _strPredicado, que neste caso é carregada com o código definido para a regra *Replace*.

4.5.4 Funcionalidade *Fail*

Um erro encontrado numa página pode ser considerado, de tal maneira grave, que não faz sentido continuar o processamento da página mostrando uma mensagem de erro indicativa dessa situação. Se a regra definida for encontrada no ficheiro XML que estamos a processar, caberá ao utilizador do editor decidir o que fazer com a página.

Mostremos um exemplo, utilizando a mesma wikipage, de como pode ser definida uma regra deste tipo. Considera-se que não faz sentido mostrar a página se a mesma não contiver emails válidos, que aqui vamos validar simplesmente pela presença ou não do carácter "@".

Visualmente esta regra define-se clicando na opção "NOT" e "$Contains$", isto porque queremos validar se o conteúdo do elemento $email$ não contém o carácter "@", além disto temos ainda de inserir qual a mensagem que aparecerá ("$Valid\ email\ not\ found!$") se a regra se verificar (Figura 44).

Esta regra é traduzida através da seguinte linha de código:

$$failure(YP1, [deep(seq(email([], E), empty), YP1), \\ not(sub\_string(E, \_, \_, \_, '@'))], 'Valid\ email\ not\ found!').$$

Tal como no exemplo anterior a pesquisa é feita ao nível de todo o documento, assim que for encontrado um elemento $email$ que satisfaça a restrição definida, é mostrada no ecrã a mensagem definida na configuração.

Neste exemplo existe um endereço de $email$ que não está correcto, pois não contém o carácter "@". Isto significa que o processamento do XML será parado e o utilizador será informado dessa situação (Figura 45).

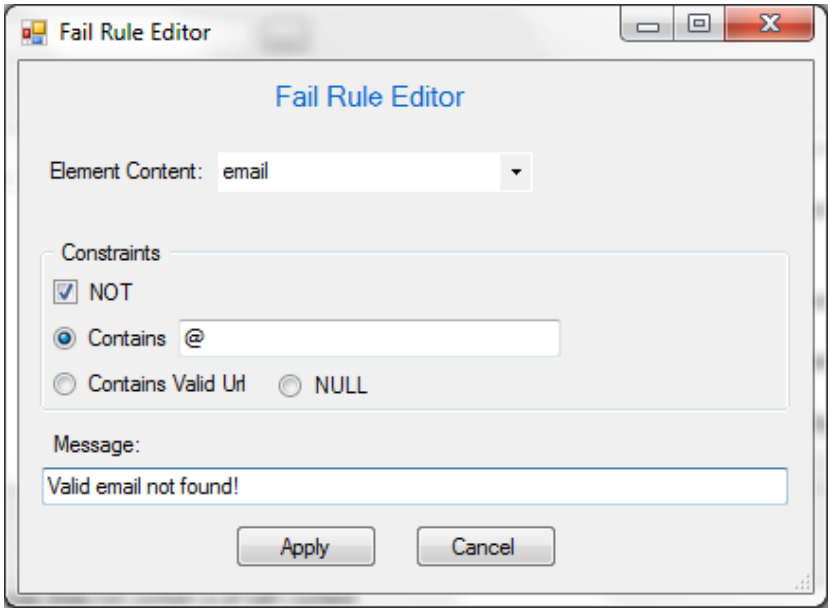

Figura 44 – Configuração da regra Fail

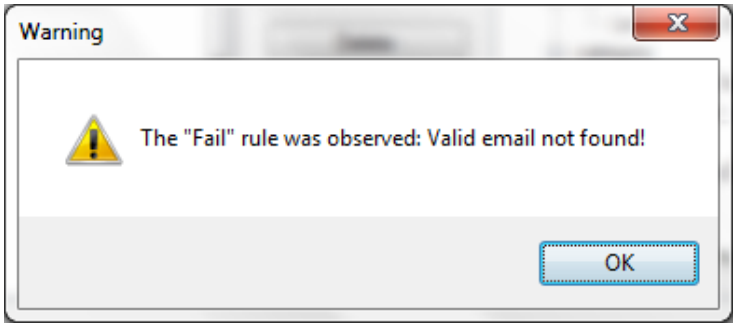

Figura 45 – Resultado da operação Fail

4.5.4.1 Implementação

Esta funcionalidade é implementada de forma análoga às anteriores.

Sempre que é configurada uma regra, a mesma é gravada em disco num ficheiro pl temporário, existindo por isso um ficheiro para cada uma das regras que vão sendo implementadas, permitindo a sua posterior reutilização. Quando a aplicação é fechada todos os ficheiros temporários criados, até então, são eliminados, à excepção do ficheiro que é criado com a aplicação de todas as regras. Este ficheiro é gerado no final da configuração das regras, quando o utilizador clica no botão $Apply\ Rules$.

4.5.5 Regras Manuais

Através da definição de regras visuais não consiguimos validar todos os aspectos que pretendemos. Para colmatar esta deficiência o editor de regras permite a definição de regras manuais, usando deste modo todas as potencialidades do Prolog e do XCentric, para as quais se assume que o utilizador esteja familiarizado com a programação em lógica.

Seleccionando a operação "$Create\ Manual\ Rule$" é apresentada ao utilizador uma janela onde poderá inserir o código que traduz a regra que quer definir. Como exemplo ilustrativo vamos considerar que pretendemos validar se todos os números de telefone (*phone*) possuem um comprimento máximo de 9 algarismos. Caso não tenham esse comprimento não serão apresentados na página.

O código que traduz esta regra é definido da seguinte forma:

$$delete(phone([], P), YP1, YP2, [deep(seq(phone([], P), empty), YP1),$$
$$(name(P, L), length(L, LP), LP =\= 9)])$$

No ecrã de edição da regra manual o utilizador terá ainda de definir quais as variáveis que vão guardar os ficheiros XML de input e output, para que o programa possa identificar as variáveis que o utilizador definiu. Para tal, o utilizador tem de preencher as caixas de texto $XML\ File\ Input\ Variable$ e $XML\ File\ Output\ Variable$, como é mostrado na Figura 46.

Depois de aplicar a regra, o XML é actualizado com o resultado desta execução. Como no exemplo que estamos a utilizar o número de telefone possui mais de 9 algarismos, este será removido da página como é provado na Figura 47.

