XATA2007

XML: Aplicações e Tecnologias Associadas

5ª Conferência Nacional

Editores:

José Carlos Ramalho
João Correia Lopes
Luís Carriço

15 e 16 de Fevereiro de 2007

Ficha Técnica:

Design Gráfico: Benedita Contente Henriques
Redacção: J. C. Ramalho, J. C. Lopes e Luís Carriço
Composição: J. C. Ramalho, J. C. Lopes e Luís Carriço
Tiragem: 100 exemplares
Edição: Fevereiro de 2007
ISBN: 978-972-99166-4-9

Prefácio:

Esta é a quinta conferência sobre XML e Tecnologias Associadas. Continuando uma política de descentralização a XATA vai, desta vez, visitar a capital.

Com a organização local da responsabilidade do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa, a XATA vai ocorrer nos dias 15 e 16 de Fevereiro nas instalações da Faculdade de Ciências.

Relativamente à submissão de trabalhos pudemos constatar o seguinte: já há um conhecimento do âmbito e contexto da XATA, o que se traduz nos objectivos dos trabalhos apresentados; o processo de revisão decorreu sem percalços, não tendo ocorrido, pela primeira vez, nenhuma falha por parte dos revisores; esta edição não foi excepção à tendência nacional nas restantes conferências e registou também uma acentuada quebra no número de trabalhos submetidos.

Queria aqui destacar o trabalho de qualidade feito pelos revisores que permitiu aos autores melhorarem substancialmente os seus trabalhos. Mesmo para aqueles que viram os seus artigos recusados, as críticas que receberam irão permitir, com certeza, orientar melhor o sentido do seu trabalho.

Na reunião da Comissão Científica onde foi feita a selecção final dos trabalhos foi delineado também o programa que ficou dividido em seis sessões que se descrevem a seguir.

Programa

Sessão I: Dialectos XML

Esta sessão é dedicada à especificação de novas linguagens XML de domínio específico.

Sessão II: Ontologias

A representação de ontologias em XML e o seu processamento tem sido um tema presente nas últimas edições da XATA. Nesta edição, marca presença de novo, mostrando a relevância do tema.

Sessão III: Edição e Processamento

Esta sessão vem, de novo, demonstrar a maturidade da comunidade portuguesa que trabalha com XML. Dedicada às tecnologias de base, vem abrir discussões sobre modelos de edição e de processamento.

Sessão IV: Bibliotecas Digitais

Esta é uma área, já reconhecida por todos, onde a aplicação do XML é fundamental. Mais uma vez, iremos discutir representações, metainformação, motores de busca e repositórios.

Sessão V: Transformação de XML e Pipelines

Esta sessão vem complementar ainda mais a sessão III mostrando e realçando a preocupação e o "saber-fazer"desta comunidade.

Sessão VI: Aplicações de XML

À semelhança de todas as outras edições da XATA, esta é a sessão mais heterogénea onde o foco é a aplicação do XML para os fins mais diversos.

Sessão Lusco-Fusco

A XATA2007 inclui uma sessão, no espírito das sessões de posters das edições anteriores, mas com uma organização diferente. Esta sessão, denominada "Lusco-Fusco", funcionará nos seguintes moldes:

- Terá a duração de 1/2 hora de apresentação e 20 minutos de discussão;
- Os trabalhos seleccionados (5 no total) repartirão entre si o tempo disponível, com um máximo de 6 minutos para cada um;
- A organização montará uma única apresentação ("slideshow"), com todas as apresentações concatenadas em sequência, que colocará a correr no início da sessão;
- Cada autor terá "a palavra"durante o decorrer dos seus slides.

Com esta sessão pretende-se dignificar os trabalhos que foram submetidos e que ainda se encontram em fase de projecto. Em edições anteriores as sessões de posters não o conseguiram fazer. Esta sessão permite dar a palavra a todos os autores para exporem a sua ideia e, como o tempo de apresentação é relativamente curto, terá um ritmo interessante, que cremos, será cativante para a audiência.

Uns dias bem "xatos" para todos ao som do fado de Lisboa ...

Fevereiro de 2007,

José Carlos Ramalho

Comissão Organizadora:

Luís Carriço — Faculdade de Ciências da Universidade de Lisboa
Rui Lopes — Faculdade de Ciências da Universidade de Lisboa
Marco Sá — Faculdade de Ciências da Universidade de Lisboa
João Correia Lopes — Universidade do Porto, INESC Porto
José Carlos Ramalho — Universidade do Minho

Comissão Científica:

José Carlos Ramalho	Universidade do Minho — Chair
Ademar Aguiar	Universidade do Porto e INESC Porto
Adérito Marcos	Universidade do Minho, CCG
Alberto Rodrigues da Silva	Instituto Superior Técnico
Alda Lopes Gançarski	Universidade do Minho
Ana Alice Baptista	Universidade do Minho
Benedita Malheiro	Instituto Superior de Engenharia do Porto
Carlos Viegas Damásio	Universidade Nova de Lisboa
Cristina Ribeiro	Universidade do Porto e INESC Porto
Francisco Couto	Universidade de Lisboa
Gabriel David	Universidade do Porto e INESC Porto
Giovani Librelotto	Centro Universitário Franciscano, Brasil
João Correia Lopes	Universidade do Porto e INESC Porto
Jorge Cardoso	Universidade da Madeira
José Borbinha	Instituto Superior Técnico
José João Almeida	Universidade do Minho
José Paulo Leal	Universidade do Porto
Luís Carriço	Universidade de Lisboa
Luís Ferreira	Instituto Politécnico do Cávado e do Ave
Luís Moura e Silva	Universidade de Coimbra
Mário J. Silva	Universidade de Lisboa
Marta Jacinto	Instituto das Tecnologias de Informação na Justiça
Miguel Ferreira	Universidade do Minho
Nuno Horta	Instituto Superior Técnico
Paulo Gomes	Escola Superior de Tecnologia e Gestão de Portalegre
Pedro Almeida	Universidade de Aveiro
Pedro Antunes	Universidade de Lisboa
Pedro Henriques	Universidade do Minho
Salvador Abreu	Universidade de Évora

Revisores Adicionais:

Alberto Simões	Universidade do Minho
Rui Lopes	Universidade de Lisboa

Agradecimentos:

Os editores deste livro de actas querem expressar o seu agradecimentos a quantos participaram na realização desta conferência.

Em primeiro lugar agradecemos a todos os autores a produção dos trabalhos que aqui ficam publicados, já que sem eles não existiria a XATA2007. Obrigado por terem aceite o desafio.

A todos os revisores, a quem foi solicitado um enorme esforço, um grande obrigado por cumprirem a árdua tarefa que lhes propusemos, nos prazos estabelecidos. A eles se deve a qualidade final desta obra e, não menos importante, o retorno crítico enviado a todos os autores, que cremos ser um contributo para a melhoria da qualidade da investigação e dos trabalhos publicados na área do XML em Portugal.

Por fim, não terminaremos sem uma palavra de apreço a todos os participantes que decidiram comparecer num evento onde se pretende privilegiar a troca e discussão de ideias. A sua presença virá com certeza engrandecer o debate, gerando novas ideias, que originarão outros tantos trabalhos e quiçá um reencontro na próxima XATA.

José Carlos Ramalho
João Correia Lopes
Luís Carriço

Patrocínios:

A Comissão Organizadora deseja expressar a sua gratidão às seguintes organizações:

- Critical Software
- Departamento de Engenharia Electrotécnica e de Computadores da Universidade do Porto
- Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa
- Departamento de Informática da Universidade do Minho
- Fundação para a Ciência e a Tecnologia
- Lasige - Large-Scale Informatics Systems Laboratory
- Microsoft Corporation Portugal

Conteúdo

XTDL, XML Tool Definition Language

Daniela Carneiro da Cruz Pedro Rangel Henriques

{danieladacruz,prh}@di.uminho.pt
Departamento de Informática, CCTC
Universidade do Minho

Resumo

A necessidade de distribuir ferramentas, leva, tipicamente, à criação de um site onde o software em causa esteja disponível e possa ser descarregado. Porém atrás disso vem a necessidade de documentar o produto: o que é; para que foi concedido; como se usa; como se instala. E não basta fazê-lo por um único meio, isto é, através do site criado para o disponibilizar; logo de seguida outras necessidades de documentação vão surgir.
Neste artigo pretendemos apresentar um dialecto XML, XTDL (XML Tool Definition Language) que criámos para descrever ferramentas. Uma descrição em XTDL é independente do tratamento que se lhe pretende dar, fornecendo detalhes sobre autoria, versões em distribuição e documentação disponível. Além disso, permite incluir uma descrição livre e informação técnica, tal como arquitectura, tecnologias utilizadas, etc. XTDL permite ainda descrever um mapa de conceitos sobre o domínio em que a ferramenta se enquadra.
Apresentaremos depois um conjunto de ferramentas, XTS (XTDL Tool Set) que desenvolvemos para processar tais descrições. XTS inclui geradores para construir um site WWW, documentação (por exemplo na forma de um folheto) e uma colecção de diapositivos (por ex., para apoiar uma demonstração ou uma comunicação oral). Caso o mapa de conceitos seja incluído o gerador do site produzirá um grafo navegável, graças ao CMG (Conceptual Map Generator, um compilador de Prolog para dotty do Graphviz desenvolvido no nosso grupo).
O XTS será apresentado num browser da Internet, onde cada utilizador poderá submeter a sua própria descrição XTDL a fim de gerar o site e a documentação que pretenda. No entanto, para que a pessoa possa submeter informação no sistema, e proceder posteriormente à sua consulta e modificação, deverá estar registada no sistema. Com este fim, será utilizado uma base de dados e a tecnologia .NET (linguagem ASP e C#).
O paper inclui ainda, como caso de estudo, a descrição de uma outra ferramenta da nossa autoria, o LISSc (LISS Compiler).

1 Introdução

No contexto deste artigo, chamamos *ferramenta de trabalho em software* (ou simplesmente *ferramenta*, com frequência designada na gíria informática pelo termo inglês *tool*) a uma aplicação informática (conjunto de programas de computador que cooperam para um fim específico) que se destina a ajudar o programador em qualquer uma das fases do ciclo de desenvolvimento de programas, desde a análise ou especificação até aos testes finais ou documentação, auxiliando-o no desempenho da tarefa em causa (o que, em certos casos, pode mesmo significar a geração automática de código).

Desenvolver uma *ferramenta* é uma das possíveis tarefas de um programador, podendo ser motivada por um pedido externo (uma encomenda de determinado cliente), ou por uma necessidade do próprio. Tipicamente os programadores gostam de automatizar as tarefas que executam repetida e sistematicamente. Devido à complexidade que envolve e engenho que requer, esta tarefa é normalmente um desafio que se realiza com entusiasmo e emoção. Construída a *ferramenta* e comprovada a utilidade, surge a questão de a disponibilizar a outros programadores (doravante designados por *utilizadores*), os quais vulgarmente começam por pedir uma cópia que o autor tende a ceder por email ou simplesmente gravando num dispositivo de memória móvel (cdrom, flash, etc.).

A partir daí as coisas complicam-se, quer porque devido ao tamanho a cópia feita por um desses meios não é fácil, quer porque a instalação requer outros procedimentos (instalação de um outro pacote de *software*, escolha de directorias próprias, execução de uma makefile, etc.). Além dessas dificuldades de instalação surgem, depois, as de utilização; é típico que o *utilizador* solicite um guia, manual, ou outra qualquer forma de documentação para perceber a fundo a *ferramenta* e a poder usar correcta e convenientemente.

Face ao que foi exposto, urge encontrar uma solução que permita, por um lado disponibilizar a *ferramenta* de forma a que os potencias *utilizadores* a possam copiar facilmente (sem as limitações de tamanho associadas ao envio por email ou gravação num dispositivo móvel), e por outro lado, possam ter, no mínimo, documentação básica sobre instalação e uso.

A solução mais adequada, para responder com simplicidade a esses dois requisitos, é a criação de um página na Internet, ou seja um site WWW. Mas, como sempre as coisas começam simples e tornam-se complexas; se não cuidamos bem do seu nascimento, rápido se tornam imperfeitas e longe daquilo que nós gostaríamos de manter e os outros de visitar. Daí ser fundamental pensar à priori (antes do nascimento) *o que incluir no* site e *como o organizar*. Além disso, mais rápido do que se pensa, é necessário produzir outros materiais com o mesmo conteúdo mas com fins diversos.

Neste contexto que caracteriza um dos problemas com que o autor de uma *ferramenta* se depara no seu dia a dia, o presente paper vai sistematizar, na secção 2, o que está envolvido na disponibilização e documentação de um *ferramenta*. Depois, na secção 3, apresenta-se um dialecto XML, o XTDL, por nós concebido para descrever uma ferramenta de forma abstracta, isto é, independente do meio de divulgação que se quer produzir. Seguindo a boa tradição do nosso grupo, deve-se começar por especificar

formalmente o objecto com que se está a lidar, para depois surgir a possibilidade de processar a dita especificação e automatizar a produção do resultado. Assim e de seguida, na secção 4, introduz-se um conjunto de processadores, XTS, capazes de tomar como entrada uma descrição XTDL e produzir um site WWW (subsecção 4.1), um folheto (subsecção 4.2) ou um conjunto de diapositivos (subsecção 4.3). Inclui-se, também, a subsecção 5.4 onde se discute uma funcionalidade extra: a geração de um navegador conceptual, para incluir no *site*, a partir da descrição (opcional) do mapa de conceitos do domínio de conhecimento no qual a *ferramenta* se insere. Antes da conclusão, na secção 6, ainda se mostra a aplicação da linguagem especifica XTDL e da família de processadores XTS a uma *ferramenta* por nós desenvolvida, um compilador para a linguagem LISS; a secção 5 será, portanto, dedicada a apresentar este caso de estudo que constituiu a motivação para o trabalho aqui em discussão

2 Disponibilização e Documentação de Ferramentas

Uma *ferramenta* poderá ser descrita por um conjunto de características (ou descriptores) que, quer no site WWW, quer num folheto de apresentação —*datasheet*— quer num conjunto de diapositivos —*presentation*— estão sempre presentes e são a todos eles comuns. De modo a desenhar a linguagem XTDL correctamente—sendo independente do tratamento que se lhe pretende dar—é importante reflectir sobre a informação que deve ser fornecida acerca da ferramenta que se quer distribuir. Embora o sistema desenvolvido seja um contributo para a documentação de *software*, não foi de todo nossa intenção, neste trabalho pragmático, fazer um estudo teórico sobre essa área. Em [Agu03] o leitor pode encontrar uma aprofundada sistematização sobre documentação de *software*, em geral e em casos particulares, e uma extensa bibliografia sobre o tema.

Assumindo que a primeira preocupação é permitir o acesso para *download*, então é óbvio que é preciso *identificar cada uma das versões disponíveis*, associando ao identificador da versão, e à respectiva data, o URL do correspondente ficheiro. Naturalmente que a *identificação da ferramenta—designação* (ou nome abreviado) e *nome completo*—e uma *synopsis* (descrição breve) são também elementos imprescindíveis.

Falar da *equipe de desenvolvimento e manutenção* é, com certeza, outro dos requisitos. Assim será preciso descrever os *autores da ferramenta*, *instituição/empresa* a que estão associados, o *contexto* (projecto, disciplina, etc.) em que surge e *quem a mantém*. Opcionalmente, uma *data de criação* pode ser útil.

Naturalmente, convém explicar *porque foi concebida*, *para que serve*, *como está arquitectada* e *como foi desenvolvida* (estratégia de programação, tecnologia empregue, etc.). Ou seja, importa permitir a inclusão de uma *descrição*, em formato livre, e de *informação técnica*, mais estruturada, com referência à arquitectura e às tecnologias utilizadas.

É também conveniente listar a *documentação—manuais* de instalação e utilização, *exemplos, relatórios e artigos*—que a equipa de desenvolvimento/manutenção, ou outros, tenha produzido sobre a *ferramenta* em causa. Naturalmente que a indicação de *outras fontes relacionadas* é uma opção que deve considerada; sobretudo num site WWW é vulgar e útil encontrar apontadores para outros *sites* afins. Por fim, considerar a possibilidade de incluir (opcionalmente) uma *secção de novidades*, que permita

ao potencial *utilizador* ter uma visão rápida da evolução da *ferramenta* e estar alerta para determinadas questões prementes, afigura-se nos também importante.

Como acréscimo e porque defendemos o uso de *mapas de conceitos* (MCs) para descrever o conhecimento associado ao universo de discurso no seio do qual um determinado trabalho específico se desenvolve, decidimos permitir ainda a definição do MC sobre o domínio em que a ferramenta se enquadra. A intenção é possibilitar a geração de um navegador conceptual, sobre esse MC, caso ele seja especificado, o qual permita facilmente explorar o conhecimento nessa área. O MC a incluir pode ser descrito numa qualquer das linguagem existentes para o efeito. Por isso mesmo e no sentido de reutilizar esforços, decidimos usar uma de nossa autoria (muito simples e baseada em Prolog) para a qual já tínhamos desenvolvido um processador que gera o grafo e o navegador.

3 A linguagem XTDL

Nesta secção, pretende-se apresentar o Schema[Fal01, TMM01, BB01] que define o dialecto XML[BPSMM00, GP01, RH02] para descrição de ferramentas, o XTDL, no sentido de dar resposta aos requisitos identificados na secção anterior.

XTDL é constituída por um conjunto de elementos e atributos que permitem expressar os descritores que definem uma *ferramenta* independentemente da transformação e que foram acima identificados. O Schema que a seguir se apresenta reflecte esses elementos comuns e acrescenta alguns atributos que permitem controlar a geração das diversas formas de documentação contempladas pelo XTS.

3.1 O Schema

Para a geração das diferentes formas de documentar uma *ferramenta* (site que pode incluir um mapa de conceitos, diapositivos e folheto) apenas é necessário um único documento XTDL.

A documentação que se pretende gerar serve objectivos bastante distintos entre si: o site WWW, para consulta de informação geral da ferramenta—versões disponíveis, equipa envolvida no seu desenvolvimento, *download* da ferramenta, etc.—vai ser acedido por um público geral interessado em usar, ou apenas saber mais sobre os detalhes da ferramenta; o folheto (a *datasheet*), para apresentar brevemente a ferramenta—descrição técnica, arquitectura, autores, etc.—vai ser lido por um público, também genérico, que pretende conhecer a ferramenta de uma forma mais superficial; por fim, os diapositivos vão ser vistos e lidos por um público mais técnico com o intuito de conhecer e discutir mais detalhadamente a ferramenta.

Considerando estes aspectos, é unânime que serão necessárias, no nosso documento XTDL, as seguintes *tags* associadas ao elemento-raíz, `tool`, conforme se pode constatar na Figura 1:

- `team` (Equipa de desenvolvimento), o qual se divide em: `authors` (Autores); `filiation` (instituição à qual os autores pertencem); `projectName` (nome do projecto no âmbito do qual se desenvolveu a *ferramenta*); `projectDate` (data de início do desenvolvimento).
- `descritption` (Descrição) que inclui um texto livre (`introd` e `context`) e uma descrição técnica (`tech-desc`), o qual ainda permite juntar alguma explicações organizadas em items (`explanation`) e apresentar a arquitectura (`arch`).

- `distribution` (Versões disponíveis) que engloba três subelementos para identificar a versão, o ficheiro e seu *path* (`version`, `file` e `samples`).
- `docs` (Documentação disponível).
- `news` (Notícias relativas à ferramenta — nova versão, nova actualização, ...).
- `links` (Links relacionados); cada *link* tem um atributo `title` que permite dizer o texto a colocar na âncora.

Importa referir que o nome da *ferramenta* não é um elemento, mas sim um atributo da raiz (`id` e `name`). Seguindo a mesma lógica incluiu-se dois outros atributos associados a `tool`: `logo` e `lastUpdate` para permitir a indicação do logotipo criado para a *ferramenta* e para guardar a data da última actualização da descrição.

Como apresentado na figura 1, associado a cada uma das *tags* são permitidos 3 atributos: `site`; `presentation`; `datasheet`. Cada um destes atributos poderá ter um de dois valores possíveis: `yes` ou `no`. Consoante o valor desses 3 atributos, decidimos como é feita a geração em cada um dos 3 casos previstos. É de notar que cada um destes 3 atributos é opcional, pelo que deveremos pensar que estratégia seguir em caso de omissão. Assim, em caso de omissão de atributos, não é gerada nenhuma documentação. Definiu-se, no entanto, uma estratégia hierárquica para facilitar a escrita do XML.