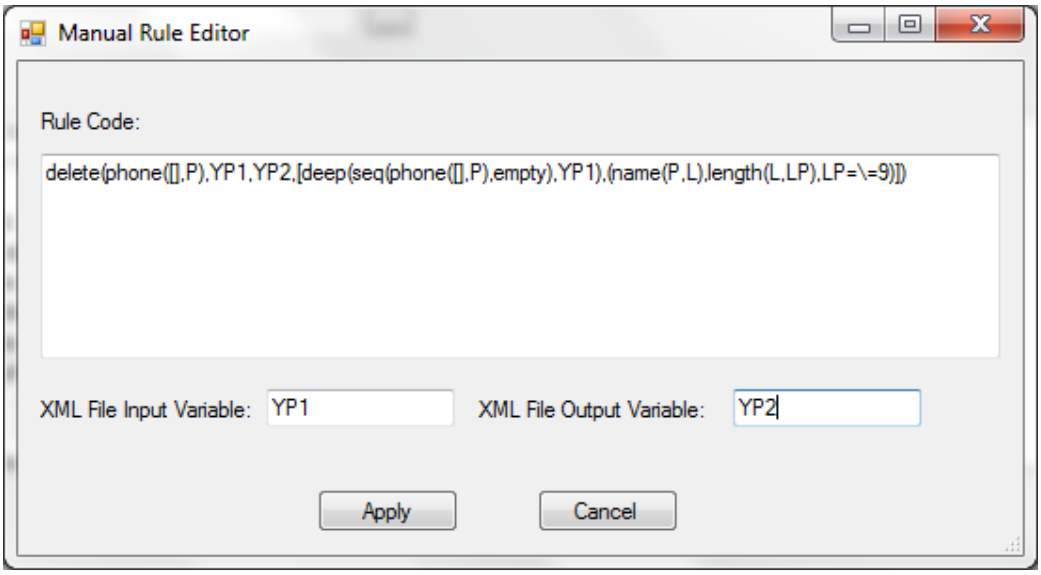

Figura 46 – Ecrã de edição de regras manuais

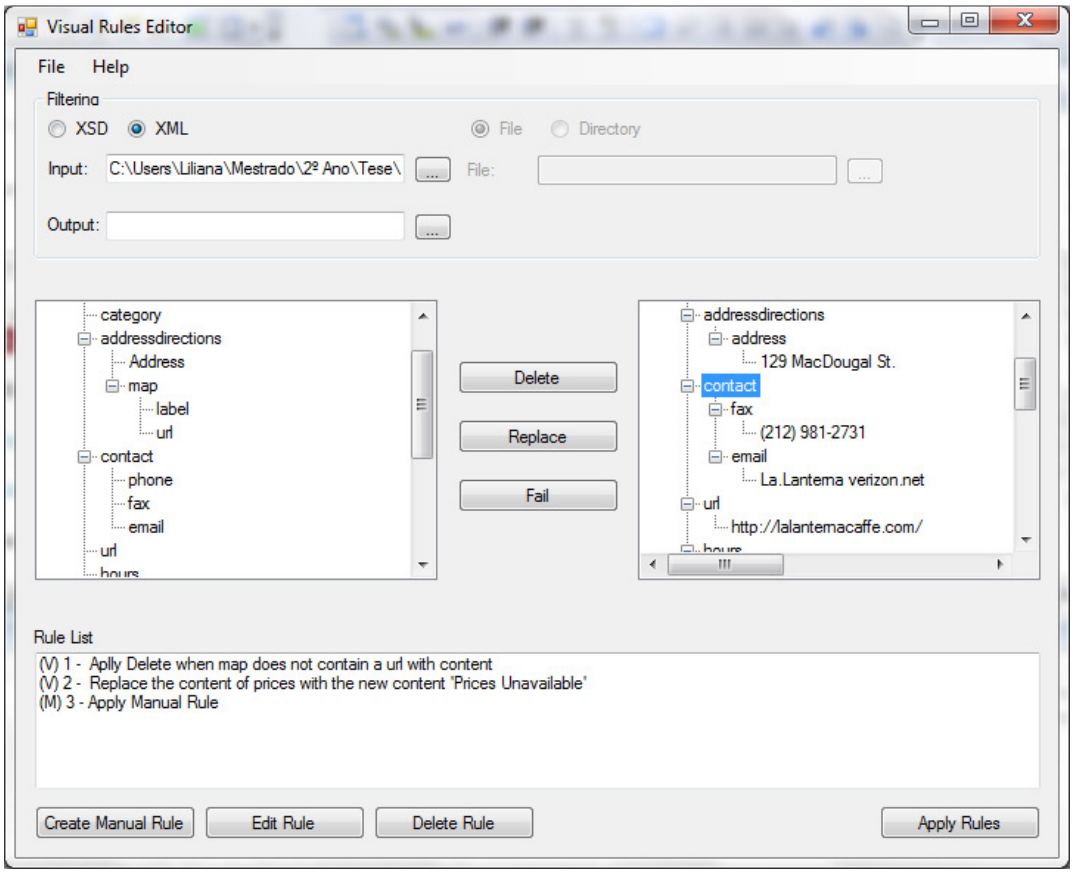

Figura 47 – Resultado da aplicação da regra manual

4.5.5.1 Implementação

A codificação desta funcionalidade é idêntica à das funcionalidades anteriores, sendo apenas alterada a instrução da regra, que é igual a que o utilizador definiu na janela de configuração de regras manuais. A identificação, no ecrã, das variáveis que guardam os ficheiros XML de input e output permite que o programa as substitua pelas suas próprias variáveis internas. Para a construção do código referente a todas as transformações sobre os documentos XML, o programa utiliza variáveis próprias para guardar esses documentos, logo para poder incluir regras que são manualmente inseridas pelo utilizador, o programa tem de saber identificar as variáveis definidas pelo utilizador para que as possa substituir pelas suas próprias variáveis.

4.5.6 Lista de Regras

À medida que as regras são executadas vão sendo adicionadas à lista de regras (Figura 48), dando ao utilizador uma visão geral de todas as regras que foram aplicadas ao ficheiro XML de input.