Consideremos, a título de exemplo, o fragmento de descrição seguinte na descrição técnica (`tech-desc`) da ferramenta, esta é composta por uma explicação (`explanation`) e pela apresentação da sua arquitectura (`arch`):

```
<tech-desc site="yes">
    <explanation>
        <paragraph site="yes" presentation="yes" datasheet="yes">
            A brief description of our tool can be composed by one
            or more paragraphs.
            Each one could have one or more items (they are opcional).
            <item presentation="no" site="no">
                This first item I don't want that show up in my
                datasheet and presentation.</item>
            <item site="yes">The Second item will be shown in
            datasheet generation.</item>
        </paragraph>
        <paragraph site="no">
            Second paragraph
            <item>A simple item that will not appear in the
            site generated</item>
        </paragraph>
    </explanation>
    <arch>
        The figure below illustrates the architecture of our Tool.
        <image label="Old Architecture" site="no">old_arch.jpg</image>
        <image label="New Architecture">new_arch.jpg</image>
    </arch>
</tech-desc>
```

Para este exemplo, ao atributo `site` da *tag* `tech-desc` é associado o valor `yes`, indicando que se pretende que este elemento seja incorporado no *site*. No entanto, o

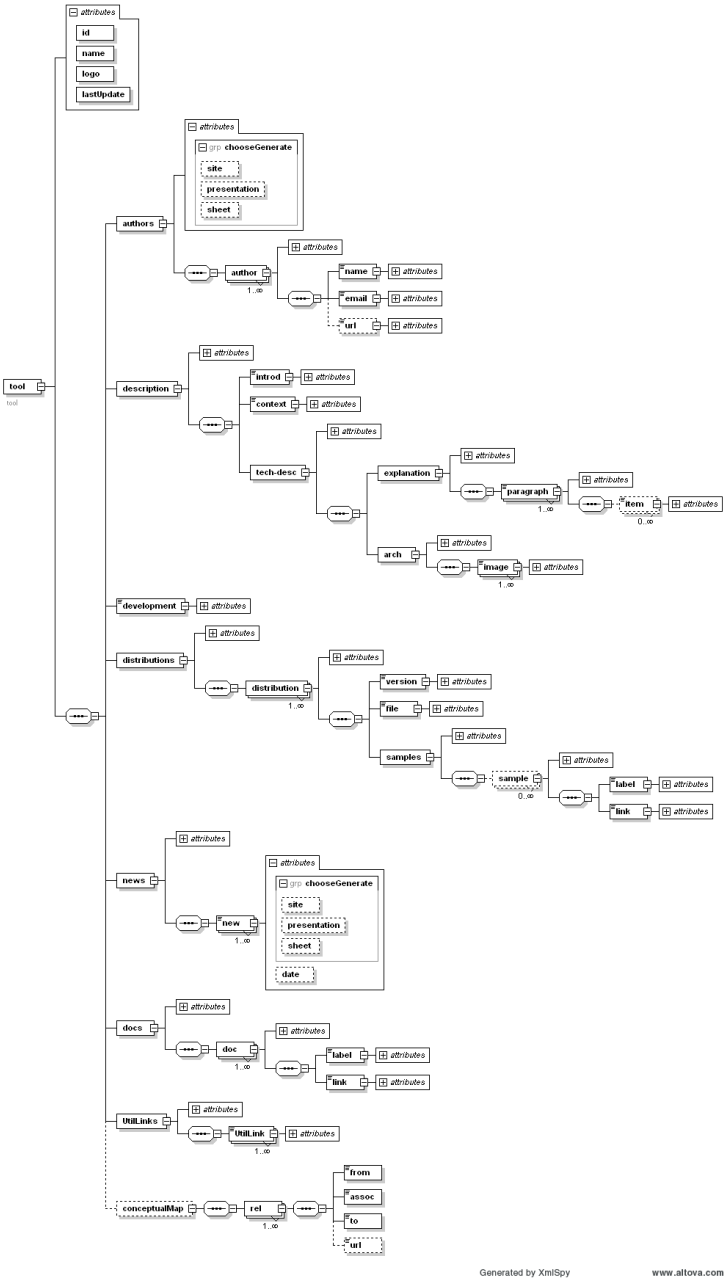

Figura 1: Schema da linguagem XTDL

elemento `explanation` não tem qualquer atributo definido; de acordo com a regra acima, o seu conteúdo seria ignorado na geração do *site* visto que o *default-value* é `no`. Contudo, seguindo uma estratégia hierárquica de herança, este nodo herda o valor do atributo `site` do seu pai (`tech-desc`), pelo que o seu conteúdo continuará a ser incluído no *site*. Apenas no caso do atributo `site` ser re-definido com o valor `no` em algum dos seus descendentes, o conteúdo será ignorado (como acontece no caso do primeiro `item` do primeiro `paragraph`, no segundo `paragraph` e na primeira `image` do elemento `arch`). O resultado da transformação aplicada ao XML apresentado (omitindo formatação própria de *site*, *datasheet* ou *presentation*) seria:

```
A brief description of our tool can be composed by one or more paragraphs.
Each one could have one or more items (they are opcional).
The Second item will be shown in datasheet generation.
The figure below illustrates the architecture of our Tool.
image = new_arch.jpg
```

Para a geração do mapa de conceitos, a linguagem XTDL prevê a definição de um conjunto de relações entre conceitos que serão agrupadas na *tag* `conceptualMap`, sendo cada relação descrita por quatro componentes a saber: `from`, `to`, `assoc` e `url`, servindo os dois primeiros para indicar a origem e destino de cada ligação, o terceiro para indicar o nome da relação que os une e por fim o quarto para associar o endereço da página respectiva.

4 O conjunto de processadores XTDL

Quando precisamos de utilizar uma ferramenta que nos seja útil, esperamos que esta seja simples, fácil de utilizar e intuitiva. Sendo assim, como *front-end* do XTS, para geração de documentação sobre ferramentas, desenvolvemos uma aplicação recorrendo à tecnologia .NET para a criação de uma *Aplicação Web* através da qual o autor pudesse submeter a descrição da sua *ferramenta* em XTDL e procedesse à geração da documentação pretendida, além de dispor da informação necessária à utilização desse conjunto de processadores. Esta *Aplicação Web* deve oferecer uma interface onde o utilizador tome as decisões relativas à geração dos 3 tipos de documentação disponíveis. A transformação de um documento XML usando XSL[Tho03, Hol01] e a apresentação do resultado é uma tarefa bastante simples na linguagem ASP. Apenas precisamos recorrer a 3 classes do C#: System.XSL, System.XML e System.XPath. O seguinte exemplo de código mostra o quão simples pode ser esta tarefa:

```
XslTransform Xslt = new XslTransform();
Xslt.Load(ourStylesheet);
XPathDocument doc = new XPathDocument("user.xml");
StringWriter fs = new StringWriter();
Xslt.Transform(doc, null, fs, null);
```

Essencialmente, após a submissão do documento XML do utilizador, é utilizado um objecto de transformação — XslTransform — onde está presente uma das nossas stylesheet — Xslt.Load(ourStylesheet), para efectuar a transformação — Xslt.Transform(doc, null, fs, null) — armazenando o resultado numa string — `fs` — que depois poderá ser utilizada para escrever a documentação pretendida (dentro das 3 hipóteses fornecidas).

Em cada uma das subsecções seguintes apresentamos a estratégia seguida para a geração de cada documentação final.

4.1 SG: Gerador de Sites

A *linguagem destino* utilizada na geração do *site* foi obviamente HTML [Ish04].

O site WWW será certamente o caso mais abrangente, isto é, aquele no qual utilizaremos a maioria dos elementos definidos no Schema apresentado na subsecção 3.1. Será usada a informação completa — se assim decidido pelo autor — relativamente: aos autores (`name, email, url`); à descrição contextual, histórica e técnica da ferramenta; às versões acessíveis para *download* e aos exemplos de utilização da ferramenta; aos documentos complementares disponíveis; aos *links* definidos para outras páginas; e ao mapa de conceitos.

Na geração desse site, pretendemos seguir um *padrão* que pode ser esquematizado da seguinte forma: o logotipo da *ferramenta* no canto superior esquerdo; o nome da *ferramenta* centrado no topo; o corpo do site subdividido em duas partes — à esquerda, os links para as componentes da descrição eleitas pelo autor para aparecerem na sua página; e no centro, a informação sobre a *ferramenta* correspondente a cada um desses links; na base da página, informação relativa aos contactos relacionados com a *ferramenta*.

Para a geração deste site WWW padrão, são necessárias várias travessias ao documento XTDL:

- a primeira para geração dos links que aparecerão na barra esquerda da página, de acordo com o valor do atributo `site`;
- a segunda para a geração da informação relativa aos *links* da travessia anterior. Esta travessia poderá, na prática, ser subdividida em tantas travessias quantas as páginas diferentes precisamos de gerar para esses links.

Para concretizar a ideia anterior, referimos que no caso do elemento `description` decidimos criar uma nova página, diferente da inicial, onde apresentaremos a descrição técnica da *ferramenta* acompanhada da arquitectura. O mesmo vai acontecer com o elemento `distribution`, para o qual também será gerada uma nova página com a informação e os *links* relativos às versões disponíveis para *download*.

A título de exemplo, mostra-se a seguir a folha de estilo XSL responsável pela primeira travessia.

```
<!-- left column -->
<td width="20%" valign="top">
    <table>
        <xsl:if test="//tool/description[@site='yes']">
            <xsl:apply-templates  select="tech-desc" mode="first" />
        </xsl:if>
        <xsl:if test="//tool/distributions[@site='yes']">
            <xsl:apply-templates  select="distributions" mode="first" />
        </xls:if>
        <xsl:if test="//tool/docs[@site='yes']">...</xsl:if>
        <xsl:if test="//tool/news[@site='yes']">...</xsl:if>
        <xsl:if test="//tool/links[@site='yes']">...</xsl:if>
    </table>
</td><!-- end left column -->

<!-- TECHNICAL DESCRIPTION -->
```

```
<xsl:template match="tech-desc" mode="first">
       <xsl:if test="(count(@site)=0]) or (@site='yes')">
          <tr><td bgcolor="#F4FEFF">
              <a href="#tech-desc">Technical Description</a></td></tr>
        </xsl:if>
    </xsl:if>
</xsl:template>
...
```

4.2 DG: Gerador de Folhetos

A *linguagem destino* utilizada na geração do folheto foi LaTeX[Lam94, GMS94].
Na geração deste documento, a ser impresso em papel para distribuição, a estratégia adoptada será diferente da proposta na secção 4.1, já que o que se pretende neste folheto é apenas uma breve apresentação da *ferramenta* e não a sua descrição por completo. Utilizaremos, além da identificação da *ferramenta* e dos autores envolvidos, a informação contida no elemento `description`—introdução, contexto, explicação técnica e arquitectura—e incluiremos um *link* para a página da *ferramenta*. No entanto, irá ser descartada informação, que neste caso nos parece irrelevante, como os *links* para outras páginas e, claro, os *links* para *download*, quer do executável correspondente às versões disponíveis, quer dos ficheiros relativos aos exemplos.

Para exemplificar o processamento de listas seguindo a herança do valor do atributo de controlo `sheet`, consideremos, em particular, o caso da geração do campo *autores* que deve figurar no folheto. Teremos de ter em atenção que a *tag* `authors` terá de ter o atributo `datasheet="yes"` (só assim deverá ser gerado o fragmento LaTeX `\author`); além deste atributo a verdadeiro, será necessário que pelo menos um dos seus descendentes não tenha o atributo `datasheet` re-definido para `no`.

```
<!-- TEMPLATE FOR "AUTHOR" -->
<xsl:template match="authors">
<xsl:if test="count(((author[@datasheet='yes'])
             | (author[count(@datasheet)=0])) &gt; 0">
  \author{<xsl:for-each select="(author[@datasheet='yes'])
             | (author[count(@datasheet)=0])">
<xsl:apply-templates select="."/>
          </xsl:for-each>}
</xsl:if>
</xsl:template>

<xsl:template match="author">
<xsl:variable name="pos"
     select="count(following-sibling::author/name[@datasheet='yes']
             | following-sibling::author/name[count(@datasheet)=0])
               - count(following-sibling::author[@datasheet='no'])"/>
<xsl:choose>
  <xsl:when test="$pos &gt; 0">
            <xsl:apply-templates select="name"/> \and </xsl:when>
  <xsl:otherwise><xsl:apply-templates select="name"/></xsl:otherwise>
</xsl:choose>
</xsl:template>
```

```
<xsl:template match="name">
<xsl:choose>
  <xsl:when test="(count(@datasheet)=0) or (@datasheet='yes')">
            <xsl:value-of select="."/></xsl:when>
</xsl:choose>
</xsl:template>
<!-- FIM DE TEMPLATE -->
```

4.3 PG: Gerador de Diapositivos

Na geração deste conjunto de diapositivos existem algumas decisões que importa salientar. A primeira de todas é, obviamente, a *linguagem destino* escolhida; neste caso decidimos utilizar na geração dos diapositivos a classe beamer[Tan06] do LaTeX[OS06].

Uma vez que cada diapositivo deverá ter uma quantidade adequada de informação, deveremos pensar que estratégia seguir para não sobrecarregar nenhum deles. Assim, nos dois primeiros diapositivos é pacífico que devemos ter: no inicial, a informação acerca dos autores que irão realizar a apresentação, o tema (nome da *ferramenta*), o local e a data; e no segundo diapositivo, o conteúdo global da apresentação, organizado em tópicos (o que nos é facilitado em LaTeX pela possibilidade de usar o comando `\tableofcontents`, dispensando uma travessia ao XTDL, como acontecia no caso do site WWW para a criação dos *links* activos na página).

A partir deste diapositivo, os restantes deverão ser preenchidos com a informação retirada de cada uma das *tags* do XTDL com o valor `yes` no atributo `presentation`. No entanto, no caso do elemento `description` que se encontra-se dividida em 3 partes—a introdução, o contexto e a descrição técnica—é necessário uma estratégia particular. Assim, cada uma destas partes dará origem a um novo diapositivo: um para a introdução; outro para o contexto; e uma série de diapositivos para a descrição técnica, a qual já apresenta uma estrutura (composta pelos sub-elementos `explanation` e `arch`), pensada para a geração de diapositivos. A *tag* `explanation` é constituída por um conjunto de `paragraph`, em que cada paragraph deverá corresponder a um diapositivo. Por fim a arquitectura (*tag* `arch`) dará origem a uma diapositivo com o texto livre e um outro por cada imagem incluída.

4.4 CMG: Gerador do Mapa de Conceitos

O mapa de conceitos que aqui se pretende gerar, deverá ser descrito através de um conjunto de factos tendo como *linguagem destino* o Prolog. A base de factos assim gerada, será depois submetida a um compilador desenvolvido pelo nosso grupo que produzirá dot[ENea06].

A base de conhecimento a ser gerada em Prolog é composta por um conjunto de factos (`map/4`), que descrevem cada associação entre um par de conceitos, na forma `map(source,association,destination,url).`; este conjunto de factos permite-nos deduzir num grafo de dependências, que pode então ser representado graficamente de modo a que o *utilizador* nele possa navegar.

A stylesheet responsável por esta tradução é a seguinte:

```
<xsl:template match="conceptualMap">
<xsl:apply-templates select="rel"/></xsl:template>
```

```
<xsl:template match="rel">
  <xsl:variable name="from" select="from"/>
  <xsl:variable name="to" select="to"/>
  <xsl:variable name="url" select="url"/>
   map('<xsl:value-of select="$from"/>', <xsl:value-of select="$assoc"/>,
       '<xsl:value-of select="$to"/>', '<xsl:value-of select="$url"/>').
```

5 Caso de estudo: LISSc, O Compilador de LISS

Nesta secção apresentamos um exemplo de aplicação prática a uma das nossas ferramentas: LISSc— um compilador de uma Linguagem de programação imperativa com Inteiros, Sequências e Sets, concebida pelo nosso grupo.
Por sermos um grupo produtor de *ferramentas* e sentirmos as dificuldades de distribuição e disponibilização que explicámos na secção 1, aplicámos o conjunto de processadores XTS à descrição da nossa *ferramenta* em XTDL e nas subsecções seguintes apresentamos o resultado obtido, por aplicação das estratégias adoptadas (secção 4).

5.1 O *site* LISSc gerado

Na secção 4.1, apresentámos aquele que seria o padrão gerado pelo processador de site WWW e também a estratégia a seguir para obter um resultado como o que se mostra na figura 2. O documento XTDL sobre o compilador LISSc foi escrito de acordo com

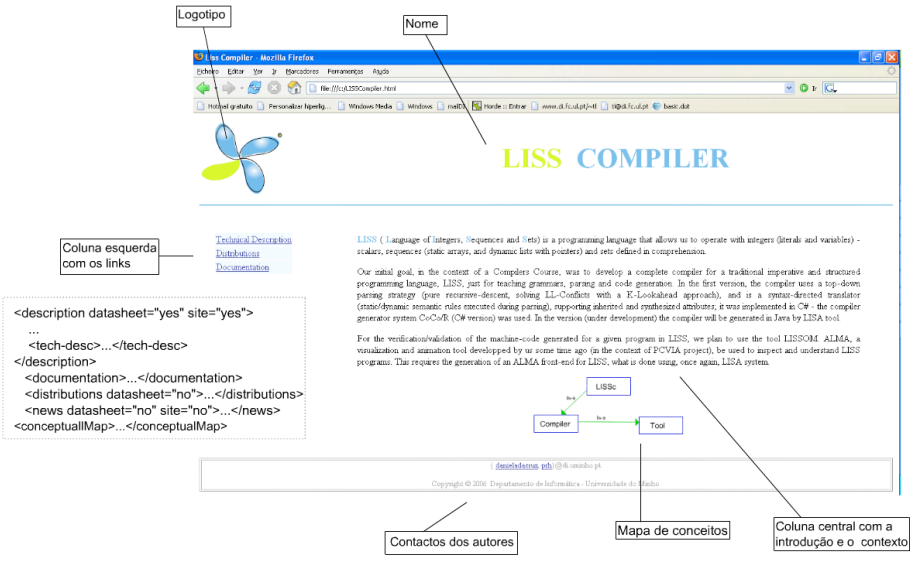

Figura 2: Página Principal do *Site* do LISSc gerado pelo SG

os princípios apresentados anteriormente, dando origem a um conjunto de páginas HTML onde poderemos encontrar a informação descritiva da *ferramenta*, bem como o conteúdo de todas as *tags* que tem valor `yes` no atributo `site`. Como já referido, adoptámos a estratégia de gerar novas página—ligadas à página principal (*homepage* visível na figura 2)—para: as versões acessíveis para *download* (ver figura 3); documentação existente; e *links úteis*.

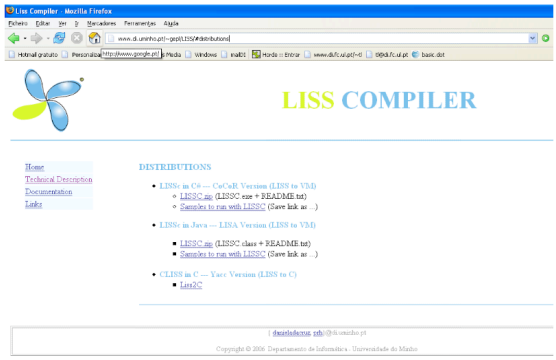

Figura 3: Página para distribuição do LISSc gerada pelo SG

5.2 O Folheto sobre o LISSc

Na secção 4.2 apresentámos a ideia e o objectivo de um folheto: informação breve e simples, destacando-se apenas os pontos relevantes referentes à *ferramenta*. Apresentámos ainda a estratégia adoptada para a geração do respectivo documento em LaTeX.

Nesta subsecção mostramos (abaixo) o resultado parcial desta estratégia, produzido pelo DG na *linguagem destino*:

```
\section{Technical Description}
\subsection{Description}
  LISSc was generated automatically from its Attribute Grammar using
  the Compiler Generator tool LISA.
  LISSc generates pseudo-code for the virtual machine VM.\\
  \begin{itemize}
     \item First version: was used CoCoR (compiler generator) and
           virtual machine MSP.
  \end{itemize}
```

e (na figura 4) o produto final, no formato PDF obtido após compilação do documento LaTeX gerado.

5.3 Os Diapositivos para apresentar o LISSc

Para o gerador de diapositivos de apoio a demos e comunicações a preocupação principal residia, como referido na secção 4.3, na quantidade de informação a distribuir

LISS Compiler

Daniela da Cruz Pedro Rangel Henriques

December 2, 2006

1 Introduction

LISSC is a Compiler for the toy imperative (or procedural) programming language LISS, that stands for Language of Integers, Sequences and Sets.

2 Context

At the beginning the language was conceived with pedagogical purposes, in the context of a Compilers Course. Nowadays it was recovered as a research project is the context of the Language processing group at University of Minho, to explore compiling techniques, generators and virtual machines.

3 Technical Description

3.1 Description

LISSC was generated automatically from its Attribute Grammar using the Compiler Generator tool LISA. LISSC generates pseudo-code for the virtual machine VM.

- First version: using CoCoR

4 Development

LISSC was developed at the Computing Department of he University of Minho by Daniela da Cruz, starting in the summer of 2005. LISS language was designed by Pedro Rangel Henriques and Leonor Barroca

- Version: 3.0

5 Site WWW

http://www.di.uminho.pt/ gepl/LISS

Figura 4: Folheto sobre o LISSc gerado pelo DG

por cada diapositivo.

No entanto, o dialecto XTDL foi pensado para deixar ao cuidado do *utilizador* a capacidade de controlar a situação e ultrapassar este problema, escolhendo a quantidade de informação a colocar em cada uma das *tags* (destacando-se, neste caso, as *tags* `paragraph` e `item`).

Apresentamos de seguida o produto final gerado pelo PG, evidenciando o resultado do critério de separação de diapositivos explicado (distribuição de texto por cada *slide*): a figura 5 mostra a estrutura, ou conteúdo, final da apresentação gerada; a figura 6 ilustra um diapositivo correspondente a um paragraph da *tag* `explanation` relativa à `tech-desc` do documento-fonte em XTDL.

Figura 5: Diapositivo que mostra a Estrutura da Apresentação do LISSc gerada pelo PG

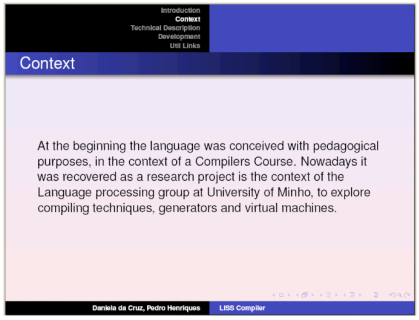

Figura 6: Diapositivo relativo a descrição técnica do LISSc gerado pelo PG

5.4 O Mapa de Conceitos do LISSc

O MC, cujo navegador irá complementar e enriquecer o site WWW, é algo bastante simples, mas que se nos afigura muito útil para a compreensão do domínio onde se enquadra a *ferramenta*. O respectivo gerador limita-se a: (1) transformar relações descritas na *tag* `rel`, do documento XTDL, em factos Prolog. Posteriormente (2) esses

factos Prolog serão traduzidos para dot do Graphviz[ENea06], de modo a que possa ser visto com o visualizador dotty[KGN06] ou WebDot[Ell06], o qual gera um diagrama navegável que é efectivamente o que nos interessa para adicionar ao *site*.

Apresentamos nesta subsecção o resultado das transformações (1) e (2) acima introduzidas. Para isso consideremos o seguinte extracto do documento XTDL referente ao compilador de LISS:

```
<conceptualMap>
     <rel>
         <from>lissc</from><assoc>is-a</assoc><to>compiler</to>
               <url>www.di.uminho.pt/~gepl/LISS</url>
     </rel>
     <rel>
         <from>compiler</from><assoc>is-a</assoc><to>tool</to>
     </rel>
 </conceptualMap>
```

Transformando (1) o mapa de conceitos anterior com a stylesheet apresentada na secção 5.4 obtemos o seguinte resultado:

```
map('lissc',is-a,'compiler', 'www.di.uminho.pt/~gepl/LISS')
map('compiler',is-a,'tool', '')
```

Este mapa de conceitos, submetido ao CMC (Conceptual Map Compiler) por nós desenvolvido, produzirá a imagem cuja descrição em dotty se apresenta a seguir.

```
digraph "LanguageProcessing@di.um.pt"
{
  node[shape=box,color="blue"];
  "LISSc" [ URL = "www.di.uminho.pt/~gepl/LISS" ]
  "Comppiler" [ URL = "" ]
  "Tool" [ URL = "" ]
  "LISSC" -> "Compiler" [ label="is-a" color="green"]
  "Compiler" -> "Tool" [label="is-a" color="green"]
}
```

Na figura2 (mostrada atrás) pode ver-se a dita imagem depois de ser processada pelo Graphviz, já incorporada no site WWW do LISSc.

6 Conclusão

Ao longo deste artigo apresentamos *uma aplicação* com o intuito de mostrar que *a anotação de documentos em XML e o seu processamento com as tecnologias XSL associadas* continua a ser *uma forma elegante e eficiente de programar*—especificar o problema e codificar uma resolução. Em muito pouco tempo e com uma dificuldade reduzida, resolvemos um problema complexo que nos surgiu no âmbito da nossa actividade como construtores de *ferramentas* de *software*, problema esse que é comum aos outros autores de *ferramentas*.

Usando um sistema de anotação (uma linguagem) específico de uma classe de problemas—definido formalmente através de um Schema XML—obtemos um especificação rigorosa

e declarativa de cada problema concreto dentro dessa classe. No caso discutido, descrevemos cada *ferramenta* concreta que pretendemos distribuir em XTDL, o dialecto XML criado para o efeito. É claro que a dita linguagem XTDL e os processadores associados não limitam a sua aplicabilidade à publicação de *ferramentas* de *software*, mas quisemos restringirmo-nos ao caso que serviu de motivação e que estava no nosso âmbito de trabalho.

Essa especificação declarativa enfatiza a estrutura do objecto que se quer tratar (as características da ferramenta a disponibilizar e documentar) e a sua semântica estática (regras contextuais que tem de ser observadas), de uma forma independente da transformação a operar. Além disso e por ser uma anotação específica de um domínio concreto de problemas, torna-se claramente fácil de usar e interpretar por quem trabalha nesse domínio. Recorrendo, depois, a um sistema de produção—formado por regras condição-reacção, também ele escrito rigorosamente num dialecto próprio de XML, o XSL—descreve-se a forma de processar a especificação referida acima.

Esta aproximação permitiu-nos pensar e desenvolver, não um processador, mas um conjunto de processadores que transformam a descrição XTDL de uma *ferramenta* em três tipos diferentes de suportes (meios de comunicação) para divulgação da dita *ferramenta*. É claro que várias outras transformações poderiam ser feitas para produzir outras peças documentais, mas pareceu-nos que as três apresentadas eram suficientes para ilustrar a ideia.

A quantidade de artefactos, actualmente existentes nos vários ambientes de programação, para reconhecer documentos XML e para os processar com folhas de estilo XSL, permitiu que se integrasse muito rápida e facilmente as componentes do XTS de modo a disponibilizá-las através de uma Aplicação Web. Para exemplificar a ideia, utilizaram-se 3 classes standard do ambiente .NET.

O trabalho realizado, como se disse atrás em tempo muito curto, terá, obviamente, de ser apurado no futuro próximo, quer quanto aos elementos previsto no XTDL, quer quanto ao XTS; aí teremos de melhorar a apresentação do resultado produzido por cada um dos geradores e de pensar em incrementar o conjunto, concebendo novos transformadores. Mas a lição retirada a nível da abordagem à resolução de problemas é que importa reter e discutir.

Agradecimentos

Os autores querem expressar o seu agradecimento a Elisabete Ferreira e Patrícia Oliveira pelo seu empenho e ajuda na implementação da linguagem XTDL e das ferramentas da *suite* XTS.

Referências

[Agu03] Ademar Aguiar. *Framework Documentation, a Minimalist Approach.* PhD thesis, DEEC, Faculdade de Engenharia da Universidade do Porto (FEUP), 2003.

[BB01] Paul V. Biron and Paul V. Biron. XML Schema part 2: Datatypes. `http://www.w3.org/TR/xmlschema-2`, May, 2001.

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML). World Wide Web Consortium, October, 2000. `http://www.w3.org/TR/REC-xml`.

[Ell06] John Ellson. WebDot Home Page, Dec., 2006. `http:///www.graphviz.org`.

[ENea06] John Ellson, Stephen North, and et al. Graphviz - Graph Visualization Software, Dec., 2006. `http:///www.graphviz.org`.

[Fal01] David C. Fallside. XML Schema part 0: Primer. `http://www.w3.org/TR/xmlschema-0`, May, 2001.

[GMS94] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[GP01] Charles F. Goldfarb and Paul Prescod. *XML Handbook*. Prentice Hall, 4th edition, 2001.

[Hol01] G. Ken Holman. *Definitive XSLT and XPath*. Prentice Hall, 2001.

[Ish04] Masayasu Ishikawa. HyperText Markup Language (HTML) Home Page. `http://www.w3.org/MarkUp/`, February, 2004.

[KGN06] Eleftherios Koutsofios, Emden Gansner, and Stephen North. dotty - A Customizable Graph Editor, Dec., 2006. `http://www.graphviz.org/cgi-bin/man?dotty`.

[Lam94] Lambert Lamport. *LaTeX A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1994.

[OS06] Tobias Oetiker and Alberto Simoes. *Uma não tão pequena introdução ao LaTeX*. Dep. de Informatica, Escola de Engenharia da Universidade do Minho, 2006.

[RH02] José Carlos Ramalho and Pedro Henriques. *XML & XSL Da Teoria à Prática*. FCA Editora, 1.st edition, 2002.

[Tan06] Till Tantau. The LaTeX Beamer Class Homepage, Dec., 2006. `http://latex-beamer.sourceforge.net`.

[Tho03] Henry Thompson. The Extensible Stylesheet Language (XSL) Family. `http://www.w3.org/Style/XSL/`, 2003.

[TMM01] Henry S. Thompson, Murray Maloney, and Noah Mendelsohn. XML Schema part 1: Structures. `http://www.w3.org/TR/xmlschema-1`, May, 2001.

XML e Preservação Digital

José Carlos Ramalho[1] and Miguel Ferreira[1] and Rui Castro[2] and Luis Faria[2] and Francisco Barbedo[2] and Luis Corujo[2]

[1] Dep. Informática, Universidade do Minho
[2] Instituto dos Arquivos Nacionais, Torre do Tombo

Resumo Actualmente estamos a substituir gradualmente os documentos físicos por documentos digitais. A constatação de que cada vez há mais informação apenas em suporte digital levanta uma série de preocupações relacionadas com a preservação de todo este manancial de informação. Não é de estranhar, portanto, que a Preservação Digital tenha emergido como uma área de investigação que tem adquirido cada vez mais importância. A principal preocupação de todos os que tentam contribuir para a área é como garantir uma maior longevidade ao material digital que é produzido diariamente. Por exemplo, material produzido está dependente de plataformas de hardware e software que normalmente se tornam obsoletos em 5 anos. O XML como formato neutro para a representação de informação surge naturalmente neste contexto. São já vários os dialectos XML produzidos e usados para e na Preservação Digital: PREMIS, METS, NISO MIX, etc. Neste artigo, caracterizámos o estado actual desta área, identificando várias normas e mostrando com um caso de estudo real (RODA: Repositório de Objectos Digitais Autênticos), como é que se podem usar e combinar aquelas normas numa solução de Preservação Digital.

1 Introdução

O Instituto de Arquivos Nacionais da Torre do Tombo (IAN/TT), tem na sua função de preservação histórica um grande desafio perante o crescimento da produção de documentos digitais pela função pública, devido à sua própria evolução no sentido do governo electrónico. Não existem actualmente estruturas que suportem os processos de incorporação e gestão de informação de arquivo electrónica. É premente garantir a preservação dos documentos digitais e o seu valor evidencial, a autenticidade, para que os testemunhos das actividades das organizações públicas sejam guardados em memória social e patrimonial.

Foi com esse objectivo que se iniciou o projecto RODA (Repositório de Objectos Digitais e Autênticos). Numa primeira instância o RODA visa a construção de um protótipo exemplificativo de uma solução de preservação digital. O protótipo pode, posteriormente, vir a ser desenvolvido na forma de produto na escala necessária para responder cabalmente às necessidades das organizações no que respeita à perservação de objectos digitais de conservação permanente.

Procura-se desta forma iniciar um processo que leve o IAN/TT a responder às solicitações governamentais e comunitárias no contexto do governo electrónico.

Neste artigo, apresentam-se de forma sintética os passos que conduziram à implementação do protótipo do RODA. Primeiro discute-se a informação que se vai guardar

e como é que esta irá ficar organizada. Depois apresenta-se a primeira abordagem à arquitectura do RODA. E, por fim, conclui-se indicando os passos e as escolhas que guiaram a sua implementação.

2 Arquitectura do Repositório

Qualquer repositório digital tem uma parte substancial da sua estrutura assente em normas de metainformação. Do repositório mais simples, como um repositório doméstico de música digital, até ao repositório governamental criado para preservar o conhecimento e o legado histórico de uma população, a metainformação é usada como suporte das funcionalidades mais básicas do repositório, e. g. facilitando o acesso aos objectos digitais armazenados.

Um repositório digital que tem como objectivo a preservação a longo prazo dos materiais custodiados tem que integrar diversos tipos de metainformação:

Descritiva – é essencial para a organização do repositório e para o acesso aos objectos digitais armazenados.

Preservação – é essencial para garantir a autenticidade dos objectos armazenados fornecendo evidências da sua proveniência e de todas as acções sobre eles realizadas.

Técnica – é essencial para garantir o bom estado dos objectos no repositório e para garantir o respectivo acesso continuado.

Estrutural – é essencial para organizar objectos digitais complexos (ex: um livro é composto por capítulos, estes por secções e estas por páginas).

Em termos funcionais, a grande maioria dos repositórios digitais segue o modelo de referência da OAIS ("*Open Archival Information System*") [TS04,fSDS02]. Este foi também o modelo escolhido para a implementação do RODA (fig.1).

Pode-se descrever este modelo funcional da seguinte maneira: o produtor prepara a informação que quer preservar organizando-a num pacote especial (SIP - *"Submission Information Package"*); estes pacotes são enviados ao sistema que sabe analisá-los, verificá-los e retirar de lá a informação arquivando-a no repositório, transformando um SIP num AIP (*"Archival Information Package"*; o administrador pode realizar acções (verificação, correcção, preservação,...) sobre os AIPs; por sua vez, o consumidor pode realizar pesquisas sobre os AIPs e a dada altura pode solicitar um determinado objecto digital que lhe é oferecido na forma de um DIP (*"Dissemination Information Package"*), ou seja, o AIP solicitado é transformado num DIP que é fornecido ao consumidor.

Esta explicação resume o modelo OAIS e resume também o funcionamento pretendido no RODA.

Na implementação deste modelo começou-se por limitar o tipo de objectos digitais que se irão tratar (no contexto de um protótipo não seria viável contemplá-los todos). No protótipo inicial do RODA foi decidido que seriam contemplados os seguintes tipos de objectos digitais:

texto estruturado – que poderá conter tabelas e imagens fixas. O formato de preservação para texto estruturado será o PDF/A.

Este formato é capaz de preservar documentos de texto estruturado (com tabelas e imagens) mantendo o aspecto e paginação do documento original.

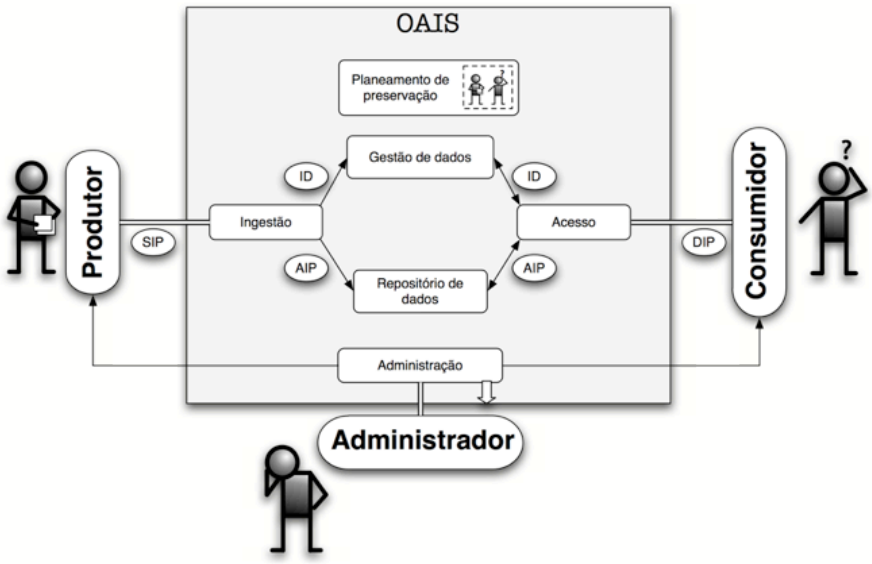

Figura 1. Modelo OAIS

as imagens bidimensionais fixas – fotografias, digitalizações, ... O formato de preservação para imagens fixas será o TIFF não comprimido. Este formato é aberto e muito bem suportado por inúmeras ferramentas de acesso livre e portanto é um bom formato de preservação.

bases de dados relacionais – para o protótipo prepararam-se interfaces com o MS Access, MS SQL Server e Oracle. O formato de preservação de bases de dados relacionais será o XML. Mais precisamente, as bases de dados serão migradas para DBML[JLRH02] ("*Data base Markup Language*"). Os alvos de preservação serão a estrutura da base de dados (tabelas e relações entre tabelas) e os dados. As funcionalidades associadas a *stored procedures* não serão alvo de preservação nesta fase. A eventual perda de informação que possa ocorrer ao não perservar outras funcionalidades (como os *stored procedures*) não é preocupante, porque a representação original de todos os objectos ingeridos será preservada e, portanto, será sempre possível no futuro derivar novas representações mais ricas que já incluam funcionalidades extra como as *stored procedures*.

Para mais detalhes sobre as resoluções relativas às taxionomias de objectos pode ser consultado [Bar06b]. As propriedades significativas a preservar sobre cada tipo de objecto foram alvo de alguma discussão e o resultado final é comparável ao estudo publicado por Ruusalepp[Ruu02] e é apresentado na secção seguinte.

3 Estrutura interna do repositório

Para definir a estrutura do repositório houve que decidir que informação se iria guardar e como é que esta seria estruturada. Além das representações físicas dos objectos é necessário guardar informação descritiva para facilitar o acesso e a procura, informação sobre as características originais do objectos (informação técnica) e informação de preservação sobre cada acção que é realizada sobre o objecto pois só assim se poderá garantir a autenticidade do mesmo e a sua possível reutilização como meio de prova ou outro. A estas há ainda a adicionar a informação estrutural que irá servir como agregador/organizador dos pacotes de informação que irão circular na arquitectura: SIP, AIP e DIP.

Ou seja, vamos ter uma solução que combina metainformação descritiva, técnica e de preservação. Dos objectivos do projecto cedo se concluiu que a metainformação descritiva e de preservação iriam representar um papel principal enquanto que a técnica teria um papel secundário. De realçar que todos os esquemas de metainformação adoptados e apresentados nas secções seguintes são dialectos XML.

3.1 Metainformação Descritiva

Há várias linguagens de anotação XML para especificar metainformação descritiva: Dublin Core[Wei97], MARC[mar06], Encoded Archival Description (EAD)[ead98].

Como o RODA será um arquivo digital interessava adoptar um esquema de metainformação descritiva multinível que permitisse descrever de forma estruturada as entidades intervenientes e as suas relações hierárquicas, pelo que a escolha recaiu sobre o EAD.

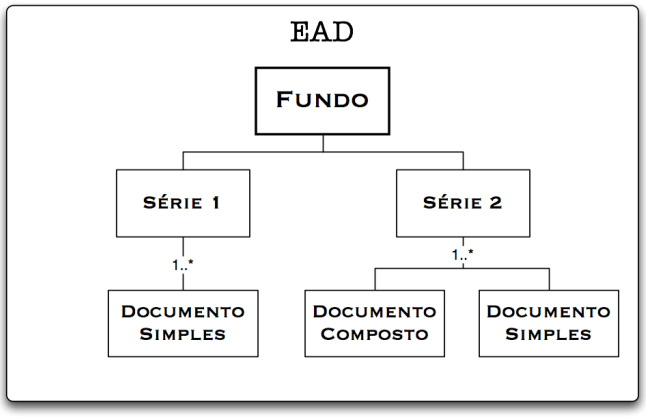

Figura 2. Esquema de um EAD exemplo

O EAD (*Encoded Archival Description*) permite definir metainformação descritiva. Esta metainformação descreve a informação de forma contextual, ajudando a categorizar e a procurar a informação. A informação deste tipo é utilizada por motores de busca para procurar informação.

Uma instância EAD contém três partes:

<eadheader> - contém informação sobre a metainformação em si.

<frontmatter> - contém informação conveniente para a renderização ou publicação da metainformação.

<archdesc> - contém a descrição de um fundo documental e a informação contextual e administrativa associada.

Cada instância contêm um ou mais elementos <c>. Estes elementos podem ser múltiplos e estar aninhados, criando uma estrutura hierárquica. Cada elemento tem um identificador único e um nível (*level*, fig. 2), que pode ter o valor[3]:

fundo - o conjunto de todos os arquivos, independentemente da sua forma ou formato, organicamente criados e/ou acumulados por certa pessoa, família, ou instituição no decurso das suas actividades ou funções.

série - conjunto de documentos reunidos porque fazem parte da mesma acumulação, processo de inserção ou actividade,ou partilham uma forma particular, ou outra qualquer relação relativa a sua criação, ingestão ou uso.

documento composto - uma unidade organizada de documentos agrupados para uso do criador ou no processo de organização arquivística, porque são relativos ao mesmo assunto ou actividade. É normalmente a unidade básica de uma série.

documento simples - A mais pequena e intelectualmente indivisível unidade de arquivo.

Cada nível de descrição contém informação descritiva adequada, seguindo o modelo da ISAD(G)[oA99]. Como exemplos de campos existem: título, datas importantes, historia biográfica, história custodial, âmbito e conteúdo, existência e localização dos originais e cópias, etc.

3.2 Metainformação de Preservação

Em 2003 a OCLC (*Online Computer Library Center*) e a RLG (*Research Libraries Group*) estabeleceram o grupo de investigação *PREservation Metadata: Implementation Strategies* (PREMIS). Em Maio de 2005 este grupo apresentou o seu relatório final, o *Data Dictionary for Preservation Metadata*[Gro05], que define o esquema que se resolveu adoptar no RODA.

O esquema está organizado segundo um modelo simples com cinco tipos de entidades envolvidas nas actividades de preservação digital:

Object - ou Objecto Digital, é a unidade discreta de informação no formato digital

[3] estes valores são os considerados pela nossa implementação, usando o atributo *otherlevel* do EAD, na definição oficial do EAD estes valores diferem um pouco

Intellectual Entity - é um conjunto coerente de conteúdos, que pode ser razoavelmente descrito como uma unidade (ex. um livro, uma imagem, uma base de dados). Uma *Intellectual Entity* pode conter outras *Intellectual Entities*, por exemplo um livro pode conter uma imagem.

Event - é uma acção que envolve pelo menos um *Object* ou *Agent* conhecidos pelo repositório de preservação.

Agent - é uma pessoa, organização ou programa associado com eventos de preservação (*Events*) no tempo de vida de um *Object*.

Rights - é um conjunto de um ou mais direitos ou permissões relativos a um *Object* e/ou *Agent*

O *PREMIS Data Dictionary* inclui unidades semânticas para *Objects*, *Events*, *Agents* e *Rights*. O quinto elemento no modelo, *Intellectual Entity*, foi considerado fora do contexto deste *Data Dictionary* pois é bem servida pelos esquemas de metainformação descritiva existente (EAD, MARC, MODS, Dublin Core, etc.) e porque é demasiado específica do domínio em consideração.

No *PREMIS Data Dictionary* a entidade *Object* tem três subtipos: *representation*, *file* e *bitstream*. Um *file* é uma sequência de *bytes* com ordem e nome, reconhecida por um sistema operativo. Um *file* tem propriedades como permissões, tamanho e data da última modificação. Um *bitstream* é um conjunto de dados dentro de um *file*, que tem algumas propriedades comuns significativas para efeitos da preservação digital. Uma *representation* é um conjunto de *files*, incluindo metainformação estrutural, necessários para renderização razoável de uma Entidade Intelectual (*Intellectual Entity*).

A entidade *Event* agrega metainformação sobre acções. Um repositório de preservação irá criar *Events* por variadas razões. Documentação sobre acções que modificam (ou seja, criam uma nova versão) de um objecto digital é fundamental para manter a proveniência digital, um elemento chave para a autenticidade. Acções que criam relações ou que modificam relações existentes são importantes para explicar as mesmas relações. Até acções que não alteram nada, como validações e análises à integridade nos objectos, podem ser importantes registar para efeitos de gestão.

3.3 Metainformação Técnica

É com esta metainformação que se descrevem as características técnicas dos ficheiros e dos seus formatos. De momento, no RODA, utiliza-se NIZO MIX[nis06] para imagens e documentos de texto e DBML para as bases de dados relacionais.

O NIZO MIX define um conjunto normalizado de elementos de metainformação para imagens digitais. O esquema utilizado data de 2002, no entanto está neste momento em período de comentário a versão de 2005, que vem trazer uma nova organização ainda mais compatível com o PREMIS.

A versão de 2002 divide a metainformação técnica em quatro secções:

1. ***Basic Image Parameters*** - que agrupa elementos fundamentais para a reconstrução do ficheiro digital como uma imagem renderizável em interfaces electrónicas.
2. ***Image Creation*** - algo como metainformação técnica descritiva, dá informação sobre aspectos logísticos e condições administrativas relativas à captura da imagem digital.

3. ***Imaging performance assessment*** - o princípio operativo desta secção é manter os atributos da imagem inerentes à sua qualidade. Estes elementos servem como métricas para medir a fidelidade da imagem corrente e dos resultados de técnicas de preservação, especialmente a migração.
4. ***Change history*** - esta secção tem a função de documentar os processos aplicados aos dados da imagem no ciclo de vida desta.