Cada regra da lista encontra-se marcada com um (V) no caso de serem regras configuradas visualmente ou com um (M) no caso de a regra ter sido definida manualmente. É a partir desta lista que o utilizador pode eliminar uma regra que já não pretende aplicar ou editar uma regra que já foi definida anteriormente. Em qualquer um dos casos, o XML, carregado na árvore do lado direito do ecrã, é actualizado com a nova versão correspondente à operação efectuada, ou seja, se foi removida uma regra da lista de regras o XML resultado será actualizado não reflectindo o resultado da regra que foi eliminada.

Cada uma das funcionalidades executadas a partir da selecção de uma regra na lista de regras será descrita e exemplificada nas secções seguintes.

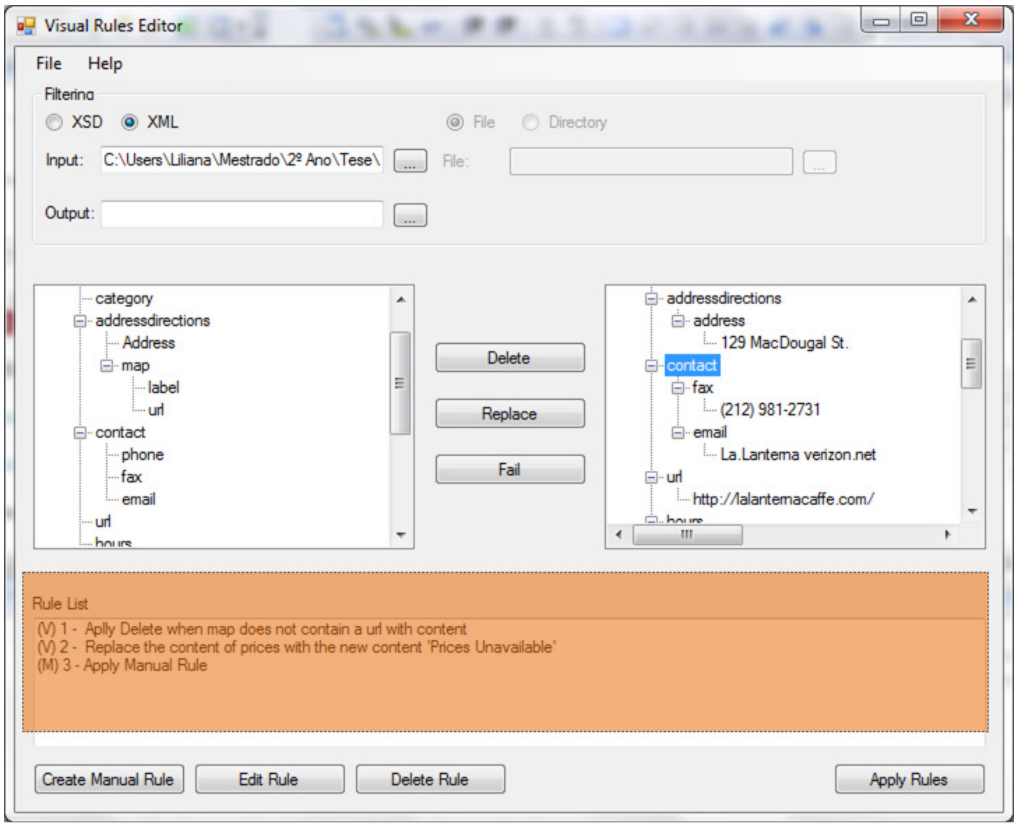

Figura 48 – Lista de Regras

4.5.7 Eliminação de Regras

Sempre que o utilizador desejar e achar necessário pode eliminar uma regra que já tenha sido eliminada, bastando para tal seleccionar da lista de regras aquela que pretende e clicar no botão "*Delete Rule*". Após a eliminação da regra o XML é actualizado com uma versão que é o resultado das regras que ainda permanecem, ou seja, o resultado da execução da regra que foi eliminada já não está reflectido no XML.

Se neste exemplo eliminarmos a última regra adicionada (regra manual) vamos verificar depois da execução da operação que o XML final voltou a apresentar no seu conteúdo o elemento `phone` (Figura 49).