O DBML[JLRH02] permite descrever uma base de dados no seu todo: informação sobre o ambiente de funcionamento, estrutura da base de dados e a informação presente na base de dados no momento da captura.

Os objectos que irão circular nos fluxos do repositório têm uma estrutura abstracta ilustrada na figura 3.

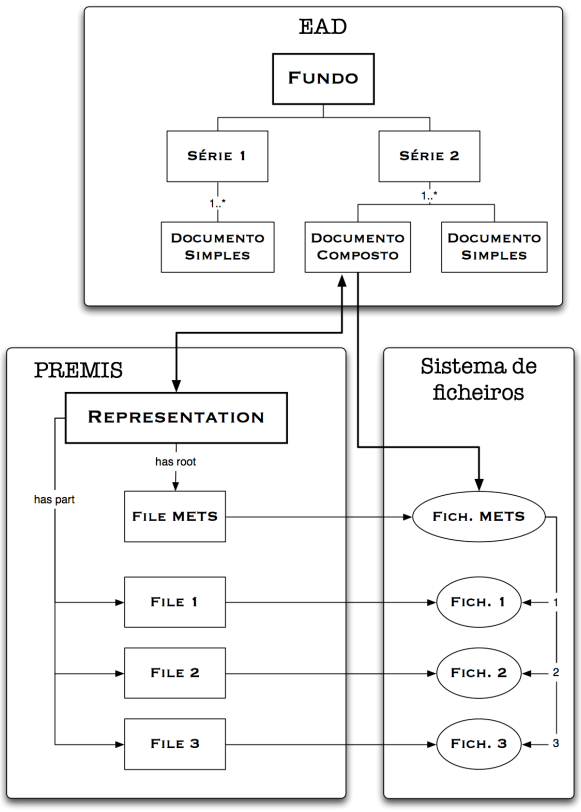

Figura 3. Esquema completo (excepto m.i. técnica)

Esta estrutura é depois implementada com algumas variações materializando os objectos que se designaram por SIP, AIP e DIP, que se definem nas secções seguintes.

3.4 SIP, AIP e DIP

Tendo decidido os esquemas de metainformação a usar no RODA passa a ser possível definir os dados que vão circular nos fluxos do protótipo.

O SIP ("Submission Information Package") é o formato usado para ingerir conteúdos no repositório digital. É composto pelo objecto digital (a sua representação física), por um conjunto de metainformação (descritiva em EAD, técnica e de preservação (em PREMIS – todo o registo de eventos previamente aplicados àquele objecto) e por uma descrição estrutural (uma espécie de lista descritiva do que vai dentro do pacote). Esta descrição estrutural é feita usando outra linguagem XML, o METS[met06,McD06] (Metadata Encoding & Transmition Standard) é uma norma para codificar metainformação descritiva, administrativa e estrutural sobre objectos guardados num repositório digital).

Na ingestão do SIP pelo repositório, a metainformação é dividida nos seus componentes funcionais que recebem um tratamento diferenciado. Neste processo, um SIP é transformado num AIP ("Archival Information Package"). Um AIP representa a forma de como um objecto digital é armazenado no repositório. O AIP tem a ele associadas a metainformação técnica e de preservação que são essenciais para a execução dos eventos de preservação. A metainformação descritiva é armazenada separadamente numa base de dados pois tem outros objectivos como facilitar o acesso e a procura de objectos digitais no repositório.

O DIP ("Dissemination Information Package") ao contrário dos outros dois tem um formato mais livre. O DIP é o formato de saída do repositório e podemos ter vários. Numa fase inicial serão suportados três DIP: um que corresponde a uma cópia da representação original, outro que corresponde a uma vista Web sobre um determinado objecto e outro que corresponde a um empacotamento num ficheiro comprimido em formato ZIP do objecto digital.

4 Implementação

Implementar um repositório de raiz é um trabalho bastante extenso e fora dos objectivos do RODA. Existem várias iniciativas *open-source* nos quais um repositório deste tipo se pode basear, mas há dois candidatos que se destacam no panorama actual: DSpace e Fedora.

O DSpace[4] é um repositório digital *open-source* para instituições de investigação. Desenvolvido numa cooperação entre a biblioteca do MIT (*Massachusetts Institute of Technology*) e os Laboratórios da Hewlett-Packard, o DSpace está disponível sob uma licença *open-source* BSD para instituições de investigação o poderem utilizar na sua forma original, ou modificar e estender conforme as necessidades. Muitas instituições de investigação por todo mundo utilizam o DSpace como solução para os mais variados tipos de arquivos digitais.

O Fedora é uma plataforma tecnológica *open-source* que oferece uma arquitectura flexível de serviços para gestão e disseminação de conteúdos. Tem no seu núcleo um modelo de dados totalmente flexível que suporta múltiplas vistas/disseminações de

[4] `http://dspace.org`

cada representação digital e das relações entre elas. Estas representações podem encapsular conteúdos geridos localmente ou fazer referência a conteúdos remotos. Vistas/disseminações dinâmicas são possíveis associando *web services* às representações. As representações existem dentro de uma arquitectura de repositório que suporta uma variedade de funções de gestão. Todas as funções do Fedora, tanto ao nível da representação como a nível do repositório, são expostas como *web services*. Estas funções podem ser protegidas com políticas de controlo de acessos de granularidade fina.

Esta combinação de características faz do Fedora uma solução atractiva em vários domínios. Alguns exemplos de aplicações que foram construídas sobre o Fedora incluem: gestão de bibliotecas, sistemas de produção de multimédia, repositórios de arquivo, repositórios institucionais, bibliotecas digitais para educação.

4.1 DSpace ou Fedora?

O passo seguinte foi a selecção da plataforma tecnológica sobre a qual seria desenvolvido o protótipo. Com esse objectivo elaborou-se um caderno de encargos bastante detalhado sobre os requisitos funcionais do projecto[Bar06a]. À luz destes requisitos fez-se um estudo comparativo das duas plataformas cujos resultados se apresentam de forma resumida no gráfico da figura 4.

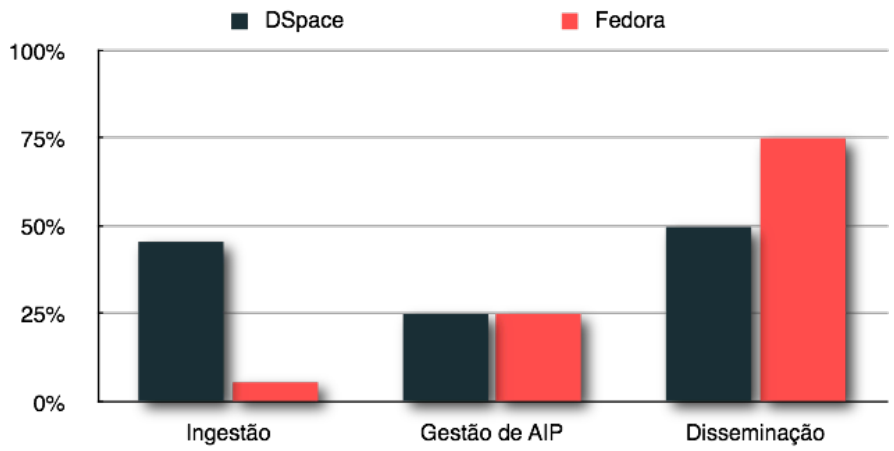

Figura 4. Comparação de requisitos entre as 2 plataformas

Como se pode ver nesta mostra de resultados, os requisitos foram agrupados nas três componentes funcionais principais do modelo OAIS: Ingestão, Gestão de AIP e Disseminação.

Observando a figura 4 seria de prever que o DSpace seria a plataforma seleccionada uma vez que ganha no cumprimento do componente principal que é a ingestão e está a par na manutenção do repositório.

No entanto, depois de se comparar a estrutura abstracta pretendida para o RODA (figura 3) com as estruturas que as duas plataformas implementam verificou-se que a estrutura do DSpace seria completamente desadequada para o fim pretendido.

O DSpace em termos de funcionalidades para o utilizador está mais completo, mas impõe uma estrutura de dados interna que é desadequada aos objectivos do RODA o que obrigaria o uso de "remendos"de modo a ser possível utilizar um esquema de metainformação descritiva hierárquico (EAD).

O Fedora é a solução mais adequada para o RODA porque não traz qualquer tipo de restrições em termos de esquemas de metainformação que se queiram usar e possui uma arquitectura de serviços que possibilita que funcionalidades sejam adicionadas ao repositório de forma elegante e independente da implementação do próprio repositório. Esta decisão coincide com a conclusão de um estudo comparativo de repositórios baseados em software livre levado a cabo no âmbito do projecto "Open Access Repositories in New Zeland"[iNZ06].

4.2 Fedora: implementação

No Fedora a unidade de informação é o objecto. Toda a informação (e metainformação) terá de fazer parte de um ou mais objectos. Um objecto está estruturado em 4 partes distintas:

PID - identificador único persistente.

Descrição - metainformação interna, propriedades e relações. Este componente do objecto é sempre necessário para a gestão interna dos objectos por parte do sistema. A metainformação interna e as propriedades são obrigatórias, as relações são opcionais.

Itens - conjunto dos ficheiros de informação e/ou metainformação contidos no objecto (*datastreams*, na terminologia do Fedora). Um objecto tem no mínimo um ficheiro com metainformação no esquema Dublin Core (DC). Este ficheiro é incluído por omissão em cada objecto e contem obrigatoriamente os campos `identifier` e `title`.

Serviços - conjunto de funcionalidades associadas ao objecto. Por omissão, a cada objecto criado é associado o serviço *Default Disseminator*. Este serviço permite que o objecto seja disseminado na sua forma mais básica, disponibilizando o acesso às propriedades e aos ficheiros. Outros serviços podem (e devem) ser associados aos objectos conforme as necessidades do próprio repositório ou da comunidade de interesse.

4.3 Tipos de Objectos

O RODA irá distinguir 3 tipos de objectos: *Objectos de Descrição (OD)*, *Objectos de Representação (OR)* e *Objectos de Preservação (OP)*. Os *Objectos de Descrição* guardarão informação descritiva, os *Objectos de Representação* conterão uma representação de um documento descrito num *Objecto de Descrição* e os *Objectos de Preservação* serão usados para guardar metainformação de preservação (PREMIS).

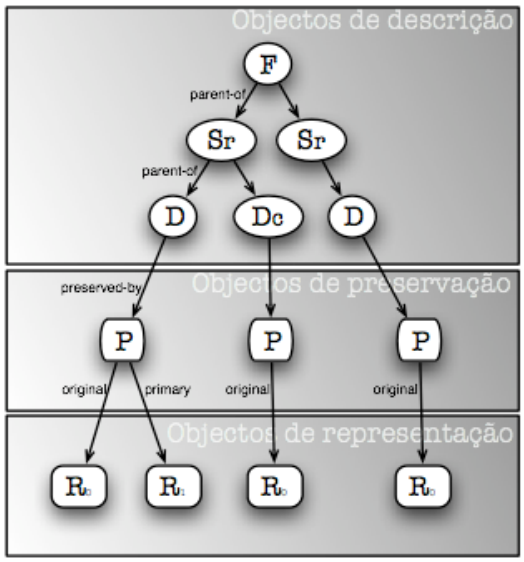

Figura 5. Objectos do* RODA *divididos por camadas

Todos os objectos relativos a um determinado fundo presente no repositório estarão relacionados através do mecanismo de relações do Fedora (usando RDF[5] – Resource Description Framework) formando uma árvore de descrição arquivística. Na raiz da árvore estará um *OD* que descreve o fundo, conectados a este estarão as descrições dos sub-fundos ou séries e estes por sua vez estarão conectados às unidades de descrição que forem necessárias para descrever adequadamente o fundo. Os *OP* estarão ligados às folhas dos *OD* e guardarão toda a metainformação de preservação relativa às representações da entidade intelectual descrita no *OD*. Por sua vez, os *OR* estarão associados aos *OP* e conterão as representações da entidade intelectual.

4.4 Conteúdos dos Objectos

Para cada tipo de objecto do repositório, *OD*, *OR* e *OP*, é necessário definir o seu conteúdo.

Os OD *são constituídos por:*

RELS-EXT - documento XML/RDF com informação sobre as relações entre o objecto e outros objectos do repositório.

DC - ficheiro com metainformação Dublin Core em formato XML. Este ficheiro está presente em todos os objectos Fedora por omissão. No RODA apenas terá o titulo (title) e o identificador (identifier) que representam o mínimo exigido pelo Fedora.

[5] `http://www.w3.org/RDF`

EAD - ficheiro com metainformação descritiva EAD em formato XML. Este ficheiro XML não será um EAD, mas um EADPART. Este formato será basicamente um elemento `<c>` do EAD mas chamado `<eadpart>` com os mesmos atributos e elementos do elemento `<c>`. A árvore de descrição do RODA será uma árvore de objectos fedora contendo ficheiros no formato EADPART (para efeitos de optimização houve que optar por uma granularidade mais fina).

Por sua vez, os OP *são constituídos por:*

RELS-EXT - (ver descrição acima).
DC - (ver descrição acima).
PREMIS+ - ficheiro(s) com metainformação de preservação PREMIS.

Por fim, os OR *têm a seguinte estrutura :*

RELS-EXT - (ver descrição acima).
DC - (ver descrição acima).
METS - ficheiro com metainformação estrutural caso a representação seja composta por mais do que um ficheiro.
FICHEIRO+ - ficheiro(s) que compõem a representação.

4.5 Relações entre Objectos

As relações entre os vários objectos estarão descritas nos ficheiros RELS-EXT de cada um dos objectos que possui ligações a outros. Este(s) ficheiro(s) está(ão) no formato XML/RDF e descreve(m) as relações com outros objectos através de triplos (ex. `<info:fedora/roda:1> <roda:parent-of> <info:fedora/roda:2>`).

As relações entre os objectos descritos acima serão também de 3 tipos e serão as seguintes:

parent-of - relação entre dois *OD*.
preserved-by - relação entre um *OD* e um *OP*.
representation-(original/primary/alternative) - relação(ões) entre um *OP* e os respectivos *OR*. Esta relação pode ter três nomes diferentes porque é necessário distinguir entre a representação original (*SIP*), a representação primária (*AIP*) e eventuais representações alternativas (*DIP*).

5 Arquitectura funcional do protótipo

O Fedora oferece uma interface ao utilizador muito pobre, pouco amigável e insuficiente para os requisitos do repositório. Além disso existe um conjunto de requisitos que não estão satisfeitos pelo Fedora, como por exemplo: ingestão com *workflow*, tarefas de gestão dos AIPs, sistemas de procura e navegação da informação baseados nos novos esquemas de metainformação, etc. O Fedora foi desenvolvido como uma arquitectura de serviços e todas as funcionalidades associadas são disponibilizadas como *web services*. Esta arquitectura permite que o repositório cresça em funcionalidades sem que

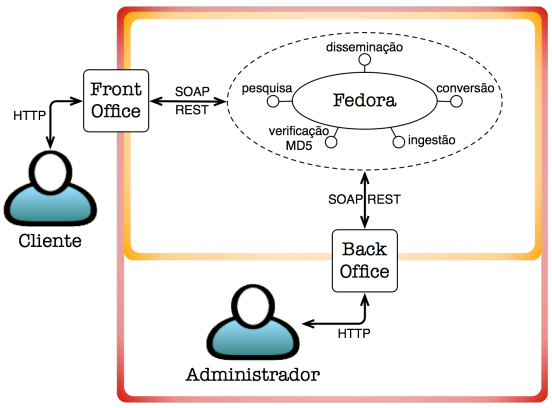

Figura 6. Arquitectura funcional do protótipo: Front e Back Office

o seu núcleo seja alterado e torna cada novo serviço independente da tecnologia usada noutros serviços já existentes não prejudicando no entanto a interoperabilidade entre os mesmos. Todas as componentes necessárias ao cumprimento dos requisitos serão desenvolvidas através de *web services* do Fedora e uma interface gráfica será disponibilizada para estes serviços na forma de um *Front Office*, para o acesso por parte dos clientes, e de um *Back Office*, para administração.

Como se pode observar na figura 6 foram identificados dois actores principais: o Cliente e o Administrador, que usam respectivamente um *Front Office* e um *Back Office*. Note-se que ambos são uma porta para uma zona mais reservada e que o Administrador já se encontra numa dessas zonas. Estas zonas de segurança são delimitadas por *firewalls*, que impedem o contacto directo com o Fedora e asseguram que a comunicação entre o Administrador e o *Back Office* seja realizada numa zona mais segura e controlada, diminuindo a possibilidade de ataques à integridade e confidencialidade do arquivo.

O Fedora possibilita que novos disseminadores de conteúdo sejam criados em tempo de execução e consequentemente novas formas de acesso aos mesmos conteúdos. Como o *Front Office* revela estes conteúdos por meio de disseminadores ao cliente, este terá de ter mecanismos para aceder a estes disseminadores e um meio de adicionar funcionalidade em tempo de execução para tratar estes disseminadores como necessário.

Para exemplificar este caso imaginemos que foi criado um novo disseminador para uma representação de um livro que oferece um método para obter uma imagem derivada de uma página do livro (cujo número é argumento do método). É claro que deve ser criada uma interface gráfica que usa este método para dar acesso ao conteúdo do livro ao cliente e que esta interface gráfica deve ser adicionada ao *Front Office* sem que haja necessidade de este ser recompilado.

Este é um caso clássico da *pattern Component Configurator* [SSRB00] que possibilita uma aplicação adicionar e remover os seus componentes em tempo de execução sem ter que modificar, recompilar ou estáticamente adicionar os componentes à aplicação.

Por falta de bases de desenvolvimento adequadas aos requisitos do projecto o *Front Office* e *Back Office* estão a ser desenvolvidos de raiz.

6 Conclusão

Os objectivos do RODA eram a construção de um protótipo. Para a construção desse protótipo especificaram-se requisitos funcionais e estruturais. À luz destes requisitos foi possível desenvolver estudos que levaram à especificação do modelo de dados e a uma série de decisões de implementação. Muitas destas decisões foram posteriormente validadas quer por experimentação da equipe do projecto quer por comparação com os resultados de equipes internacionais a trabalhar em projectos semelhantes.

Neste momento, a preservação digital é uma área que está a suscitar interesse na comunidade internacional. Naquilo que tenta fazer, o RODA é um projecto pioneiro: trabalha com normas que ainda estão a ser especificadas, algumas incompletas e tenta resolver tecnologicamente problemas que ainda não foram resolvidos na plataforma seleccionada (Fedora).

Algumas das ideias aqui discutidas foram já validadas com casos de estudo reais o que tem permitido à equipe avançar com mais segurança.

Podemos resumir o estado actual do RODA da seguinte forma: a sua especificação está feita, a solução tecnológica está planeada, o repositório central já está em exploração, alguns serviços (disseminadores) já estão operacionais, o *front-office* e o *back-office* finais estão em desenvolvimento, o ingestor de bases de dados está concluído (falta integrá-lo) e foi desenvolvido o primeiro protótipo de um construtor de SIP.

O que falta fazer pode resumir-se a: melhorar interfaces, criar serviços de administração e, mais importante que tudo o resto, criar as estruturas físicas e humanas para a implementação final do RODA (equipamento, recursos humanos e um plano para funcionamento em autonomia).

Uma coisa é certa, o RODA continuará a "rolar"por mais uns tempos...

Referências

[Bar06a] Francisco Barbedo. Especificação de requisitos. Technical Report 41012-005, IAN/TT, 2006.

[Bar06b] Francisco Barbedo. Taxionomias de objectos digitais a integrar no roda. Technical Report 41012-006, IAN/TT, 2006.

[ead98] Ead - encoded archival description. http://www.loc.gov/ead/, 1998.

[fSDS02] Consultative Committee for Space Data Systems. Washington: National Aeronautics and Space Administration, 2002.

[Gro05] PREMIS Working Group. Data dictionary for preservation metadata: final report of the premis working group. Technical Report Final report, OCLC Online Computer Library Center & Research Libraries Group, 2005.

[iNZ06] Project: Open Acess Repositories in New Zeland. Technical evaluation of selected open source repository solutions. Technical Report Version 1.3, Tertiary Education Commission of New Zeland, 2006.

[JLRH02] M. H. Jacinto, G. R. Librelotto, J. C. Ramalho, and P. R. Henriques. Bidirectional conversion between xml documents and relational data bases. In *7th International Conference on CSCW in Design*, Rio de Janeiro - Brasil, 2002.

[mar06] Marcxml. http://www.loc.gov/standards/marcxml, 2006.

[McD06] J. P. McDonough. Mets: standardized encoding for digital library objects. *International Journal on Digital Libraries*, 6(2), 2006.

[met06] Mets: An overview & tutorial. http://www.loc.gov/standards/mets/METSOverview.v2.html, 2006.

[nis06] Niso metadata for images in xml. http://www.loc.gov/standards/mix/, 2006.

[oA99] International Council on Archives. volume 0-9696035-6-8. International Council on Archives, 1999.

[Ruu02] Raivo Ruusalepp. Ahds preservation metadata framework. Technical report, Estonian Business Archives, Ltd, 2002.

[SSRB00] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture*, volume 2. John Wiley and Sons, 2000.

[TS04] K. P. Thomaz and A. J. Soares. A preservação digital e o modelo de referência open archival information system (oais). *DataGramaZero - Revista de Ciência da Informação*, 5, 2004.

[Wei97] S. Weibel. The dublin core: a simple content designation model for electronic resources. *Bulletin of the American Society for Information Science*, 24(1):9–11, 1997.

Alternativas ao XML: YAML e JSON

Rúben Fonseca Alberto Simões
{rubenfonseca,ambs}@di.uminho.pt

Departamento de Informática
Universidade do Minho

Resumo O XML tem sido eleito como a linguagem de anotação por excelência, possuindo ao mesmo tempo boas capacidades para serialização de estruturas computacionais e transporte de dados independente da plataforma. Recentemente porém, novos formatos de dados têm surgido. Alguns deles têm tido uma boa aceitação porque resolvem alguns problemas ou limitações do XML, sendo em algumas situações um bom complemento ou substituto do mesmo. Neste artigo iremos apresentar dois desses formatos de dados - o YAML e o JSON - fazendo uma abordagem geral dos mesmos e analisando algumas métricas que nos poderão ajudar a decidir se e quando usar estas alternativas.
Palavras-chave: XML, YAML, JSON, serialização, anotação

1 Introdução

Nos últimos anos, o XML tem sido adoptado em áreas extremamente diversas: da justiça às finanças, dos hospitais às telecomunicações, da agricultura ao jornalismo. O XML tem sido a sintaxe de escolha para novos formatos de documentos na maior parte das aplicações de computador. É usado nas *mainframes* na bolsa de valores de *Wall Street* para trocar acções. Crianças em casa guardam o estado dos seus jogos de PC em XML. Fãs do futebol recebem resultados em directo no seu telemóvel usando XML. O XML é simplesmente a sintaxe de documentos mais robusta, fiável e flexível alguma vez inventada.

Significa isto que não necessitamos de mais nenhum formato de dados no mundo? Será que a investigação e o trabalho nesta área já atingiu a maturidade e pouco ou nada se poderá inventar? Será que o XML é mesmo a melhor solução para *todos* os problemas nesta área?

Dezenas de alternativas ao XML existem no mercado[Dum06]. Algumas delas não têm a massa crítica necessária para se tornarem conhecidas. Podiam ser estudados formatos como as *S-expressions* ou o *Data::Dumper*. No entanto, de entre as várias alternativas, duas têm-se destacado nos anos mais recentes: o YAML e o JSON. Ao longo deste artigo faremos uma análise comparativa de ambas estas linguagens, tentando analisar de várias perspectivas se elas se apresentam como uma alternativa ao XML, ou se apenas têm um universo mais reduzido de aplicação.

Antes de começar, analisaremos alguns problemas ou limitações bem conhecidos do XML.

2 O XML não é perfeito

O XML é um standard definido e recomendado pela W3C para anotação de documentos. São inegáveis as vantagens que o XML trouxe a todo o mundo da representação e transferência de dados ou documentos. A sua flexibilidade, robustez e relativa simplicidade permitiu uma rápida adopção por parte de todo o tipo de entidades. O seu predecessor - o SGML - já estava em uso desde 1986, o que fez com que já existisse extensiva experiência técnica e aplicações disponíveis.

No entanto, como tudo no mundo, o XML também tem os seus pontos fracos. De seguida mencionaremos apenas os mais comuns apontados.

- A sintaxe é simultaneamente palavrosa e redundante.
  Isto pode afectar a leitura humana, a eficiência das aplicações e obriga normalmente a custos de armazenamento mais elevados. Este característica torna o XML difícil de aplicar em casos em que a largura de banda é limitada, embora a compressão possa reduzir o problema em alguns casos[LS00]. Este problema é particularmente grave para aplicações multimédia em telemóveis e PDAs que usem XML para descrever imagens e vídeo.

- A construção dos parsers não é tão trivial como parece.
  Embora teoricamente o XML permite a construção de parsers simples e consistentes, um bom parser de XML têm de ser desenhado para processar dados arbitrariamente aninhados e efectuar testes adicionais para detectar dados mal formatados ou erros de sintaxe (devido, em parte, ao problema descrito no item anterior). Isto pode causar carga adicional significativa para os usos mais básicos do XML, particularmente nos sistemas embebidos[vE04]. Além disso, quando os recursos são limitados, problemas de segurança podem surgir se o XML provem de fontes não confiáveis, podendo levar à exaustão de recursos ou a *stack overflows* (e.g. XML composto por um número muitíssimo elevado de `<elemento>` aninhado).

- Os requisitos básicos de processamento não suportam um grande conjunto de tipos de dados.
  Por esta razão, a interpretação semântica envolve trabalho adicional para inferir os tipos de dados num documento. Por exemplo, não existe nenhum mecanismo em XML, para que `3.14159` seja um número de vírgula flutuante em vez de uma palavra de sete caracteres. As linguagens de *XML Schema* adicionam esta funcionalidade[CEM03].

- A sintaxe do XML é difícil de usar para humanos.
  A sequência de teclas que se usam para escrever expressões XML num teclado convencional de computador são particularmente difíceis de usar, dificultando a criação rápida de documentos anotados com XML. O uso de geradores ou de editores especializados para o efeito poderão diminuir o peso deste problema.

Quer se concorde com estes pontos fracos do XML ou não, será bom manter uma mente aberta e ver se existem alternativas que resolvam estes problemas ou que apenas complementem o XML em contextos mais específicos. Neste caso específico, até que ponto o YAML ou o JSON são bons candidatos a resolver estes problemas? Será que conseguem manter o mesmo nível de robustez e flexibilidade do XML? Como é o desempenho relativo dos parsers existentes? Valerá a pena tentar?

3 YAML

Talvez a primeira questão que surja é "porquê o nome YAML?". Existem um bom número de ferramentas que adoptaram acrónimos na forma `YA*`, significando "Yet Another XXX". Por isso dois nomes - aparentemente contraditórios - têm sido atribuídos para o acrónimo YAML: "Yet Another Markup Language" e o mais recursivo "YAML Ain't a Markup Language". No entanto, existe alguma lógica para isto: o YAML oferece grande parte do que as linguagens de anotação oferecem, mas com uma notação muito leve.

O YAML é um formato de serialização de dados legível por humanos, que se inspirou em conceitos de linguagens como o XML, C, Python, Perl e também o formato do correio electrónico especificado no RFC 2822. Foi proposto por Clark Evans em 2001 e desenhado em conjunto com Ingy döt Net e Oren Ben-Kiki[OBK05].

Embora não seja menos genérico que o XML, o YAML é em grande parte mais simples de ler, editar, modificar e produzir que o XML. Ou seja, quase tudo o que é possível de representar em XML pode ser representado em YAML, e ao mesmo tempo, de uma forma mais compacta.

O YAML foi definido para suportar apenas caracteres no sistema UTF8 ou UTF16, sendo o comportamento dos parsers indefinido quando a stream YAML está em qualquer outra codificação.

Os espaços de nomes são um exemplo de algo que a especificação corrente do YAML não suporta mas que podem ser mais ou menos adaptados em cima do mesmo.

3.1 Exemplo

O YAML foi criado com a crença de que todos os dados podem ser adequadamente representados por combinações de listas, mapas e registos. Uma lista é uma colecção ordenada de zero ou mais valores; um mapa é uma coleção de zero ou mais pares nome/valor; um registo é um tipo primário como uma string.

A sintaxe é relativamente simples e foi desenhada tendo em mente a legibilidade pelos humanos, mas ao mesmo tempo que permitia que os dados sejam facilmente transformados nas estruturas mais comuns das linguagens de alto nível.

Desta maneira, o YAML incide mais na representação limpa e compacta das estruturas de dados que se encontram em linguagens como Perl, Python, Ruby,

e com menos relevância, o Java. Existem bibliotecas YAML para quase todas as linguagens de programação.

Vejamos um exemplo concreto. Imagine-se o seguinte documento XML com a descrição de um clube de xadrez.

```
<?xml version="1.0"?>
<club>
  <players>
    <player id="kramnik"
            name="Vladimir Kramnik"
            rating="2700"
            status="GM" />
    <player id="fritz"
            name="Deep Fritz"
            rating="2700"
            status="Computer" />
    <player id="mertz"
            name="David Mertz"
            rating="1400"
            status="Amateur" />
  </players>
  <matches>
    <match>
        <Date>2002-10-04</Date>
        <White refid="fritz" />
        <Black refid="kramnik" />
        <Result>Draw</Result>
    </match>
    <match>
        <Date>2002-10-06</Date>
        <White refid="kramnik" />
        <Black refid="fritz" />
        <Result>White</Result>
    </match>
  </matches>
</club>
```

A representação acima é bastante clara. No entanto, sofre de alguns dos problemas mencionados na secção anterior quanto à verbosidade, fraca legibilidade e dificuldade na elaboração deste documento. Compare-se agora estes mesmos dados com a versão em YAML.

```
players:
  Vladimir Kramnik: &kramnik
    rating: 2700
    status: GM
  Deep Fritz: &fritz
    rating: 2700
    status: Computer
```

```
8     David Mertz: &mertz
9       rating: 1400
10      status: Amateur

11  matches:
12    -
13      Date: 2002-10-04
14      White: *fritz
15      Black: *kramnik
16      Result: Draw
17    -
18      Date: 2002-10-06
19      White: *kramnik
20      Black: *fritz
21      Result: White
```

Há algumas coisas interessantes acerca deste formato. O site do YAML[OBK05] oferece a especificação exacta, mas este simples exemplo fornece uma ideia dos elementos básicos. A especificação também inclui maneiras intuitivas de incluir strings contendo parágrafos múltiplos. A necessidade de aspas é mínima, e os tipos de dados são inferidos a partir de padrões (por exemplo, se os dados se parecem com uma data são tratados como tal, a não ser que sejam protegidos com aspas explicitamente).

Pode-se usar referências para todos os nomes (notar o operador `&` e `*`). Na realidade, todas as entidades podem ser referenciadas em qualquer parte do texto. Com isto podemos obter e usar estruturas partilhadas para serializar a informação de maneira mais lógica e eficiente do que no XML. O YAML mantém a distinção entre colecções ordenadas e associadas. Como bónus final, qualquer um pode editar YAML num editor de texto convencional.

Uma das grandes vantagens do uso de YAML não está relacionado com o seu poder sintáctico. As interfaces uniformes em todas as linguagens suportadas são um grande trunfo para o YAML. Por exemplo, para ler e poder manipular os dados do exemplo anterior em Perl seria tão simples como:

```
1  use YAML ();
2  my $club = YAML::LoadFile('club.yml');
3  my $club_yamlstr = YAML::Dump($club);
```

Mas como se comportará o YAML para realizar tarefas de anotação de documentos?

3.2 Anotação de documentos

O YAML não foi construído com a anotação de documentos em mente. Embora devido à sua estrutura simples e flexível seja possível realizar algum trabalho nesta área, o YAML não se aproxima nem de perto ao poder que o XML

tem para anotar documentos. Talvez um exemplo ajude a entender este ponto. Considere-se o seguinte excerto XML:

```
<paragraph>
    O artigo <artigo id="1253">XML é fixe</artigo> é bom.
</paragraph>
```

A mistura de elementos com o texto é algo que dificilmente é conseguida no YAML. Este facto dificulta imenso o uso do YAML nesta área - na realidade nunca foi o seu objectivo. No entanto, o YAML trás uma vantagem: por definição não é possível escrever um documento mal formado por misturar *overlapping tags* (e.g. `<b><p></b></p>`).

Pelas razões apresentadas, o YAML não é um substituto, nem sequer um concorrente com o XML para anotação de documentos.

3.3 Serialização de dados

É nesta área que o YAML mais brilha. As suas características permitem facilmente representar os tipos de dados mais comuns das grandes linguagens de programação. A própria especificação YAML defende que todos os tipos de dados se podem representar à custa de combinações de listas, mapas e registos[OBK05].

Ao possuir uma sintaxe limpa, torna a serialização de dados mais legível à inspeção humana. No entanto, sem perder a generalidade, o YAML apresenta maneiras alternativas de expressar o mesmo tipo de dados. Por exemplo:

```
--- # Filmes favoritos, formato bloco
- Casablanca
- Spellbound
- Notorious
--- # Lista de compras, formato inline
[milk, bread, eggs]
```

```
--- # Bloco
name: John Smith
age: 33
--- # Inline
{name: John Smith, age: 33}
```

Um outro trunfo do YAML é que, as representações suportadas (listas, mapas e registos) correspondem a tipos nativos em todas as linguagens dinâmicas. Isto permite não gastar muito tempo a transformar uma stream YAML numa estrutura de dados a ser consumida pela aplicação. Os programadores destas linguagens sentem-se imediatamente confortáveis tanto na importação de dados em YAML, como na sua exportação ou criação.

Talvez por este facto, o YAML foi escolhido como formato de serialização por omissão na linguagem de scripting Ruby. É também utilizado cada vez mais para

escrever ficheiros de configuração, já que permite, com uma sintaxe limpa, simples e intuitiva, escrever estruturas de configuração arbitrariamente complexas que podem ser carregadas para qualquer linguagem de uma forma completamente uniforme.

3.4 Análise de desempenho do parser

Para ter uma ideia de como o YAML se comporta face ao XML efectuou-se um teste comparativo de desempenho. As condições foram as seguintes:

- Ficheiros de gerados automáticamente, com n elementos de texto aleatório
- Parser de XML: libxml2[1] (2.6.26) + XML::LibXML[2] (1.62)
- Parser de YAML: libsyck[3] (0.55) + YAML::Syck[4] (0.72)
- Computador pessoal Intel Core Duo T2300 1.66Ghz, 1GB RAM
- Linux 2.6.18.3

Nenhum dos parsers usados tira partido do multi processamento disponível na plataforma.

Os elementos aleatórios têm a seguinte forma.

```
<contacts>
    <person>
        <number>855904</number>
        <name>sdfasdfasdfasdfasdffasdfa</name>
        <place>aw da dk dfw  awfhkda hlhwhdfah jadfkw hw</place>
        <date>51 53 9 30 10 106 4 333 0</date>
    </person>
    (...)
</contacts>
```

Depois de gerar ficheiros de vários tamanhos, estes foram submetidos aos respectivos parsers (escritos em Perl, usando as bibliotecas C), e os tempos de parsing foram anotados. Os resultados obtidos estão nas figuras 1 e 2.

Como se pode observar, o YAML é ligeiramente mais eficiente quando o input é mais pequeno. No entanto não escala convenientemente com o tamanho do input, ou pelo menos não tão eficientemente como o parser de XML escala. Podemos concluir que o YAML é mais apropriado para pequenas e médias serializações. Isto explica porque é que o YAML é usado mais frequentemente para serialização de objectos e formato de ficheiros de configuração, onde o tamanho do ficheiro não é significativo para a perda de desempenho.

O parser de YAML é mais eficiente ao nível da memória, necessitando de menos memória para processar a mesma entrada. O ponto de comutação acontece

[1] `http://xmlsoft.org/`
[2] `http://search.cpan.org/~pajas/XML-LibXML-1.62001/`
[3] `http://yaml.kwiki.org/index.cgi?LibSyck`
[4] `http://search.cpan.org/~audreyt/YAML-Syck-0.72/`

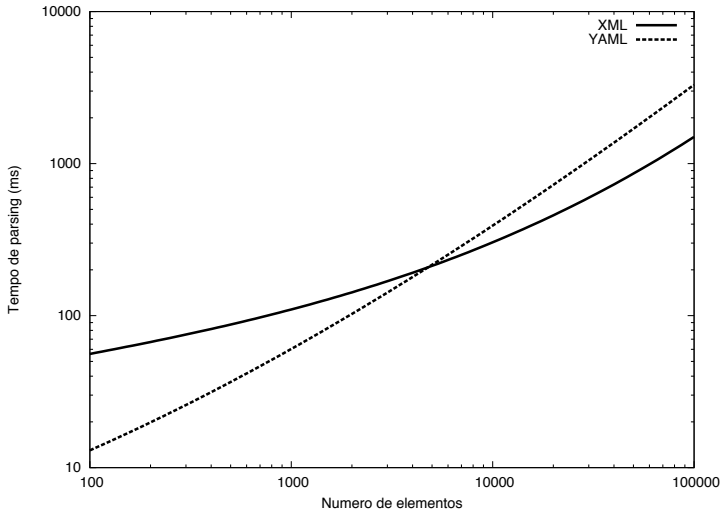

Figura 1. Tempo de execução em função do número de elementos

neste caso quando a entrada atinge o 1Mb. A partir deste ponto, o parser de XML libxml2 começa a ser mais eficiente no tempo de parsing, gastando no entando sempre mais memória do que o parser de YAML syck.

No entanto, estes resultados deverão ter em conta que o parser de XML libxml2 é um parser resultante de muitos anos de trabalho de uma comunidade numerosa, e por isso produz resultados supreendentemente eficientes quando comparados com outros parsers conhecidos, tanto ao nível de tempo de parsing como no consumo de memória[Chi04]. Por outro lado, o parser Syck é novo e trabalho de poucas pessoas, com muita margem para evoluir em termos de funcionalidades e desempenho.

4 JSON

O JSON (pronunciado como a palavra inglês *Jason*) é um acrónimo para "JavaScript Object Notation", e é um formato de texto para a serialização de dados estruturados. É derivado dos *object literals* do JavaScript, conforme definido no standard ECMAScript[Int99].

O JSON pode representar quatro tipos primários (strings, números, booleanos e nulos) e dois tipos estruturados (objectos e vectores). Um objecto é uma colecção não ordenada de zero ou mais pares nome/valor, onde o nome é uma string e o valor é uma string, número, booleano, nulo, objecto ou vector. Um vector é uma sequência ordenada de zero ou mais valores[Cro06].

Assim o JSON foi desenhado com o objectivo de ser simples, portável, textual, e um subconjunto do JavaScript. Iremos ver mais à frente o que isto pode significar no contexto da aplicação em serialização de dados na Web.

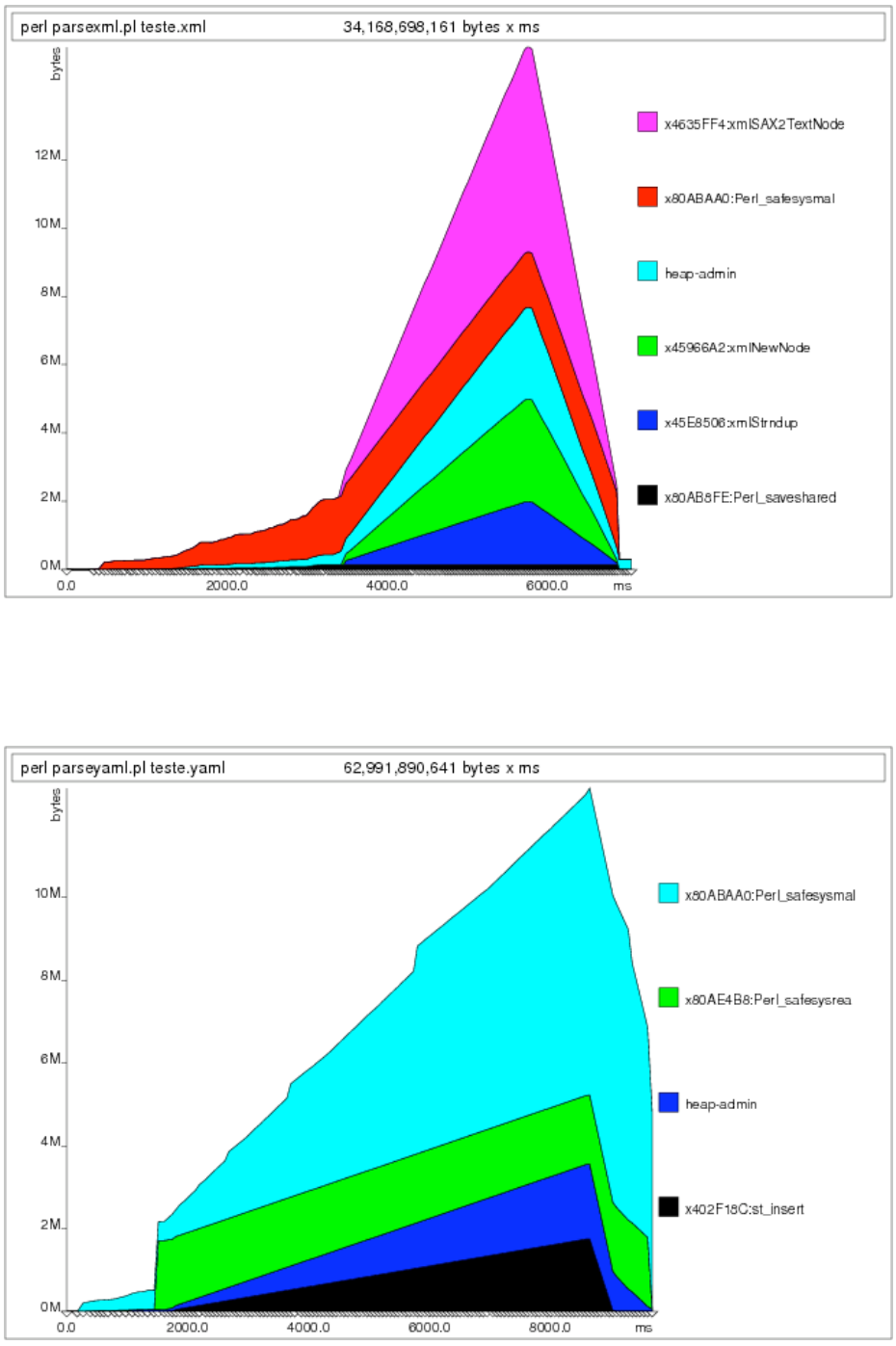

Figura 2. Consumo de memória do parser de XML e YAML para entradas de 10000 elementos

Assim como o YAML, o JSON apenas suporta texto em codificação UTF8 ou UTF16.

4.1 Exemplo

As similaridades de sintaxe entre o YAML e o JSON são mais do que à partida se pode imaginar[wtls05]. Atente-se ao seguinte exemplo:

```
{
    "type": "menu",
    "value": "File",
    "items": [
        {"value": "New", "action": "CreateNewDoc"},
        {"value": "Open", "action": "OpenDoc"},
        {"value": "Close", "action": "CloseDoc"}
    ]
}
```

Apesar de esta representação aparentar isso, a indentação e o uso de espaços em branco é totalmente opcional no JSON. O mesmo exemplo poderia ser escrito apenas num linha sem que isso resultasse em erro no compilador.

Um olhar atento poderá concluir correctamente que o JSON é praticamente um subconjunto funcional do YAML. Na realidade a maior parte dos parsers de YAML conseguem lidar com grande parte do JSON, já que este último partilha grande parte das características do YAML em modo *inline*. Este facto é mera coincidência e não foi decisão aquando do desenho do formato. O YAML e o JSON foram concebidos isoladamente e por equipas diferentes.

Devido à sua simplicidade, o JSON não suporta referências para elementos, e obriga ao uso de aspas para se manter compatível com a sintaxe do JavaScript.

4.2 Anotação de documentos

Sendo um subconjunto funcional do YAML, o JSON apresenta praticamente o mesmo poder expressivo e capacidade de anotação (quase nula) do YAML. Também este formato não foi desenhado com a anotação de documentos em mente, e por isso, normalmente não substitui o XML para a realização deste tipo de tarefas.