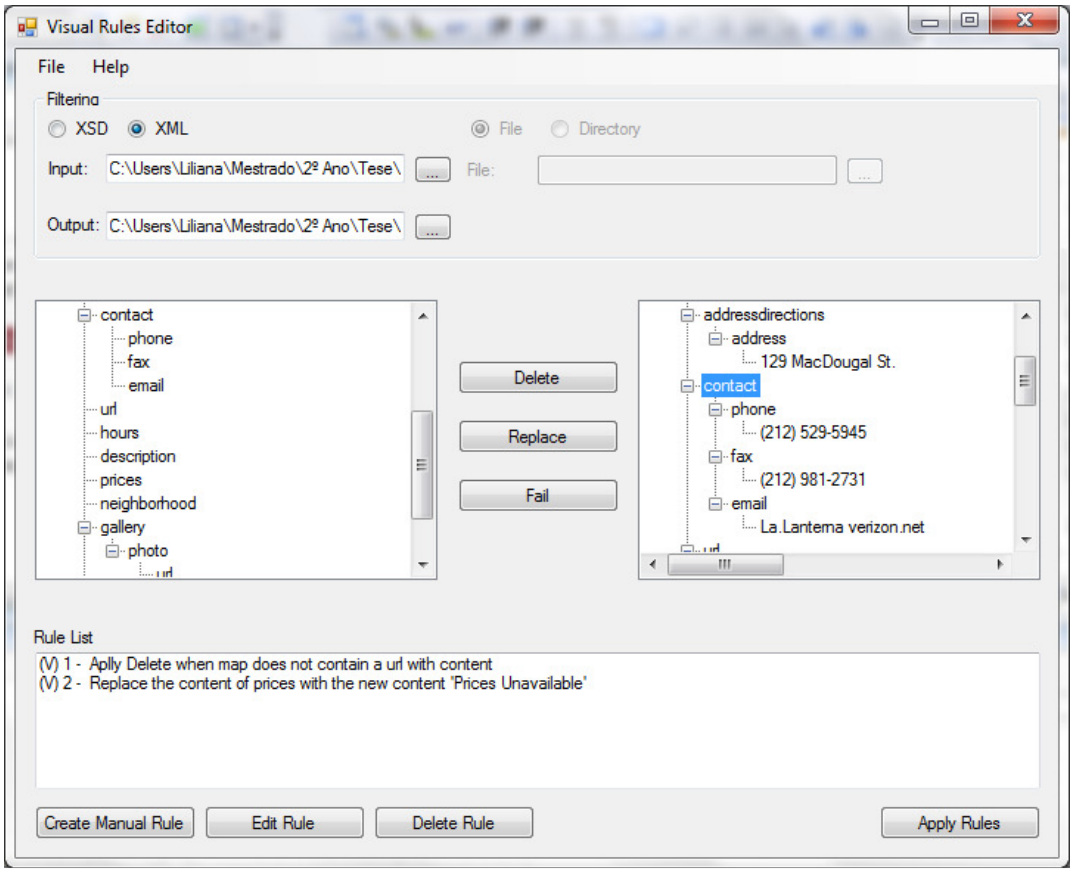

Figura 49 – Resultado da remoção de uma regra

4.5.7.1 Implementação

Á medida que as regras de validação e transformação de conteúdo vão sendo implementadas, vai sendo carregado um objecto em memória que armazena todas as regras configuradas até então. A eliminação de uma regra consiste na remoção dessa regra do objecto. A identificação dessa regra é feita internamente através de uma numeração automática correspondente ao índice que ocupa na lista de regras. Depois da eliminação da regra, todas as outras regras, existentes nesse objecto e que não foram eliminadas, voltam a ser aplicadas ao ficheiro de input, resultando num novo ficheiro sem a aplicação da regra eliminada. O ficheiro .pl que continha a execução da regra que se pretende eliminar é também removido do disco.

4.5.8 Edição de Regras

A edição de regras consiste na alteração das definições de regras que já foram aplicadas.

O utilizador pode a qualquer momento determinar que uma regra já definida não faz sentido estar configurada daquela forma, e como tal pode alterá-la. Para isso, deve seleccionar da lista de regras aquela que pretende modificar e clicar no botão

"*Edit Rule*". O ecrã que é apresentado diz respeito à regra que foi seleccionada, ou seja, se tiver sido uma regra de *Replace,* o ecrã será o de edição de regras de *Replace*, caso tenha sido uma regra manual, o ecrã será o de edição de regras manuais carregados com as configurações da regra (Figura 50).

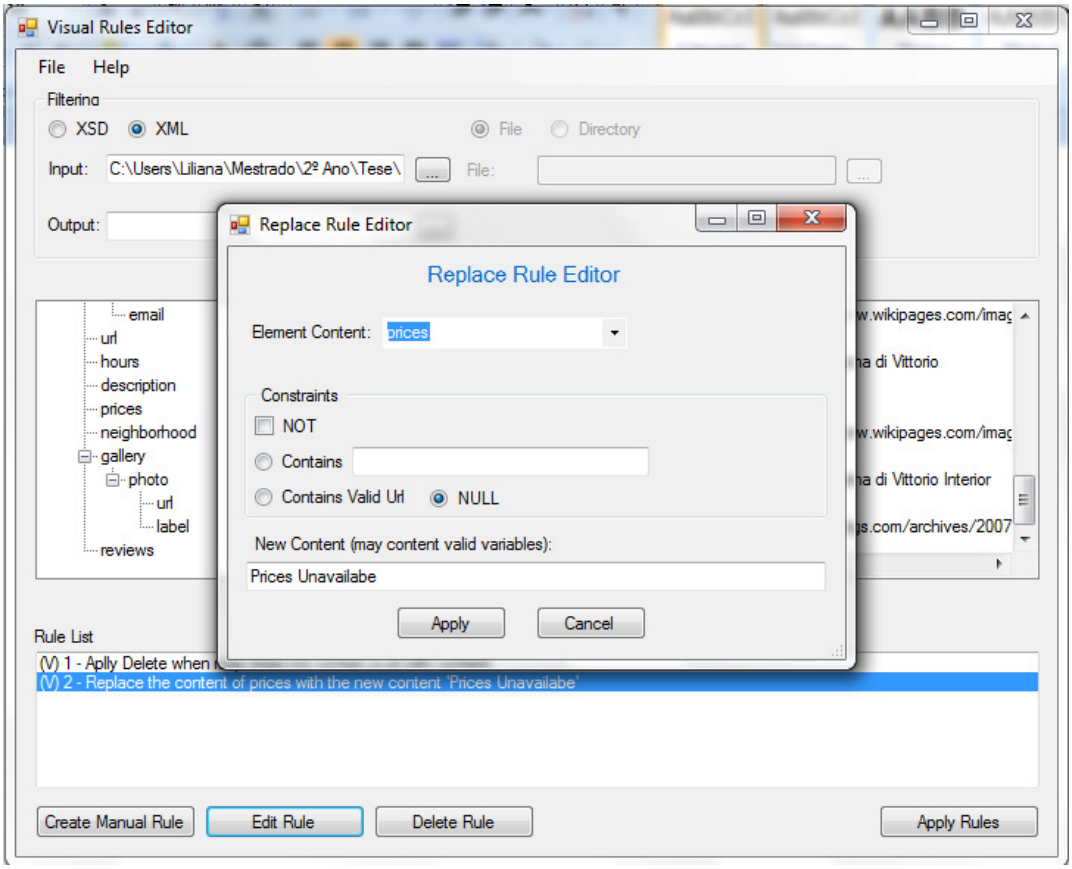

Figura 50 – Edição de uma regra de Replace

Agora o utilizador pode redefinir a regra como pretende e aplica-la novamente. O resultado final será a aplicação da nova versão da regra, se a mesma se verificar. A regra que tinha sido definida antes da alteração deixa de estar reflectida no XML.

Como exemplo, vamos alterar a regra que substitui o conteúdo vazio do elemento *prices* por um novo conteúdo. Para demonstrar a aplicação da edição de regras vamos apenas alterar o texto do novo conteúdo, inserido anteriormente, por outro "*Without Information*". Com base na Figura 51 podemos verificar que o conteúdo de *prices* foi agora substituído pelo novo conteúdo.

4.5.8.1 Implementação

A implementação desta funcionalidade consiste no carregamento da janela de configuração da regra respectiva, ou seja, com base no objecto de regras carregado em memória, e no índice que essa regra ocupa, é carregada a janela de configuração

respectiva ($Replace, Fail, Delete$ ou $Manual$) com os valores de configuração iniciais. O objecto que guarda em memória todas as regras, contém também todas as configurações iniciais dessa regra, ou seja, quando se pretende editar uma regra, é com base nesta informação que a janela é carregada novamente. Depois de realizadas todas as alterações pretendidas, a regra é substituída no objecto de regras e volta a ser aplicada ao ficheiro de input.

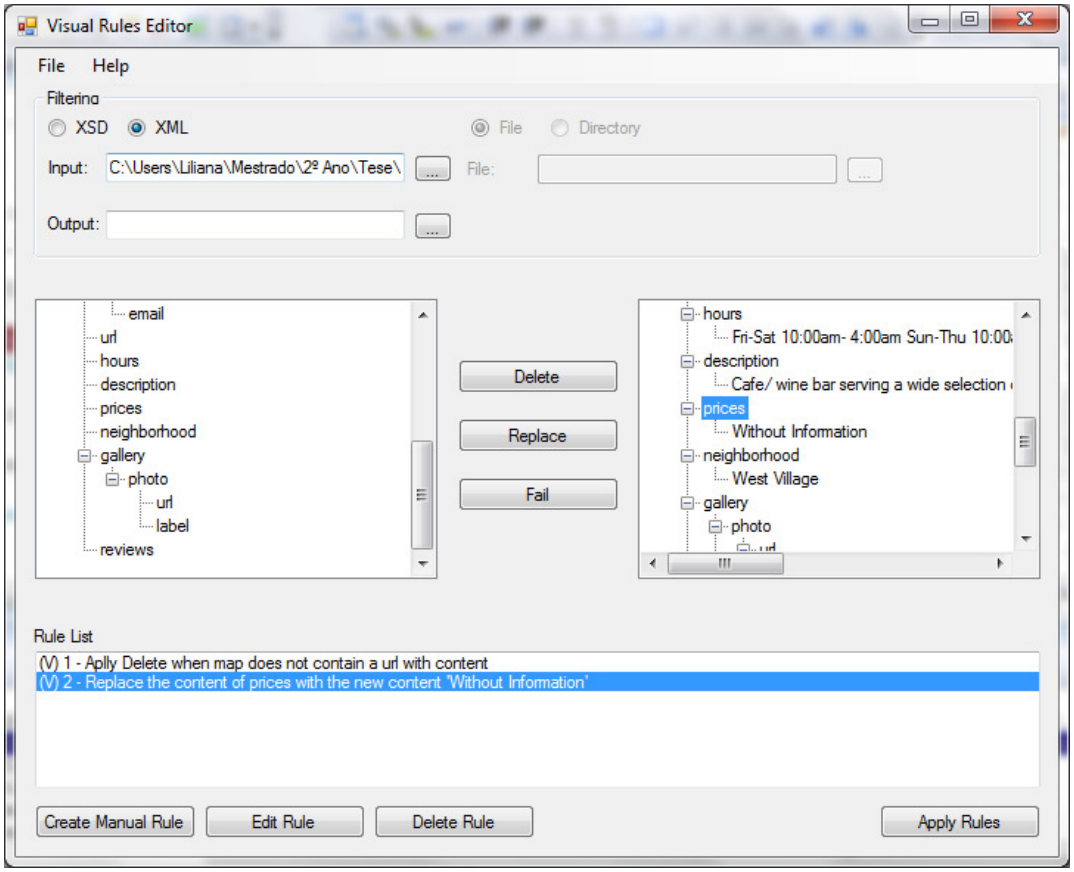

Figura 51 – Resultado da execução de uma regra editada

4.5.9 Aplicação de regras a uma directoria

O editor de regras permite aos utilizadores a aplicação de regras a uma directoria de ficheiros XML, ou seja, todas as regras serão aplicadas aos ficheiros que estiverem nessa directoria e que respeitem a estrutura do ficheiro XSD seleccionado previamente. Para tal, o utilizador terá, depois de seleccionar o ficheiro XSD, de especificar qual o caminho da directoria. Com esta selecção o ficheiro XSD é directamente carregado para a árvore do lado esquerdo do ecrã. Depois de especificado o caminho para a directoria, todos os ficheiros que existem nela são validados com base no XSD escolhido, sendo apresentada uma lista dos que

respeitam e dos que não respeitam (não serão considerados na aplicação final das regras) a sua estrutura.