4.3 Transporte de dados

Como subconjunto do YAML, também aqui o JSON brilha como formato de serialização. Além de possuir a maior parte das vantagens do YAML nesta área, o JSON possui um trunfo fortíssimo que fez com que fosse aceite rapidamente pela comunidade e usado extensivamente: o JSON é um subconjunto do "JavaScript Object Notation". O que isto significa? Considere-se o seguinte código JavaScript no contexto de uma aplicação web que acaba de fazer uma invocação remota.

```
1 the_object = eval("(" + http_request.responseText + ")");
```

No contexto de aplicações que usem invocações remotas na Web (e já que o AJAX[Gar05] se considera indispensável para a construção de um novo site bem sucedido) o JSON é um recurso extremamente útil, visto que permite transportar um objecto serializado arbitrariamente complexo, e transformá-lo num objecto JavaScript com um simples *eval*. Temos assim um parser poderosíssimo de JSON incluido em todos os grandes browsers de internet existentes do mercado.

Este facto faz com que o JSON esteja cada vez mais a ser usado para transporte de dados serializados no contexto da internet, em detrimento do XML. Embora todos os browsers mais usados suportem correctamente o o processamento de XML, ainda nem todos possuem um bom e completo processador e interrogador de XML, incorporado numa linguagem dinâmica embutida, o que faz com que o seu uso aumente a complexidade das aplicações Web que dele fazem uso. O JSON é assim, neste contexto, um óptimo substituto do XML.

Mas como se comporta o JSON em termos de desempenho?

4.4 Análise de desempenho

Realizamos também um teste de desempenho ao JSON, exactamente nas mesmas condições que o teste da secção 3.4. Para tal foi usado:

– Parser de JSON: libsyck (0.55) + JSON::Syck[5] (0.72)

O leitor atento já percebeu que o parser de JSON é o mesmo que foi usado o parsing de YAML. Isto deverá reforçar a ideia passada na secção 4.1 onde concluímos que o JSON e o YAML partilham (por coincidência) grande parte da sintaxe. Logo o parser Syck consegue atacar ambas as linguagens de uma maneira uniforme. Os resultados no entanto são interessantes, conforme pode ser observado na figuras 3 e 4.

Como se pode verificar, também o JSON não escala tão bem como o XML quando o tamanho do input aumenta significativamente. No entanto, o JSON escala melhor do que o parser de YAML. A explicação poderá vir do facto da especificação do JSON ser menos complexa, sendo assim um formato menos flexível, mas mais simples e fácil de processar. O consumo de memória durante o parsing é idêntico ao do YAML (e melhor que o de XML), talvez resultado do uso da mesma biblioteca para o parser. No contexto de aplicações Web (onde o JSON é mais usado), podemos concluir que o JSON é uma boa aposta pois além das suas vantagens funcionais, é ligeiramente mais rápido que o XML para inputs de pequena ou média dimensão (no nosso teste, até 1Mb de entrada).

[5] `http://search.cpan.org/~audreyt/YAML-Syck-0.72/`

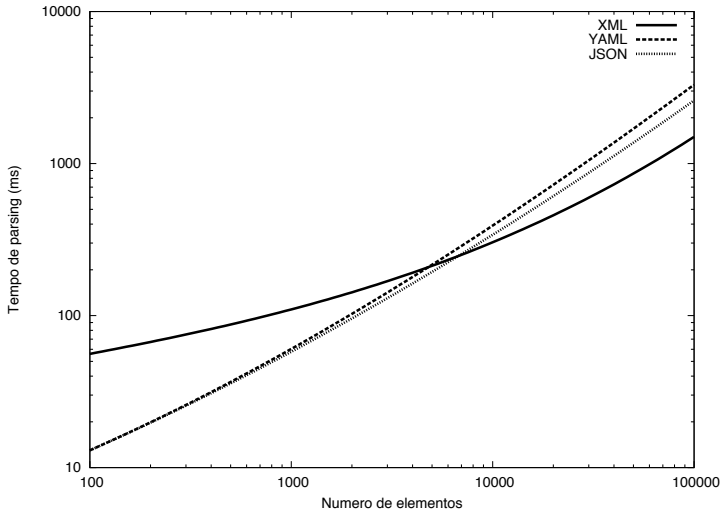

Figura 3. Tempo de execução em função do número de elementos

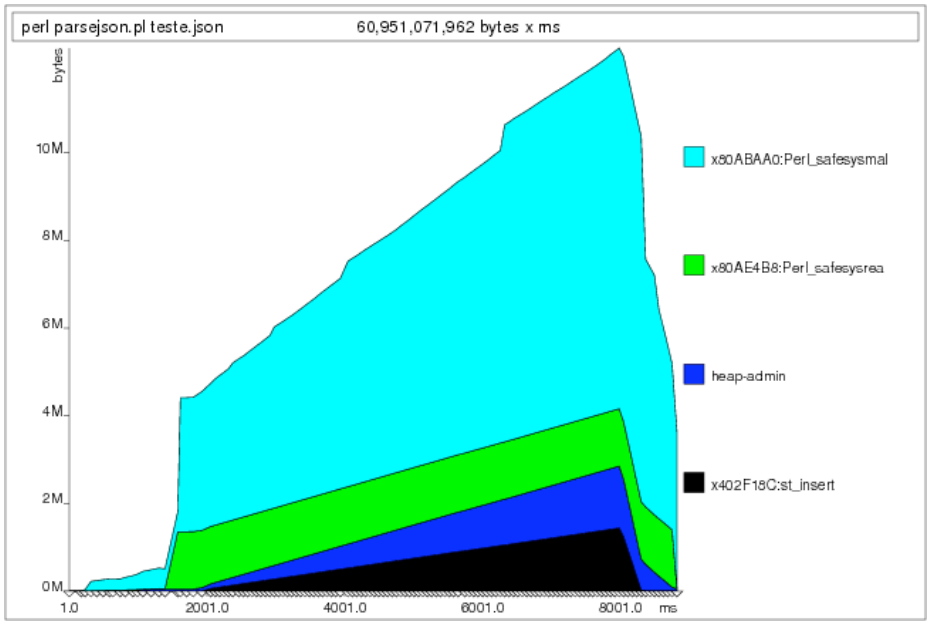

Figura 4. Memória consumida pelo parser de JSON para input de 10000 elementos

5 Conclusão

Embora o XML conte com uma enorme comunidade de utilizadores, não é perfeito nem a única alternativa para todos os problemas de anotação e serialização. Alguns desses problemas foram estudados e atacados por outros formatos como o YAML e o JSON.

Apesar de não competirem com o XML como ferramenta para anotação de documentos, tanto o YAML como o JSON demonstram boas capacidades no que toca à serialização de dados. O YAML apresenta uma sintaxe limpa, fácil de ler e editar, e muito bem estruturada. O JSON, sem grande perda de generalidade, coloca um pouco mais de açúcar na sua sintaxe de modo a torná-lo verdadeiramente um subconjunto do JavaScript.

Devido às suas características, tanto o YAML como o JSON permitem usar parsers simples, e serem transformados transparentemente para os tipos de dados nativos das grandes linguagens de scripting. Ao mesmo tempo são uniformes (i.e. usam-se sempre da mesma maneira nas diferentes linguagens suportadas). O JSON possui adicionalmente um forte trunfo: existem bons parsers (e fáceis de invocar), completamente funcionais em todos os grandes browsers em utilização actual, o que permite o uso eficiente de serializações para na construção de aplicações Web interactivas.

A especificação do YAML é mais complexa do que apresentado. O YAML suporta ainda vários documentos no mesmo ficheiro, atalhos, tipos de dados forçados, blocos e aliases. O YAML Cookbook é uma boa fonte para aprender mais detalhes sobre estes[OBK06].

É certo que não são substitutos, mas algumas das tecnologias que assentam sobre XML podem ser mais facilmente implementadas com YAML ou JSON. É necessário XML para RPC? Será necessário XML para RDF? É necessário usar XML para tudo? Não. É o YAML ou o JSON um substituto perfeito? Não, mas em algumas circunstâncias poderá ser mais útil usá-los em detrimento do XML. Todos eles são excelentes standards, e todos têm o seu lugar.

Referências

[CEM03] Charles E. Campbell, Andrew Eisenberg, and Jim Melton. Xml schema. *SIGMOD Rec.*, 32(2):96–101, 2003.

[Chi04] Suren A. Chilingaryan. Xml benchmark. `http://xmlbench.sourceforge.net/`, 2004. Provides benchmarking toolset for all available multiplatform C/C++ (and some Java) XML parsers.

[Cro06] D. Crockford. Json specification. `http://www.ietf.org/rfc/rfc4627.txt`, 2006.

[Dum06] Edd Dumbill. Exploring alternative syntaxes for xml. `http://www-128.ibm.com/developerworks/xml/library/x-syntax.html`, 2006. Artigo sobre alternativas ao XML.

[Gar05] Jesse James Garrett. Ajax: A new approach to web applications, 2005. `http://adativepath.com/publications/essays/archives/000385.php`.

[Int99] Ecma International. Standard ecma-262. `http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf`, 1999. ECMAScript Language Specification.

[LS00] Hartmut Liefke and Dan Suciu. Xmill: an efficient compressor for xml data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM Press.

[OBK05] Brian Ingerson Oren Ben-Kiki, Clark Evans. Yaml specification. `http://yaml.org/spec/`, 2005.

[OBK06] Brian Ingerson Oren Ben-Kiki, Clark Evans. Yaml cookbook. `http://yaml4r.sourceforge.net/cookbook/`, 2006. Conjunto de exemplos do uso extensivo de YAML.

[vE04] Robert van Engelen. Code generation techniques for developing light-weight xml web services for embedded devices. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 854–861, New York, NY, USA, 2004. ACM Press.

[wtls05] why the lucky stiff. Yaml is json. `http://redhanded.hobix.com/inspect/yamlIsJson.html`, 2005. Artigo que compara as semelhanças entre YAML e JSON.

Data Model for Geographic Ontologies Generation

Marcirio Silveira Chaves, Catarina Rodrigues and Mário J. Silva

Department of Informatics
Faculty of Sciences - University of Lisbon
1749-016 Lisbon, Portugal
{mchaves,crodrigues,mjs}@xldb.fc.ul.pt

Abstract: Structured geographic information has an important role in many applications, such as geographically-aware search engines and named-entity (e.g., proper names) recognizers. This information is compiled from resources like dictionaries, gazetteers and ontologies. The quality of the applications depends critically on the richness of these resources. This article describes a new version of the Geographic Knowledge Base (GKB), an environment for integrating geographic data and generating ontologies. We introduce the new metamodel of GKB and show how the information in its repository is transformed into OWL representations. We also present the data integration process of the administrative and physical domains, which deals with geographic relationships in both domains.

Keywords: modeling, geographic ontologies, ontology generation

1 Introduction

A well-defined model has an important role in any kind of system, facilitating its reuse and extension. The model-driven approach defines relationships among concepts in a domain and precisely specifies the key semantics and constraints associated with these domain concepts [1]. Considering that all information represented in a system is constrained to the model, the modeling phase is crucial to determine what and how is possible to represent within an application.

The modeling becomes more complex when we try to bring together several views of different communities. The geographic knowledge domain is an example in which geographers, geographic engineers and computer scientists (when coping with geographic information) use different terminologies to represent the same information. These terminologies need to be integrated in a model, which must be able to consistently represent the geographic concepts. A consensual notion concerns the existence of two geographic domains: administrative and physical.

Geographic Knowledge Base (GKB), an environment for integrating geographic data and generating ontologies, is implemented based on a model able to support the storage of the information from administrative and physical domains [2,3]. The first version of GKB (GKB 1.0, henceforth) contains two instances: one

stores data about the geo-administrative and network (web) domains of Portugal and the other contains data from all over the world.

The information stored in GKB 1.0 is exported to ontologies in the OWL format. One of these ontologies (Geo-Net-PT01) contains the integrated geographic administrative information of Portugal and is freely available from `http://xldb.fc.ul.pt/geonetpt`. Applications using Semantic Web technologies can easily work with the ontologies exported from GKB [4,5]. Other examples of use of the ontologies generated by GKB 1.0 can be found in [6,7].

This article describes a new version of GKB (hereafter GKB 2.0), which supports a couple of new requirements. A more detailed representation of the geographic names and a better control of information sources among other improvements should be provided by GKB 2.0. In addition, full support to the modeling of the physical domain and its integration with the administrative domain should also be present in the next version of GKB.

The main contribution of this article is a detailed description of the model for integrating geographic information from multiple domains and making this knowledge available as ontologies in a Semantic Web format. We are working on data about Portugal, but the model could be applied to represent geographic knowledge from any country or geographic region.

The structure of this article is as follows: Section 2 introduces the GKB requirements and Section 3 details the design of the new GKB model. Section 4 describes the geo-physical domain introducing new physical concepts and relationships. Section 5 shows the conversion of the GKB 2.0 model to OWL ontologies and Section 6 presents related work. Section 7 gives the final remarks.

2 GKB 2.0 Requirements

It is necessary to introduce the notion of the *feature* concept in this work, borrowed from the GIS (Geographic Information System) world, before we describe the requirements for GKB 2.0. A *feature* is "a meaningful object in the selected domain of discourse" as defined in ISO 19109 [8]. Examples of *features* include the `Douro river` and the `Álvares Cabral Avenue`. Feature types, in the above examples are `river` and `avenue`, and their names are `Douro` and `Álvares Cabral`, respectively.

GKB 2.0 is centered on the notion of feature as in GKB 1.0, but includes a number of extensions:

Support for relationships between types: GKB 1.0 only supported relationships between features, such as `Lisbon` is `part-of Portugal`. GKB 2.0 also supports relationships between feature types, such as `rivers` are `part-of continents`.

Support for generic property sets: All the major classes (feature, feature type and name) may now include arbitrary attributes, whose support is defined in the metamodel. For example, attributes to a river include `source`, `outlet` and `length`, while a soil has a `PH level` and a mountain has an

`altitude`. GKB 2.0 also provides a more detailed specification of feature names. Attributes of geographic names, like the language in which a name is given can be captured. In addition, it may store other attributes of the geographic names, such as time and demonyms. For example, `Olissipo` is an `historic` name of `Lisbon` and `lisboeta` one of the Portuguese `demonyms` of the inhabitants of `Lisbon`.

Better control of information sources: Data lineage was very simplistic and recorded only at the feature level in GKB 1.0. For example, all information related to the *Municipality of Lisbon* was associated to one information source. In GKB 2.0, each name, type and relationship can be independently associated to a distinct information source. This extension allows us to know, for example, that two types are provided by distinct information sources and the relationship between them is derived from a third information source.

3 GKB Metamodel

A model is an abstraction of phenomena in the real world, and a metamodel is yet another abstraction, highlighting properties of the model itself [9]. In GKB the metamodel describes the classes and relationships that are used to model geographic information. The GKB 2.0 metamodel supports representations of multiple information domains related to geography, such as the administrative and physical domains. In each of these domains, the information is organized using concepts from the metamodel (e.g. features, feature types and their relationships).

3.1 Base Model

The core of the GKB 2.0 metamodel is represented in Figure 1. The class *Feature* is associated with the class *Type*, which stores the types (feature `Douro` is of type `river`). The class *Type-Relationship* captures the relationships between types (a `Municipality` is `part-of` a `Country`).

The class *Relationship-Type* stores the supported relationships, such as `part-of` and other relationships of geographic nature. Considering that a `river` is `part-of` a `continent`, the class *Type-Relationship* stores this relationship.

Features may be specialized by *Feature-Footprints*, which capture coordinate data. Coordinates represent centroids, bounding boxes and polygons. The feature `Serra da Estrela` has a centroid at `40°20N, 7°38W`.

3.2 Attributes and Names Representation

Figure 2 represents the base model of Figure 1 extended with the classes used for representing names and the attributes of types, features and names. Different geographic types have different attributes. A river has a length, a source and a mouth, while a mountain has an altitude and a municipality has population.

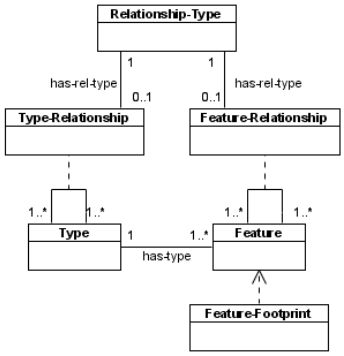

Figure 1. GKB 2.0 base class metamodel.

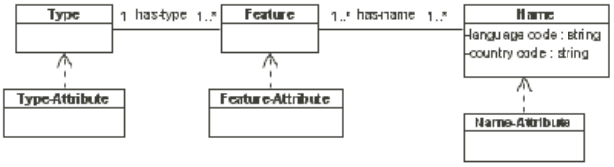

Figure 2. GKB 2.0 names and attributes model.

The classes *Type-Attribute* and *Feature-Attribute* add property sets to the base classes *Type* and *Feature*, respectively.

GKB 2.0 also provides better modeling of the geographic names. Each name is associated with a language and its country code (e.g., PT-BR), which are captured in the class *Name*. The language coding adopted follows the language tag standard <language code"–"country code> defined by RFC 3066 of the IETF [10]. Names can also be extended with sets of attributes capturing preferences (e.g., a name is preferred or alternative), time (e.g., used in the nineteenth century), use (e.g., rare, colloquial) and demonyms. These are stored in the class *Name-Attribute*.

For example, the river `Tajo` has its equivalent in Portuguese `Tejo` and its historic name `Tagus`, from Latin. The attribute historic is captured in the class *Name-Attribute*, whereas the class *Name* stores the names (`Tajo`, `Tejo` and `Tagus`), its languages (ES, PT and LA) and its country codes (ES and PT), respectively.

3.3 Inter-domain Relationships

GKB 2.0 stores information on administrative and physical domains. In addition to modeling relationships between features in the same domain, it supports inter-domain relationships. Figure 3 shows the model representing the inter-domain relationships.

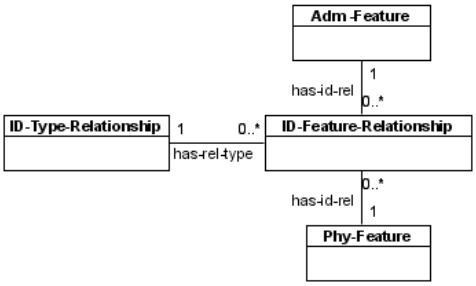

Figure 3. Inter-domain relationships in GKB 2.0.

The class *Adm-Feature* contains information from the administrative domain and the class *Phy-Feature* information from the physical domain. The class *ID-Type-Relationship* (ID stands for inter-domain) stores relationships such as `part-of` and `adjacency`. For example, the `municipalities of Lisboa` and `Setúbal` (administrative domain) are `adjacent` to the `river Tejo` (physical domain). This relationship is stored in the class *ID-Feature-Relationship*.

Other relationships such as `crosses`, `touches` and `intersects` are implicit on the footprint data and are not modeled as *ID-Feature-Relationship* in GKB. For example, the `Douro river crosses` the `municipality of Porto` and `intersects` the `Biótopo Alto Douro Internacional`.

3.4 Data Provenance

In GKB 2.0 the information sources are individually assigned both to types (concepts) and names (instances). Figure 4 shows how instances of the classes type, name and relationship are associated to their information source.

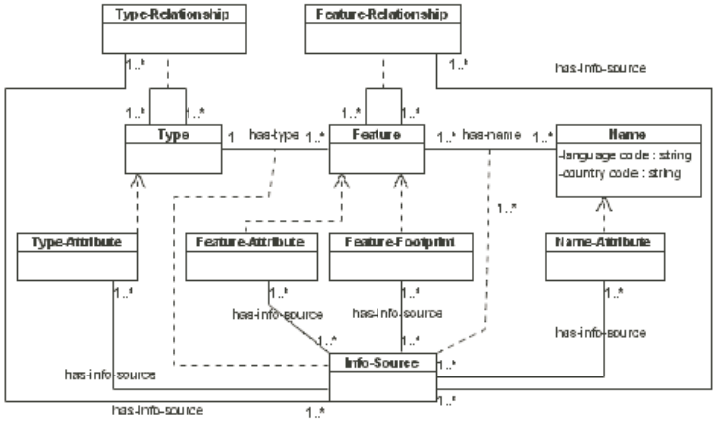

Figure 4. GKB 2.0 information sources model.

Information sources (IS) are now independently associated to each individual attribute and relationship in the model, and to each association between a feature and its names and type.

This approach also allows applications using GKB 2.0 to reason on the credibility of information assigning weights to each piece of data based on the level of authority of the associated information source. For example, a GIS should probably work just with the knowledge associated to coordinates provided by a state authority.

3.4.1 Information Sources

The information being used in this work for the Ontology of Portugal comes from *Ministério do Ambiente* (MA - Ministry of the Environment), *Instituto Geográfico do Exército* (IGeoE - Geographic Institute of the Army), *Instituto Geográfico Português* (IGP - Portuguese Geographic Institute), *Instituto da Água* (IA - Institute of the Water), *Instituto Nacioanal de Estatística* (INE - National Institute of Statistics), *Correios e Telégrafos de Portugal* (CTT - Portuguese Post Office) and *Instituto de Pesquisa da Marinha* (IMAR - Institute of Marine Research).

Information about the uses and occupations of the ground should come from the *Ministério da Economia*, but it is only available in a descriptive form at the moment. Such information will allow us geocoding to municipalities and through extrapolation represent their occupations of the ground. Other possible inferences concern to the ground and population or uses of the ground and climate.

Table 1 presents the geo-physical concepts provided by the information sources and the corresponding number of features (more than 23,000 features). Most of the information in Table 1 comes from IGP, which is the centre of reference to deal with geo-physical information in Portugal. All of these features have coordinate information. These coordinates represent polygons, which define the area occupied by each concept. The datum used in GKB 2.0 will be the European Terrestrial Reference System 1989 (ETRS89) - (`euref.org`). Statistics on data of administrative domain integrated in GKB have been published elsewhere [2].

4 Model of the Physical Domain

The previous sections described the GKB 2.0 metamodel for representing geographic information from multiple domains. This section presents the conceptual representation of the geo-physical domain and its integration with the geo-administrative domain. This last domain was previously presented in detail [2].

The geo-physical domain is harder to model than the administrative domain, because it has a much richer set of associations. This also makes the modeling of inter-domain relationships involving the physical domain more complex.