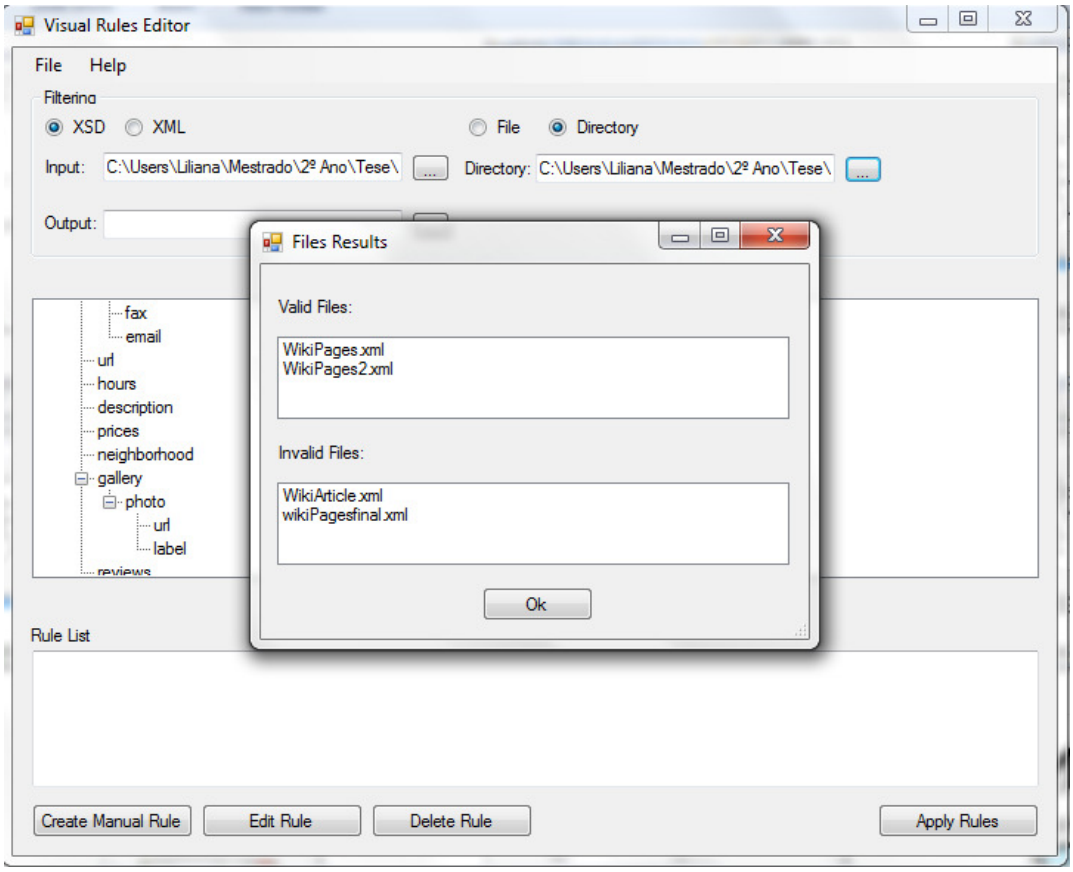

Figura 52 – Listagem de ficheiros XML com base no XSD seleccionado

Depois da selecção do ficheiro XSD e da directoria a ser validada, o utilizador pode proceder à configuração das regras que entender. Á medida que as regras vão sendo processadas, os resultados são mostrados na árvore do lado direito do ecrã usando um dos ficheiros da directoria como exemplo.

No final, a execução das regras será aplicada a todos os ficheiros da directoria, que forem instâncias do XSD e será gerado um ficheiro resultado por cada um dos ficheiros XML válidos.

A seguir é apresentado um exemplo de como são aplicadas as regras a uma directoria. Pela Figura 52 existem apenas dois ficheiros que são instâncias do XSD. Este exemplo irá aplicar apenas duas regras, uma que só é verificada no ficheiro "$WikiPages.xml$" e outra que apenas se aplica ao ficheiro "$WikiPages2.xml$".

Considerando que o último ficheiro é representado pelo XML da Figura 53 e o primeiro ficheiro é representado pelo XML já apresentado nos capítulos anteriores podem ser definidas as seguintes regras:

- Eliminação da secção map se o elemento url não se encontrar preenchido (aplicado ao primeiro ficheiro)

$$delete(map(Map), YP1, YP2, [deep(seq(url([], U), empty), Map), (U =*= empty)]).$$

- Substituição do conteúdo do elemento *category* por um novo conteúdo ("$Cafe/\ Wine\ bar$") se aquele não contiver essa string (aplicado ao segundo ficheiro)

$$replace(category([], X), category([], 'Cafe/\ Wine\ bar'), YP1, YP2,$$
$$[deep(seq(category([], X), empty), YP1),$$
$$not(sub\_string(X, \_, \_, \_, 'Cafe/\ \text{Wine bar}'))]),$$

```
< yellowpage >
    < name > Caffe Reggio </name >
    < category > Category </category >
    < addressdirections >
        < Address > 119 MacDougal Street New York, NY 10012 </Address >
        < map >
        < label > Map it </label >
        < url > http://www.onnyturf.com/subway/?address
                = 119 + MacDougal + Street, +NYC, +NY </url >
        </map >
    </addressdirections >
    < contact >
        < phone > 212 − 475 − 9557 </phone >
        < fax ></fax >< email ></email >
    </contact >
    < url > http://www.cafereggio.com/index.html </url >
    < hours > Open late </hours >
    < description > Historic restaurant and cafe in the ... </description >
    < prices > Moderate </prices >
    < neighborhood > West Village </neighborhood >
    < gallery >
    < photo >
        < url > http://www.wikipages.com/images/a/a2/Cafereggio.jpg </url >
        < label > Caffe Reggio </label >
    </photo >
</gallery >
< reviews ></reviews >
</yellowpage >
```

Figura 53 – Exemplo do ficheiro WikiPages2.xml

Se consultarmos a árvore do lado direito do ecrã, verificamos que apenas uma das regras foi aplicada, isto porque só um dos ficheiros está a ser utilizado como exemplo. Quando o utilizador clicar no botão "*Apply Rules*" as regras serão aplicadas a todos os ficheiros da directoria. Comparando o segundo ficheiro com o resultado da aplicação das regras (Figura 54) verificamos que o conteúdo da categoria (elemento *category*) foi substituído pelo novo conteúdo.

```
< yellowpage >
        < name > Caffe Reggio </name >
        < category > Cafe/Wine Bar </category >
        < addressdirections >
                < Address > 119 MacDougal Street New York, NY 10012 </Address >
                < map >
                < label > Map it </label >
                < url > http://www.onnyturf.com/subway/?address
                                = 119 + MacDougal + Street, +NYC, +NY </url >
                </map >
        </addressdirections >
        < contact >
                < phone > 212 − 475 − 9557 </phone >
                < fax ></fax >< email ></email >
        </contact >
        < url > http://www.cafereggio.com/index.html </url >
        < hours > Open late </hours >
        < description > Historic restaurant and cafe in the ... </description >
        < prices > Moderate </prices >
        < neighborhood > West Village </neighborhood >
        < gallery >
        < photo >
                < url > http://www.wikipages.com/images/a/a2/Cafereggio.jpg </url >
                < label > Caffe Reggio </label >
        </photo >
</gallery >
< reviews ></reviews >
</yellowpage >
```

Figura 54 – Ficheiro resultado referente ao ficheiro de input WikiPages2.xml

4.5.9.1 Implementação

A implementação desta regra é feita através dos métodos implementados para as regras já descritas. No entanto, só vão sendo aplicadas ao ficheiro de exemplo que é mostrado no ecrã. Só na aplicação final das regras é que o código de execução abrange todos os ficheiros existentes na directoria. A aplicação final das regras consiste na execução de um ficheiro .pl que contém o código de todas as regras que foram sendo configuradas. É nesta altura que os ficheiros XML válidos da directoria são percorridos para que se possa efectuar a transformação de cada um com base nas restrições definidas.

5. Conclusão

O rápido crescimento das aplicações Web e o aumento da sua complexidade, contribuiu em muito para o aparecimento de técnicas de modelação que fossem capazes de validar e verificar a sintáctica e semântica dessas aplicações. Depois da análise de tudo o que já foi implementado e do que ainda falta fazer nesta área, são de notar algumas lacunas ao nível de validações do conteúdo de aplicações Web.