We divide the geo-physical domain into six sub-domains: hydrography, relief, soil uses and occupations, communication pathways, soils and climate. Figure 5

Table 1. Number of geo-physical features of each type obtained for the geographic ontology of Portugal.

Types	# features	Types	# features	Types	# features
Adega Reconhecida	311	Estabelecimento Dormida	2847	Nascente	220
Albufeira (área)	80	Estação Arqueológica	456	Parque Campismo	150
Albufeira (ponto)	1800	Estância Termal	35	Parque Nacional	55
Aldeia Preservada	217	Ferrovias	103	Parque Natural	12
Aquífero	125	Fortificação	263	Património Mundial	11
Área Paisagem Protegida	18	Gruta	7	Paul	441
Area Protegida	49	Jardim Botânico	7	Praia	591
Bacia de Escoamento	209	Kartódromo	30	RAMSAR	10
Bacia Hidrográfica	129	Lagoa	10	Rede Hidrográfica	7589
Biótopo	120	Litologia	2245	Regadio	90
Campo Golf	65	Local Culto	23	Região Natural	437
Casino	8	Marina	26	Reserva Natural	9
Edifício Notável	3877	Monumento Natural	5	Zona Fito geográfica	278
Espécie Notável	2432	Museu	524		

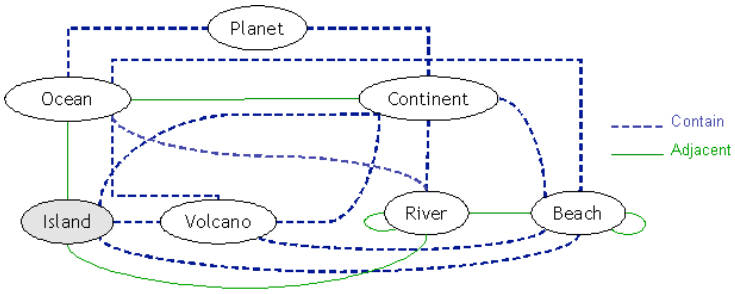

Figure 5. Excerpt of the geo-physical conceptual representation in GKB 2.0.

presents a small excerpt of the geo-physical representation, restricted to concepts from the hydrography and soil sub-domains.

The main problem in the representation of the geo-physical domain is how to correctly model the large number of relationships among the concepts. An example of the complexity of the relationships arise for the `island` concept, which is related to `oceans` and `rivers` under the `adjacency` relationship. However, its relationships are more complex when dealing with the *part-of* relationship. In the example of Figure 5, an `island` may be `part-of` a `continent`, a `beach` or a `volcano`. An `island` may also be related to other physical concepts, which have been omitted for the sake of clarity and space.

Figure 6 shows some instances of the concepts in Figure 5. It is possible to observe the multiplicity of relationships between features. It describes the geo-physical relationships for the `Rio Douro`, which is `adjacent` to `Albufeira Crestuma-Lever`. It also intersects four features: two from the geo-physical domain (`Intensidade Sísmica 6` and the `Biótopo Alto Douro Internacional`), and two from geo-administrative domain (the municipalities of `Porto` and `Vila Real`). The latter are examples of inter-domain relation-

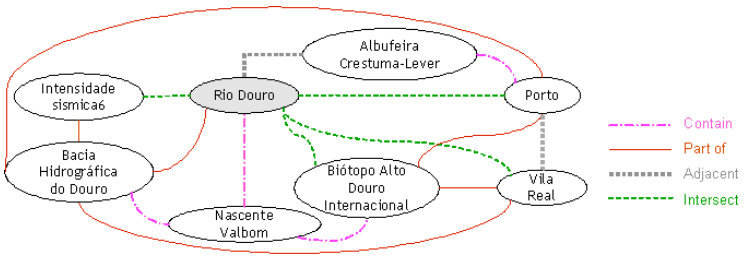

Figure 6. Excerpt of the inter-domain relationships among some geo-physical instances in GKB 2.0.

ships. The `Rio Douro` is also related to other physical instances, which have been omitted for the sake of clarity and space.

5 Publishing GKB contents for Semantic Web Applications

Once integrated in GKB 2.0, geographic knowledge can be exported in OWL, the Semantic Web format, for use by applications working with geographic information. Considering the different needs in the Semantic Web environment, generated ontologies may be organized in different ways. GKB 2.0 may export its stored knowledge with multiple data-structure organizations, corresponding to different visions of the information. It is possible, for example, to generate name-centered or feature-centered views. A named entity recognizer requires a name-centered representation for names look-up, whereas GIS applications are probably more interested in a loading of features satisfying some criteria and their associated attribute.

A name-centered view allows an application to know, for example, how many types a name is related to faster than with a feature-centered view. On the other hand, the feature-centered view enables the capture of the information (e.g., relationships, attributes and footprints) related to every single feature easier than with a name-centered representation.

Figure 7 presents an excerpt of the name-centered OWL format description of the geo-administrative name `Lisboa`. `Lisboa` is a `preferred name` (gn:att="P") provided by the INE `information source` (gn:is="INE") and has two other names. `Lissabon` is an `alternative name` (gn:att="A") in German and `Olissipo` is its `historic name` (gn:att="H"). The Wikipedia information source provides both names. `Lisboa` is also the name of different feature types. It may be a `province` (PRO), a `municipality` (CON), a `street` (RUA) and an `avenue` (AVE), among others.

The geographic knowledge output in OWL ontologies reflects the classes, relationships and the cardinality in the model described along the Section 3. The OWL sub-language used here is OWL-Lite. The restrictions of OWL-DL and OWL-Full are still not necessary in our representation. This fact allows any OWL reasoner handle the generated ontologies.

```
<gn:Geo-Name rdf:ID="GEO_ADM_30">
  <gn:names>
    <rdf:Bag>
      <rdf:li gn:name="Lisboa" xml:lang="PT-PT" gn:att="P" gn:is="INE"/>
      <rdf:li gn:name="Lissabon" xml:lang="DE-DE" gn:att="A" gn:is="WIKI"/>
      <rdf:li gn:name="Olissipo" xml:lang="EL-GR" gn:att="H" gn:is="WIKI"/>
    </rdf:Bag>
  </gn:names>
  <gn:geo_type_id rdf:resource="#PRO"/>
  <gn:geo_type_id rdf:resource="#CON"/>
  <gn:geo_type_id rdf:resource="#RUA"/>
  <gn:geo_type_id rdf:resource="#AVE"/>
</gn:Geo-Name>
```

Figure 7. An excerpt of a name-centered OWL representation from GKB 2.0.

```
<gn:Geo_Feature rdf:ID="GEO_PHY_145">
  <gn:names>
    <rdf:Bag>
      <rdf:li gn:name="Douro" xml:lang="PT-PT" gn:att="P" gn:is="IGeoE"/>
      <rdf:li gn:name="Duero" xml:lang="ES-ES" gn:att="A" gn:is="IGP"/>
    </rdf:Bag>
  </gn:names>
</gn:Geo_Feature>
```

Figure 8. An excerpt of a feature-centered OWL representation from GKB 2.0.

Figure 8 gives an excerpt of geographic names and attributes exported from a GKB 2.0 model to an OWL ontology. The feature `GEO_PHY_145` is describing the names and attributes for the `Douro river`. The `GEO_PHY` prefix represents features from physical domain whereas the `GEO_ADM` prefix encodes the features from administrative domain. The Douro river has a preferred name `Douro` in Portuguese `PT-PT` and an alternative name `Duero`, in Spanish `ES-ES`. Each name is associated with an information source. The first is provided by `IGeoE` whereas the second one is given by `IGP`.

Figure 9 shows an excerpt of how the feature types are represented in OWL. The excerpt describes the details about the Douro river. The type `Rio` and its attributes are provided by `IGeoE`. The source of Douro river is represented by the element `source_river` with a internal link to the feature, which has the identifier `GEO_PHY_120` (*Serra de Urbião* in Spain). The outlet of the river is `GEO_ADM_238` (municipality of Porto). Its tributaries are in number of 10, but Figure 9 just illustrates two of them, whose identifiers are `GEO_PHY_400` (`Paiva river`) and `GEO_PHY_401` (`Sousa river`). The `length` of the `Douro river` is `850 km`.

Figure 10 shows examples of the relationships involving the `Douro river`. The `Douro river` `intersects` the `municipality of Porto` (GEO_ADM_238) and the `Ribeira da Granja river` (`GEO_PHY_300`). In addition, the Douro river is `adjacent` to `Albufeira do Carrapatelo` (`GEO_PHY_198`) and is `part-of` the `Bacia Hidrográfica do Douro` (`GEO_PHY_100`). In this case, the `intersects`

```
<gn:geo_type_id gn:is="IGEO" rdf:resource="#Rio"/>
<rdfs:comment>Serra de Urbião - Spain</rdfs:comment>
<gn:source_river gn:is="IGEO" rdf:resource="#GEO_PHY_120"/>
<rdfs:comment>Porto - Portugal</rdfs:comment>
<gn:outlet_river gn:is="IGEO" rdf:resource="#GEO_ADM_238"/>
<gn:tributary gn:is="IGEO">
    <rdf:Bag>
        <rdf:li rdf:resource="#GEO_PHY_400"/>
        <rdf:li rdf:resource="#GEO_PHY_401"/>
    </rdf:Bag>
</gn:tributary>
<gn:length unit="km" gn:is="IGEO">850</gn:length>
```

Figure 9. An excerpt of types and its attributes in OWL in Geo-Net-PT02.

```
<rdf:Bag>
    <rdf:li>
        <gn:Geo_Relationship>
            <gn:rel_type_id rdf:resource="#INTERSECTS"/>
            <gn:geo_id>
                <rdf:Bag>
                    <rdf:li rdf:resource="#GEO_ADM_238"/>
                    <rdf:li rdf:resource="#GEO_PHY_300"/>
                </rdf:Bag>
            </gn:geo_id>
        </gn:Geo_Relationship>
    </rdf:li>
    <rdf:li>
        <gn:Geo_Relationship>
            <gn:rel_type_id rdf:resource="#ADJ"/>
            <gn:geo_id rdf:resource="#GEO_PHY_198"/>
        </gn:Geo_Relationship>
    </rdf:li>
    <rdf:li>
        <gn:Geo_Relationship>
            <gn:rel_type_id rdf:resource="#PRT"/>
            <gn:geo_id rdf:resource="#GEO_PHY_100"/>
        </gn:Geo_Relationship>
    </rdf:li>
</rdf:Bag>
```

Figure 10. An excerpt of relationships between geo-administrative and geo-physical domains in OWL in Geo-Net-PT02.

relationship was generated from coordinates, since it is not explicitly represented in GKB 2.0.

Finally, an other important characteristic of GKB 2.0 is that information can also be partially exported. Applications using ontologies generated by GKB 2.0 may have interest only in parts of its contents. For example, one may be inter-

ested only in the generation of features containing just historic names, features associated with types above a specific degree of granularity (e.g., localities or streets) or features provided from specific information sources.

Geo-Net-PT02 intends to be a reference for geographic information processing on the Portuguese territory, providing a large (wide and deep) coverage of concepts and instances. Geo-Net-PT01 filled this need for the geo-administrative domain whereas Geo-Net-PT02 is going to be enriched with geo-physical data, coordinates and multilingual names for features.

6 Related Work

Other works dealing with an environment to integrate and export ontologies are often in the scope of private companies or public initiatives. In both cases details about the construction of the resources are omitted or really do not exist. Two geographic information resources that are available for free or at a nominal license fee are the Getty Thesaurus of Geographic Names (TGN) [11] and Geonames (`geonames.org`).

TGN is a structured vocabulary including names and associated information about both current and historical wide-world places. Details about the construction of this resource are not available and the data-integration procedure is also unknown to us. TGN contains about 1 million places over the world, including both political entities (e.g. nations) and physical features (e.g. rivers) [12]. However, TGN lacks detailed information about Portugal (notoriously some of the most important street names).

Geonames is a large public geographical database, which contains a worldwide list of geo-administrative and geo-physical names. For Portugal, it has just 23,503 names in both domains. Historic feature types and names for Portugal (e.g., provinces) are also not provided by Geonames. Considering just the administrative domain, Geo-Net-PT01 has 34,519 distinct names for the geographic types above the level of localities (not including streets names) [13]. This leads us to conclude that the coverage of names supported by Geonames is very limited for Portugal.

7 Final Remarks

This article described ongoing work about the new version of GKB, detailing the data model and ontology format for the exported data. The model enables the generation of geographic ontologies representing knowledge integrated from geo-administrative and geo-physical domains. We also showed different ways for publishing GKB contents for Semantic Web applications, facilitating their parsing. Future work includes the processing (Extraction, Transformation and Loading (ETL)) of geographic data covering the physical domain and generation and release of Geo-Net-PT02.

Acknowledgements

Marcirio Chaves is member of the XLDB Node of Linguateca supported by grant POSI/PLP/43931/2001 from FCT, co-financed by POSI. This work is supported by grant POSI/SRI/47071/2002 (GREASE) from FCT, co-financed by POSI.

References

1. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2) (February 2006) 25–31
2. Chaves, M.S., Silva, M.J., Martins, B.: GKB - Geographic Knowledge Base. DI/FCUL TR 05–12, Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa (July 2005)
3. Chaves, M.S., Silva, M.J., Martins, B.: A Geographic Knowledge Base for Semantic Web Applications. In Heuser, C.A., ed.: Proc. of the 20th Brazilian Symposium on Databases, Uberlândia, Minas Gerais, Brazil (October, 3–7 2005) 40–54
4. Freitas, S., Afonso, A.P., Silva, M.J.: Mobile Geotumba: Geographic Information Retrieval System for Mobile Devices. In: Proc. of the 4th MiNEMA Workshop, Sintra, Portugal (July, 2-3 2006) 83–87
5. Silva, M.J., Martins, B., Chaves, M.S., Cardoso, N., Afonso, A.P.: Adding Geographic Scopes to Web Resources. CEUS - Computers, Environment and Urban Systems - Elsevier Science **30**(4) (July 2006) 378–399
6. Cardoso, N., Martins, B., Chaves, M.S., Andrade, L., Silva, M.J.: The XLDB Group at GeoCLEF 2005. In Peters, C., Gey, F.C., Gonzalo, J., Müller, H., Jones, G.J.F., Kluck, M., Magnini, B., de Rijke, M., eds.: Accessing Multilingual Information Repositories, 6th Workshop of the Cross-Language Evalution Forum, CLEF 2005, Vienna, Austria, 21-23 September, 2005, Revised Selected Papers. Volume 4022 of Lecture Notes in Computer Science., Springer (2005) 997–1006
7. Chaves, M.S., Santos, D.: What kinds of geographical information are there in the portuguese web? In Vieira, R., Quaresma, P., da Graça Volpes Nunes, M., Mamede, N., Oliveira, C., Dias, M.C., eds.: Proc. of the 7th Workshop on Computational Processing of Written and Spoken Portuguese, PROPOR 2006. Volume 3960 of Lecture Notes in Computer Science., Itatiaia, Rio de Janeiro, Brazil, LNAI 3960 - Springer (13th - 17th May 2006) 264–267
8. ISO - International Organization for Standardization: ISO 19109:2005 Geographic information – Rules for application schema. 1 edn. (June, 23 2005)
9. van Gigch, J.P.: System Design Modeling and Metamodeling. First edn. ISBN: 0306437406. Springer, New York (July 1991)
10. Alvestrand, H.: Request for Comments: 3066 - Tags for Identification of Languages. `http://www.ietf.org/rfc/rfc3066.txt` (January 2001) Accessed in January 2007.
11. TGN: Getty Thesaurus of Geographic Names. `www.getty.edu/research/conducting_research/vocabularies/tgn/` Accessed in January 2007.
12. Martins, B., Silva, M.J., Chaves, M.S.: Challenges and resources for evaluating geographical IR. In: Proc. of the 2nd Workshop on Geographic Information Retrieval at CIKM, Bremen, Germany (November 4th 2005) 65–69
13. Santos, D., Chaves, M.S.: The place of place in geographical IR. In: Proc. of the 3rd Workshop on Geographic Information Retrieval, SIGIR'06, Seattle, USA (August 10th 2006) 5–8

Navegando na Rede Semântica dos Topic Maps com o Ulisses

Giovani Rubert Librelotto[1], Renato Preigschadt de Azevedo[1], and José Carlos Ramalho[2] and Pedro Rangel Henriques[2]

[1] UNIFRA, Centro Universitário Franciscano, Santa Maria - RS, 97010-032, Brasil
{librelotto, rpa.renato}@gmail.com
[2] Universidade do Minho, Departamento de Informática
4710-057, Braga, Portugal
{jcr, prh}@di.uminho.pt

Abstract. A norma ISO-IEC 13250 *Topic Maps*, composta principalmente de tópicos interligados através de associações, define uma rede semântica estruturada sobre um sistema de informação, criando uma ponte entre a gestão de informação e os domínios de representação de conhecimento. A estrutura de dados que um topic map representa é um grafo; portanto, uma das melhores maneiras de visualizar um grafo é percebendo sua estrutura de vértices (tópicos) e arestas (associações). Assim sendo, este artigo tem por objetivo a visualização eficiente desta camada semântica através de navegadores Web, permitindo que todos os vértices (tópicos) do grafo topic map sejam acessados em uma navegação conceitual, seja por ligações HTML (*links*), seja navegando no próprio grafo visualmente. Para apresentar essa abordagem de navegação em *Topic Maps*, divide-se o artigo em 3 partes: primeiro, apresentam-se os conceitos básicos de *Topic Maps*, para então, descrever algumas técnicas de sua visualização. Por fim, descrever-se-á a ferramenta de visualização desenvolvida, caracterizando as técnicas empregadas em seu desenvolvimento.

1 Introdução

Topic Maps [Biezunsky et al., 1999] é um formalismo para representar conhecimento acerca da estrutura de um conjunto de recursos de informação e para o organizar em tópicos. Esses tópicos possuem ocorrências e associações que representam e definem relacionamentos entre os tópicos. A informação sobre os tópicos pode ser inferida ao examinar as associações e ocorrências ligadas ao tópico. Uma coleção desses tópicos e associações é designada *topic map*. *Topic Maps* pode ser visto como um paradigma que permite organizar, manter e navegar através da informação, permitindo transformá-la em conhecimento.

A idéia de navegação conceitual reflete a forma que a mente humana pensa: baseado em informações associadas. Por exemplo, quando se pensa no *Brasil*, automaticamente pode-se pensar:

– é um país que está situado na América do Sul (associação entre *país* e *continente*, no contexto *geografia*);

– possui mais de 170 milhões de habitantes (ocorrência do tipo *população*);
– é governado pelo presidente *Luís Inácio Lula da Silva* (associação entre *país* e *presidente*, no contexto *política*);

Assim, o que se pretende neste artigo é apresentar a atualização de uma ferramenta para visualização de *Topic Maps* baseada em conceitos de navegação em grafos, chamada *Ulisses*.

Tendo em vista que os *Topic Maps* são grafos compostos por tópicos (onde cada tópico representa um tema, identifica recursos e está associado com outros tópicos), o *Ulisses* foi desenvolvido inicialmente para providenciar navegadores estáticos que permitem percorrer estas redes de conceitos. Esses navegadores são compostos de páginas HTML que descrevem os tópicos e usam a idéia de hiperligações para implementar as associações e as ligações às ocorrências.

Portanto, o que se defende neste artigo é a proposta de navegação dinâmica sobre qualquer topic map sem a necessidade de gerar um conjunto de páginas HTML referente a cada tópico encontrado, além de permitir uma visualização do grafo do topic map a partir do tópico (vértice) onde o navegador se encontra.

Para ilustrar as idéias aqui introduzidas, esse artigo inicia apresentando ontologias e a norma *Topic Maps* na Seção 2. Uma técnica de visualização de *Topic Maps* encontra-se na Seção 3. O *Ulisses* é apresentado na Seção 4. A Seção 5 discute os trabalhos relacionados. Por fim, uma síntese do artigo e os trabalhos futuros são apresentados na conclusão.

2 Ontologias e Topic Maps

Uma ontologia [Gruber, 1993] é uma especificação ou *formalização de uma conceitualização*. Uma conceitualização é um conjunto de conceitos e suas relações entre si. Alternativamente, uma ontologia pode ser entendida como uma teoria lógica a qual dá uma explicação explícita de uma conceitualização, projetada para ser compartilhada por agentes para vários objetivos [Guarino and Giaretta, 1995].

Na área de Sistemas de Informação, na qual se encaixa este trabalho, ontologia é definida como um conjunto de conceitos e termos ligados entre si (numa rede) que podem ser usados para descrever alguma área do conhecimento ou construir uma representação para o conhecimento [Swartout and Tate, 1999]. Segundo Chandrasekaran [Chandrasekaran, 1999], ontologias são teorias de conteúdo sobre os tipos de objetos, propriedades de objetos e relacionamentos entre objetos que são possíveis em um domínio de conhecimento específico.

Historicamente, a norma *Topic Maps* (ISO 13250) [Biezunsky et al., 1999] foi definido para facilitar a fusão de diferentes esquemas de índices. Um formato comum para a anotação, usado para a indexação, é um passo crucial em direção ao objetivo da interoperabilidade entre esquemas de índices. O que é necessário ainda é a interoperabilidade semântica. Enquanto que a especificação *Topic Maps* garante interoperabilidade sintática, ontologias provém interoperabilidade semântica. Se for construído a partir de uma ontologia válida, *Topic Maps* podem prover interoperabilidade semântica não somente entre cada topic map, mas entre as aplicações que usam-nas.

Topic Maps é um formalismo para representar conhecimento acerca da estrutura de um conjunto de recursos de informação, organizando-o em *tópicos*. Esses tópicos têm ocorrências em recursos de informação e associações que representam e definem os relacionamentos entre os tópicos. A informação sobre os tópicos pode ser inferida ao examinar as associações e ocorrências ligadas ao tópico. Uma coleção desses tópicos e associações é chamada de topic maps.

Os *Topic Maps* podem ser definidos como uma descrição de um ponto de vista sobre uma coleção de recursos, organizado formalmente por tópicos, e pela ligação de partes relevantes do conjunto de informação aos tópicos apropriados [Pepper, 2000]. Um mapa de tópicos expressa a opinião de alguém sobre o que os tópicos são, e quais as partes do conjunto de informação que são relevantes para cada tópico.