A linguagem XCentric, baseada em programação em lógica com restrições, que permite a unificação de termos com símbolos de função de aridade flexível, provou ser a indicada para o processamento de documentos XML, como se pode ver pelos exemplos apresentados ao longo deste documento. A framework que utiliza esta linguagem, VeriFLog, apresenta-se como sendo uma ferramenta fundamental para a verificação de conteúdos Web, podendo ser utilizada como uma linguagem intermédia para outras linguagens de programação com sintaxe semelhante à do XML.

A principal contribuição deste trabalho foi a criação de uma aplicação gráfica que disponibiliza, de forma mais amigável para o utilizador, todas as funcionalidades que o VeriFLog já suportava e acrescenta outras que tornam a utilização dessa framework mais robusta e completa.

Hoje em dia as aplicações Web colaborativas são cada vez mais comuns. Qualquer pessoa pode contribuir com o seu conhecimento para a sua construção. A aplicação desenvolvida surge como uma ferramenta ideal para validar o conteúdo deste tipo de aplicações, já que, como as páginas são construídas por diversas pessoas é bastante provável que o conteúdo da página tenha erros ou desobedeça a critérios de qualidade. Desta forma, de um modo bastante simples, podem ser definidas regras com restrições que irão validar se o conteúdo da página produzindo transformações no mesmo.

Como trabalho futuro pensa-se na integração desta aplicação com a rede, ao nível da *firewall*, de modo a ser possível filtrar o conteúdo que uma pessoa pode ou não visualizar numa determinada página. A ideia não é o bloqueio de endereços, mas sim remover, substituir ou bloquear acessos a alguns critérios que não obedecem às restrições impostas. Acreditamos ainda que esta ferramenta poderá ser estendida de modo a ser utilizada como um *plug-in* do browser para apresentação do conteúdo de acordo com determinadas restrições no lado do cliente ou para verificar a qualidade do site em termos de design e facilidade de leitura, por exemplo, a Google impõe um conjunto de boas práticas [2] (por exemplo: garantir que todas as páginas estão

acessíveis a partir de pelo menos um *link* estático, manter o número máximo de *links* numa determinada página; detectar a existência de *links* errados e corrigir o HTML correspondente) para o desenho de páginas, que podiam ser verificadas por uma adaptação desta ferramenta.

6. Referências Bibliográficas

[Alexandre L. e Coelho J., 2011a] - Liliana Alexandre and Jorge Coelho, XCentric-based Visual Approach to Web Content Verification, 9ª XML: Aplicações e Tecnologias Associadas, Escola Superior de Estudos Industriais e de Gestão, Vila do Conde, 1 e 2 de Julho 2011; 71-82

[Alexandre L. e Coelho J., 2011b] - Liliana Alexandre and Jorge Coelho, Filtering XML Content for Publication and Presentation on the Web. Sixth IEEE International Conference on Digital Information Management, The University of Melbourne, Australia. IEEE 2011

[Alfaro L., 2001a] - Alfaro L. Model checking the world wide Web. Proceedings of the Computer Aided Verification, 13th International Conference, London, U.K. (Lecture Notes in Computer Science, vol. 2102), Berry G, Comon H, Finkel A (eds.). Springer: Berlin, 18–22 July 2001; 337–349.

[Alfaro L., 2001b] - Alfaro L, Henzinger TA, Mang FY. MCWEB: A model-checking tool for website debugging. Proceedings of the WWW Posters, Hong Kong, 2001; 86–87.

[Alpuente M. et al., 2005] - Alpuente M, Ballis D, Falaschi M. A rewriting-based framework for Web sites verification. Electronic Notes in Theoretical Computer Science 2005; 124(1):41–61.

[Alpuente M. et al., 2006a] - Alpuente M, Ballis D, Falaschi M, Romero D. A semi-automatic methodology for repairing faulty websites. Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM), Pune, India. IEEE Computer Society: Washington, DC, U.S.A., 2006; 31–40.

[Alpuente M. et al., 2006b] - Alpuente M, Ballis D, Falaschi M. Rule-based verification of Web sites. International Journal on Software Tools for Technology Transfer 2006; 8(6):565–585.

[Alpuente M. et al., 2007] - Alpuente M, Ballis D, Falaschi M, Ojeda P, Romero D. A fast algebraic Web verification service. Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR), Innsbruck, Austria, 2007; 239–248.

[Bellettini C. et al., 2004] - Bellettini C, Marchetto A, Trentini A. WebUML: Reverse engineering of Web applications. Proceedings of the 2004 ACM Symposium on Applied Computing SAC, Nicosia, Cyprus, 2004; 1662–1669.

[Bordbar B. e Anastasakis K., 2005] - Bordbar B, Anastasakis K. MDA and analysis of Web applications. Proceedings of the Trends in Enterprise Application Architecture, Trondheim, Norway (Lecture Notes in Computer Science, vol. 3888). Springer: Berlin, 2005; 44–55.

[Benzaken V. et al., 2003] - Benzaken V., Castagna G. e Frisch A. CDuce: an XML-centric generalpurpose language. In Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming, pages 51–63, Uppsala, Sweden, 2003. ACM Press.

[Bry F. e Schaffert S., 2002] - Bry F. e Schaffert S.. The XML Query Language Xcerpt: Design principles, Examples, and Semantics. In 2nd Annual International Workshop Web and Databases, volume 2593 of LNCS.Springer Verlag, 2002.

[Castelluccia D. et al., 2006] - Castelluccia D, Mongiello M, Ruta M, Totaro R. WAVer: A model checking-based tool to verify Web application design. Electronic Notes in Theoretical Computer Science 2006; 157(1):61–76.

[Chen J. e Zhao X., 2004] - Chen J, Zhao X. Formal models for Web navigations with session control and browser cache. Proceedings of the 6th International Conference on Formal Engineering Methods, ICFEM, Seattle, WA, U.S.A., 2004; 46–60.

[Coelho J. e Florido M., 2004] - Coelho J. and Florido M. CLP(Flex): Constraint Logic Programming Applied to XML Processing. In Ontologies, Databases and Applications of Semantics (ODBASE), volume 3291 of LNCS. Springer Verlag, 2004.

[Coelho J. e Florido M., 2006a] - Coelho J. and Florido M.. VeriFLog: Constraint Logic Programming Applied to Verification of Website Content. In Int. Workshop XML Research and Applications (XRA'06), volume 3842 of LNCS. Springer-Verlag, 2006.

[Coelho J. e Florido M., 2006b] - Coelho J, Florido M. VeriFLog: A constraint logic programming approach to verification of Web site content. Proceedings of the International Workshops on Advanced Web and Network Technologies and Applications (APWeb), Harbin, China, 2006; 148–156.

[Coelho J. e Florido M., 2006c] - Coelho J. and Florido M. Unification with flexible arity symbols: a typed approach. In Informal proceedings of the 20th International Workshop on Unification (UNIF'06), Seattle, USA, 2006.

[Coelho J. e Florido M., 2007a] - Coelho J, Florido M. Type-based static and dynamic Web site verification. Proceedings of the Second International Conference on Internet and Web Applications and Services (ICIW), Le Morne, Mauritius, 2007; 32.

[Coelho J. e Florido M., 2007b] – Coelho J., Florido M.: XCentric: logic programming for XML processing. WIDM 2007:1-8

[Conallen J., 1999] - Conallen J. Modeling Web application architectures with UML. Communications of the ACM 1999; 42(10):63–71.

[Despeyroux T., 2004] - Thierry Despeyroux. Practical semantic analysis of Web sites and documents. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, WWW, pages 685–693. ACM, 2004.

[Di Lucca GA e Di Penta M., 2003] - Di Lucca GA, Di Penta M. Considering browser interaction in Web application testing. Proceedings of the 5th International Workshop on WebSite Evolution (WSE), Amsterdam, The Netherlands. IEEE Computer Society Press: Silver Spring, MD, 2003; 74.

[Di Sciascio E. et al., 2003] - Di Sciascio E, Donini FM, Mongiello M, Totaro R, Castelluccia D. Design verification of Web applications using symbolic model checking. Proceedings of the 5th International Conference on Web Engineering, ICWE, Sydney, Australia (Lecture Notes in Computer Science, vol. 3579). Springer: Berlin, 2005; 69–74.

[France R. et al., 1998] - France R., Evans A. Lano K., B. Rumpe - The UML as a formal modeling notation. Computer Standards & Interfaces 19 1998, 325–334

[Graunke PT. et al., 2003] - Graunke PT, Findler RB, Krishnamurthi S, Felleisen M. Modeling Web interactions. Proceedings of the Programming Languages and Systems, 12th European Symposium on Programming, ESOP, Warsaw, Poland (Lecture Notes in Computer Science, vol. 2618), Degano P (ed.). Springer: Berlin, 7–11 April 2003; 238–252.

[Harmelen F. e Meer J., 1999] - Frank van Harmelen and Jos van der Meer. Webmaster: Knowledge-based verification of Web-pages. In Ibrahim F. Imam, Yves Kodratoff, Ayman El-Dessouki, and Moonis Ali, editors, IEA/AIE, volume 1611 of Lecture Notes in Computer Science, pages 256–265. Springer, 1999.

[Harel D., 1987] - Harel D. Statecharts: A visual formulation for complex systems. Science of Computer Programming 1987; 8(3):231–274.

[Henzinger M. R., 1995] - Henzinger M. R., Henzinger T. A., and Kopke P. W.. Computing simulations on finite and infinite graphs. In IEEE Symposium on Foundations of Computer Science, pages 453–462, 1995

[Hosoya H. e Benjamin P., 2000] - Hosoya H. e Benjamin P. XDuce: A typed XML processing language. In Third International Workshop on the Web and Databases (WebDB2000), volume 1997 of Lecture Notes in Computer Science, 2000.

[Kutsia T., 2002] - Kutsia T. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proceedings of Joint AICS'2002 - Calculemus'2002 conference, volume 2385 of Lecture Notes in Artificial Intelligence, pages 290–304, Marseille, France, 2002. Springer Verlag.

[Licata DR. e Krishnamurthi S., 2004] - Licata DR, Krishnamurthi S. Verifying interactive Web programs. Proceedings of the IEEE International Conference on Automated Software Engineering, Linz, Austria. IEEE Computer Society: New York, 2004; 164–173.

[Manar H. et al., 2009] - Manar H. Alal_, James R. Cordy, and Thomas R. Dean. Modelling methods for Web application verification and testing: state of the art. Softw. Test., Verif. Reliab., 19(4):265_296, 2009.

[Manar H. et al., 2007] - Manar Alalfi H., James Cordy R., Dean Thomas R. A Survey of Analysis Models and Methods in Website Verification and Testing; School of Computing, Queen's University, Kingston, Ontario, Canada, 2007

[Rob P. e Coronel C., 2004] - Rob P, Coronel C. Database Systems: Design Implementation and Management (5th edn). Course Technology: New York, January 2004.

[Ricca F e Tonella P., 2000] - Ricca F, Tonella P. Web site analysis: Structure and evolution. Proceedings of the International Conference on Software Maintenance, San Jose, CA, 2000; 76–86.

[Winckler M. e Palanque PA., 2003] - Winckler M, Palanque PA. StateWebCharts: A formal description technique dedicated to navigation modelling of Web applications. Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification, DSV-IS, Madeira Island, Portugal, 2003; 61–76.

7. Referências Sites

[1]. *Department of Computer Science.* http://www.cs.umd.edu/~nau/cmsc421/chapter08.pdf (acedido em 09 de Junho de 2011).

[2]. *Google webmaster central.* http://www.google.com/support/webmasters/bin/answer.py?answer=35769#1 (acedido em 21 de Junho de 2011).

[3]. *SWI - Prolog.* www.swi-prolog.org (acedido em 19 de Maio de 2011).

[4]. *SWI-Prolog interface to C#.* http://www.swi-prolog.org/contrib/CSharp.html (acedido em 10 de Dezembro de 2010).

[5]. *Visual C# Developer Center.* http://msdn.microsoft.com/en-us/vcsharp/aa336809 (acedido em 21 de Junho de 2011).

[6]. *W3C Extensible Markup Language.* http://www.w3.org/XML/ (acedido em 04 de Junho de 2011).

[7]. *W3C XML Schema.* http://www.w3.org/XML/Schema (acedido em 24 de Junho de 2011).

[8]. *Wikipages.* http://www.wikipages.com/ (acedido em 21 de Maio de 2011).

[9]. *Wikipedia.* http://en.wikipedia.org/wiki/Main_Page (acedido em 25 de Maio de 2011).