Permitindo criar um mapa virtual da informação, os recursos de informação mantém-se em sua forma original e não são modificados. Então, o mesmo recurso de informação pode ser usado de diferentes formas, por diferentes mapas de tópicos. Como é possível e fácil modificar um mapa, a reutilização da informação é conquistada.

3 Visualização de *Topic Maps*

Antes de apresentar o *Ulisses*, esta seção introduz um modo de visualização de *Topic Maps*, que é o método seguido pelo próprio navegador.

3.1 Visualização de Tópicos e Associações

Como dito anteriormente, um topic map pode ser visto como um grafo, onde os tópicos podem ser vistos como os vértices e os relacionamentos como os arcos. Cada tópico é representado como um vértice no grafo. Assim, a visualização de cada vértice – disponibilizada quando o navegador se posiciona sobre um tópico em específico – deve conter todas as suas características: seus tipos, suas instâncias, seu indicador de tema, seus nomes e suas ocorrências, além de incluir uma referência a todas as associações a qual ele faz parte.

Os vértices são rotulados usando os nomes dos tópicos. Desta forma, sempre que um vértice está relacionado com outros (sejam relações classe/instância ou associações propriamente ditas), o rótulo do vértice que está relacionado é o seu próprio nome.

Cada associação binária pode ser representada como um arco conectando os dois tópicos. Ao se posicionar em determinada associação do grafo, as informações sobre a mesma serão apresentadas. Por exemplo: em uma associação do tipo *orientar*, os tópicos participantes são visualizados (*Brasil* e *Brasília*) e os respectivos papéis de atuação (*país* e *capital*), além do contexto onde ela está inserida (*localização*). Por sua vez, os papéis de atuação são usados para designar os arcos que saem desse vértice (isto é, para descrever as associações nas quais o tópico atua).

3.2 Filtragem em *Topic Maps*

Um topic map pode conter milhões de tópicos e associações. Portanto é essencial selecionar a informação relevante pois pode ser impossível mostrar todo o conjunto de tópicos e associações de uma forma eficiente.

Técnicas de filtragem de informação são necessárias para selecionar e apresentar somente a informação relevante. A ferramenta aqui apresentada habilita os utilizadores a filtrar tópicos e associações de acordo com seu tipo ou pelos papeis de atuação em associação, por exemplo.

É possível fazer aqui uma analogia entre *Topic Maps* e mapas geográficos. Geralmente, não se encontra toda a informação desejada sobre um país em um único mapa; em vez disso, existem sub-mapas específicos de diversos contextos, tais como: mapas topográficos, mapas políticos, mapas econômicos, etc.

Da mesma forma, tópicos e associações podem ser classificados em diferentes contextos. Sendo assim, sub-mapas[3] distintos dentro de um topic map poderão fornecer as informações necessárias ao utilizador de acordo com seu interesse. Se ele estiver interessado em *teatros*, os tópicos relevantes seriam *autor*, *comédia*, *cultura*, *ator*, etc. Esta é uma forma de filtrar a informação de acordo com um contexto específico.

Uma vez filtrados os tópicos e associações, eles necessitam ser representados eficientemente. Uma sugestão para a redução do número de tipos a serem representados é a agregação de tópicos e associações com um algoritmo de classificação, como o *galois lattices* [Godin et al., 1995]; este algoritmo diz que se pode agrupar objetos que compartilhem propriedades comuns. Esses grupos são chamados de classes. Portanto, pode-se somente distinguir classes de tópicos na representação em vez de diferenciar todos os tópicos. Por exemplo, os tópicos *Brasil*, *América do Sul* e *Brasília* podem ser representados da mesma maneira porque eles pertencem à mesma classe *lugar*. Este mecanismo de classificação torna possível visualizar *Topic Maps* com diferentes níveis de detalhes.

Obviamente, a informação visualizada é menos precisa, mas isso pode ser aceitável dependendo do propósito da navegação. Contudo, é possível visualizar uma informação mais precisa quando o utilizador foca em uma parte específica do topic map. Por exemplo, ao estar situado no tópico *Brasília*, a informação mostrada será que *Brasília é uma cidade*, o que é mais preciso que simplesmente afirmar que *Brasília é um local*. O mesmo acontece quando verifica-se a associação entre *Brasília* e *Brasil*; a informação apresentada é *Brasília é uma cidade situada no Brasil*.

4 *Ulisses*– o navegador

A idéia sobre a qual se baseou o *Ulisses* é a idéia da navegação conceitual, a qual pode ser descrita como: quando se está posicionado sobre um certo conceito, a

[3] Um sub-mapa é o conjunto de tópicos e associações que representam um sub-domínio, dentre vários que podem existir num topic map. Um mesmo tópico pode estar inserido dentro de vários sub-mapas, de acordo com as relações que ele possui.

ferramenta de navegação mostrará as informações associadas a este conceito em particular; se for escolhido algum dos outros conceitos relacionados, a navegação muda para a visão deste novo conceito; se for escolhido algum dos recursos de informação, o sistema mostrará o conteúdo do próprio recurso.

O *Ulisses* foi projetado com base em três razões principais:

- Ser um protótipo eficiente, diminuindo o tempo de criação de um navegador para *Topic Maps*;
- Permitir visualização do topic map em formato de um browser Web e em formato de um grafo;
- Criar uma ferramenta que usa *Topic Maps* que seja facilmente distribuída, instalada e usada por todos;
- Ser baseado em XML e XTM (XML Topic Maps).

Em termos de implementação, foi estabelecido que a navegação seria realizada através de *Web browsers*. Então, todas a informação sobre cada conceito seria apresentada em páginas Web, não necessitando assim de utilitários extras, pois todo o computador adaptado à internet teria capacidade de navegar no topic map, independente de plataforma ou sistema operativo.

4.1 Abordagem escolhida na criação do *Ulisses*

Além da geração de um grafo para a visualização do topic map, o desenvolvimento do *Ulisses* seguiu uma abordagem paralela: a criação de um conjunto de páginas, onde cada uma representa um tópico ou uma associação contido no topic map.

Os *links* determinam as relações entre os vértices; desta forma, os *links* são identificados pelos nomes dos tópicos que eles representam. Por exemplo, na página gerada para o tópico *Brasília* (conforme citado na subseção anterior), haveria a menção de uma associação com o tópico *Brasil*, onde a frase "*é uma cidade que está situada no*" representa o papel de atuação de *Brasília* perante *Brasil*. A palavra *Brasil*, nesta frase, possui uma ligação à página que representa o tópico *Brasil*.

O algoritmo de funcionamento seguido pelo *Ulisses* para a criação das páginas, as quais oferecerão a navegação conceitual a partir do topic map, pode ser descrito assim:

1. Definição da hierarquia de tópicos do topic map, percorrendo-se todos os tópicos e ligando-os de acordo com as relações classe/instância. Se no final desse processo verificar que não há um tópico raiz, é então criado um novo tópico que servirá de raiz para o topic map. Este novo tópico terá, como suas instâncias, todos os tópicos raízes. Desta forma, todas as sub-árvores estarão conectadas entre si, criando a taxonomia do topic map;
2. Criação de uma página inicial para o topic map, usando o nome do tópico raiz para a sua identificação e mostrando a hierarquia de tópicos de acordo com seu tipo;

3. Criação de uma página para cada tópico, onde é apresentado os seus tipos, seu indicador de tema, suas ocorrências, as associações a qual ele faz parte e os tópicos que são suas instâncias. Deste modo, quando se faz uma referência a algum tópico, cria-se uma ligação Web (*link*) para o tópico relacionado;
4. Criação de uma página para cada associação, indicando seu tipo e seu contexto, além de seus membros e respectivos papeis de associação;
5. Criação de um grafo para cada tópico visualizado, para dar a noção ao utilizador em que vértice do grafo do topic map ele se encontra.

Seguindo esses passos, o *Ulisses* efetua a criação do do Web site que corresponde a navegação conceitual do topic map.

4.2 *Ulisses* 1.0: navegador conceitual estático

Em um primeiro momento, a variante inicial desenvolvida para o *Ulisses* era responsável pela geração de um Web site de uma só vez, através da transformação prévia do topic map (em formato XTM), para um conjunto de páginas HTML com algumas componentes em *Javascript* que não voltam a ser geradas em toda a navegação, definindo assim a criação de um Web site estático [Librelotto et al., 2003].

Esta versão inicial do *Ulisses* oferecia opções que permitem rápidas modificações em todo o website, como por exemplo, alterações em termos de *layout* (cores e tamanho das fontes), comentários a serem incluídos nas páginas criadas e a inserção de imagens.

O principal problema detectado com essa versão era o fato de sempre gerar uma página HTML para cada tópico encontrado no topic map, gerando um diretório no sistema operacional com vários arquivos neste formato, sendo que todos eles possuem em média 1,3 KB de tamanho físico. Isto vem do fato que cada página criada possui apenas as informações referente a cada um dos tópicos individualmente, portanto não é um volume grande de dados a ser visualizado no browser.

4.3 *Ulisses* 2.0: navegador conceitual dinâmico

Na versão atual do *Ulisses*, as transformações em tempo de execução são implementadas com transformações XML, processadas por um código PHP (*PHP Hypertext Processor*[4]). A página inicial é uma chamada a uma PHP à raiz do topic map (que vai servir de ponto de entrada para a navegação). O código PHP aplica a transformação no topic map e produz uma visão HTML. Nessa visão, todos os *links* gerados são chamadas ao mesmo PHP mas com diferentes parâmetros, que indicam a localização da nova posição. Ou seja, permitirá a visão do tópico relacionado.

Esta é a solução mais prática porque somente necessita de dois arquivos: o topic map (especificado de acordo com a sintaxe XTM) e o código PHP. Contudo, as transformações em tempo de execução consomem um tempo por vezes elevado,

[4] http://www.php.net/docs.php

pois o topic map é interpretado em tempo de execução. Desta forma, a cada nova chamada ao código PHP, o topic map deve ser interpretado mais uma vez, para formar a visualização a ser disponibilizada no navegador.

A principal contribuição desta nova variante do *Ulisses* é a visualização do grafo do topic map em todas as páginas geradas. A geração deste grafo fica a cargo do *Graphviz*[5], uma ferramenta *open-source* para geração de grafos e redes abstratas. Desta forma, em todas as páginas Web do *Ulisses* é posto um grafo, no qual situa-se o tópico em questão e as suas ocorrências externas e internas, além de todos os tópicos relacionados com este.

No grafo gerado, cada vértice é programado como um *link*, o que permite que ao acionar algum dos vértices do grafo, a visualização do tópico acionado será gerada instantaneamente, apresentando todas as informações referentes ao novo vértice, além do grafo sob o seu ponto de vista.

A variante atual também permite a navegação sobre topic maps representados de acordo com a sintaxe XTM. Visando tornar o *Ulisses* mais abrangente, foram adicionadas folhas de estilos XSL[6] que permitem transformar os topic maps representados na sintaxe HyTime para a sintaxe XTM [Ogievetsky, 2000], de modo a permitir gerar navegadores para topic maps em HyTime. Assim, a navegação conceitual pode ser gerada para os principais formatos XML para a representação de *Topic Maps*. Além disso, o tradutor das sintaxes AsTMa=[7] e LTM[8] para a sintaxe XTM pode ser utilizado para os topic maps que se encontram num destes formatos não-XML.

A seguir são apresentados as principais páginas geradas automaticamente pelo *Ulisses*, bem como alguns de seus *layouts*.

Página principal do topic map: A página principal de um topic map no *Ulisses* é apresentada na Figura 1. Esta imagem expõe no seu lado esquerdo, a hierarquia de tópicos, associações, ocorrências e papéis de atuação obtida a partir da ontologia do topic map, representados na forma de um menu.

No corpo do navegador, cada um dos ítens apresentados serve como um acesso ao tópico correspondente, na forma de uma ligação Web. Desta forma, ao selecionar um dos *links*, será carregado a página referente ao mesmo.

Visualização de um tópico: A visualização de um tópico pode ser composta por 6 partes, conforme pode ser visto na Figura 2:

Nome do tópico e seu tipo: no cabeçalho da página aparece o nome do tópico (neste caso, *Giovani Librelotto*). O tipo do tópico é uma ligação ao seu tipo. No caso, este tópico é uma instância de *Person*;

Ocorrências Internas e Externas: as ocorrências que caracterizam o tópico em questão, juntamente com seus tipos, são apresentadas nesta parte. Por

[5] http://www.graphviz.org
[6] http://www.w3.org/Style/XSL/
[7] XTM::AsTMa – http://cpan.uwinnipeg.ca/htdocs/XTM/XTM/AsTMa.html
[8] XTM::LTM – http://cpan.uwinnipeg.ca/htdocs/XTM/XTM/LTM.html

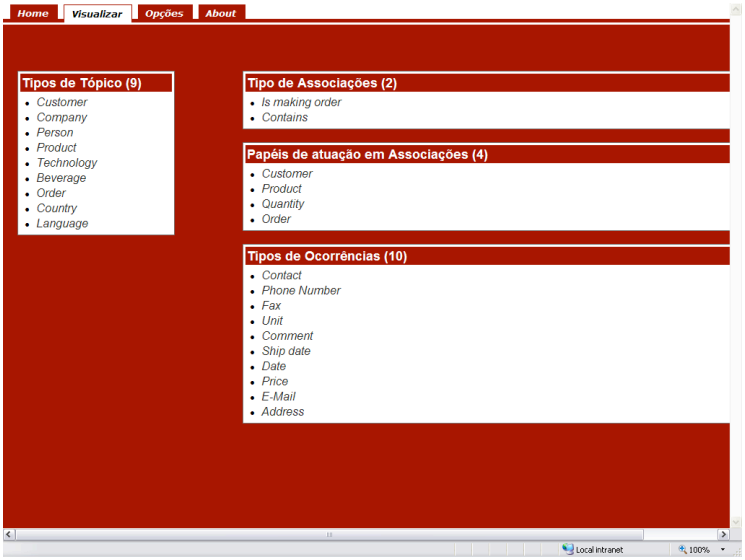

Fig. 1. Visualização inicial de um topic map no *Ulisses*

exemplo: uma ocorrência externa é a URL definida como uma referência a um recurso do tipo *E-mail*, além de uma ocorrência interna do tipo *Address*;

Papéis de atuação: para cada tópico associado, é visualizado o papel de atuação do tópico em questão em cada associação na qual ele participa. Por exemplo: *Giovani Librelotto* é o comprador (*Customer*) responsável pela compra 01 (*Order 01*);

Instâncias: lista dos tópicos que são instâncias do tópico em questão. No caso da Figura 2, não há qualquer instância pois o tópico corrente não é tipo de nenhum outro.

Grafo do topic map: para evitar de gerar um grafo completo que abranja todo o topic map, será criado um grafo que tem como contexto o tópico em questão, no qual encontram-se todos os tópicos associados e as ocorrências deste tópico.

Como pode ser visto na Figura 2, o *Ulisses* permite a visualização da rede semântica formada pelo topic map, sob a ótica do tópico atual. Além disso, é possível reconfigurar a geração do grafo de forma a definir quantos níveis do grafo devem ser mostrados na imagem gerada, para que fique de acordo com a visão do utilizador.

Visualização de uma associação A visualização de uma associação pode ser composta por 5 partes, conforme pode ser visto na Figura 3, onde se apresenta a visualização da associação *Contains*:

Nome do tipo da associação: o nome do tipo da associação aparece no alto da página (neste caso, *Contains*);

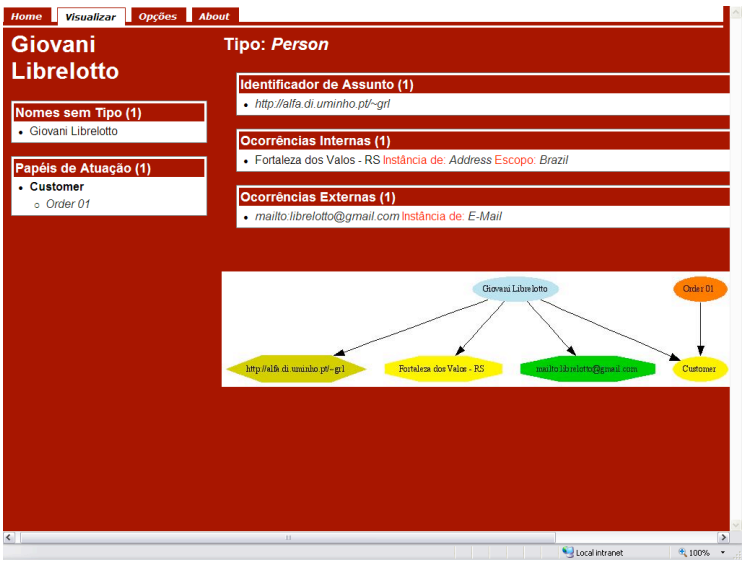

Fig. 2. Visualização de um tópico no *Ulisses*

Papeis de Atuação: cada membro da associação está relacionado aos demais membros através de um papel de atuação. Desta forma, lista-se todos os tópicos que desempenham cada um dos papéis de atuação.

Na Figura 3, o grafo do topic map não está totalmente visível devido ao tamanho do mesmo. Por isso, destaca-se o grafo referente à associação Contains na Figura 4.

5 Trabalhos Relacionados com o *Ulisses*

Atualmente, os navegadores (*browsers*) para *Topic Maps* mais conhecidos pela comunidade acadêmica de Semantic Web são: *Omnigator*, *Topic Map Designer*, *xSiteable* e *TMNav*.

O *Ontopia Omnigator*[9] talvez seja o navegador de *Topic Maps* mais difundido. O *Omnigator* é uma aplicação que permite carregar e navegar sobre qualquer topic map, usando um browser para a Web.

Um grande inconveniente do *Omnigator* é que a versão disponibilizada livremente está limitada a 500 tópicos e associações; desta forma, quando um topic map possuir uma quantidade superior a este limite, a navegação não é permitida ao utilizador.

Entretanto, uma vantagem significativa do *Omnigator* é o seu módulo chamado *Vizigator* [Gennusa, 2004]. O *Vizigator* propicia uma navegação gráfica de *Topic Maps* fornecendo: uma visão geral da estrutura da informação, de forma intuitiva

[9] Disponível em: http://www.ontopia.net/omnigator/models/index.jsp

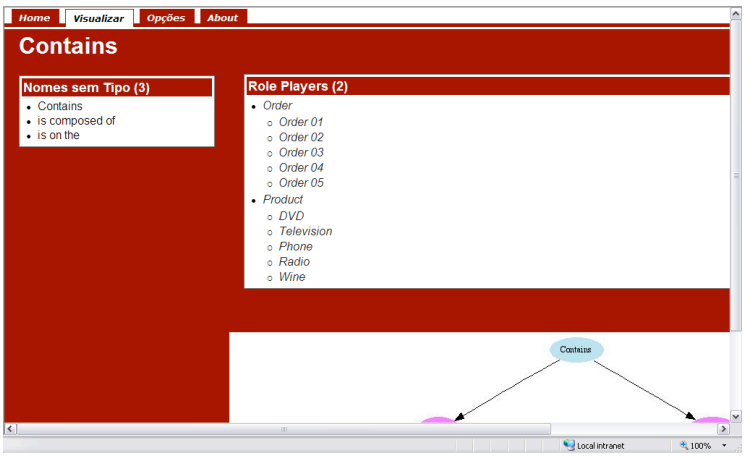

Fig. 3. Visualização de uma associação no *Ulisses*

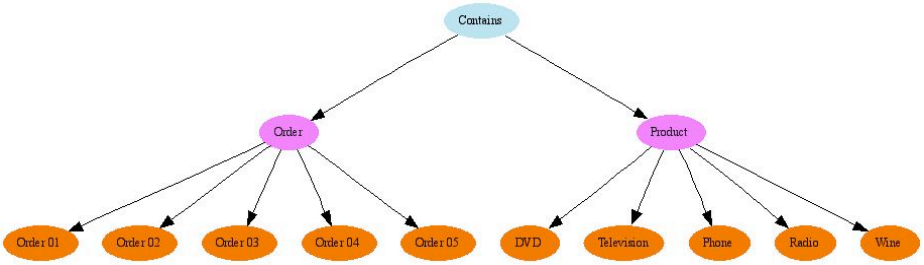

Fig. 4. Grafo de um tipo de associação no *Ulisses*

e instantânea; uma travessia no grafo formado pelo topic map; a habilidade de ver e entender os relacionamentos; visões em diferentes níveis de granularidade; e acesso intuitivo a modelos de dados não-familiares ao utilizador.

A visualização gráfica fornecida pelo *Vizigator* se caracteriza, portanto, num diferencial a favor do *Omnigator*, por propiciar duas navegações distintas sobre *Topic Maps*: numa visão HTML contendo toda a informação sobre cada tópico (como faz o *Ulisses*) ou numa visão gráfica, apresentando o grafo propriamente dito (como faz o *Topic Map Designer*).

O *Topic Map Designer*[10] é um aplicativo que permite a edição de *Topic Maps* e sua conseqüente visualização gráfica. Contudo, ele não tem como objetivo ser um ambiente completo para a criação de *Topic Maps* e suporta somente topic maps no formato HyTime e possui limitações quanto ao tamanho de topic maps (não podem ter mais de 100 tópicos).

[10] Topic Map Designer: http://www.topicmap-design.com/en/tutorial.htm

O *xSiteable*[11] é uma ferramenta de desenvolvimento de websites criada em XSLT, com um pacote de administração PHP. De modo geral, possui características muito similares ao *Ulisses*, quanto à formação do website, representação das características dos tópicos e associações, assim como na visualização de todas estas informações em uma página HTML. Ambas utilizam tecnologia XSLT para a geração de páginas HTML, a partir de um documento XTM.

Porém, o *xSiteable* apenas possui a opção de criar uma página HTML por tópico. Ao contrário do *Ulisses*, o *xSiteable* não permite a criação de um website dinamicamente. Além disso, não suporta qualquer outro formato de topic maps, além de XTM e CSXTM[12].

A mesma vantagem acima – permitir a navegação de topic maps armazenados num modelo relacional – o *Ulisses* possui sobre o *TMNav*. O *TMNav*[13] é uma aplicação *Java/Swing* para propiciar a navegação em topic maps que utiliza uma interface baseada em grafos. Ele funciona sobre o aplicativo *TM4J*, permitindo assim uma navegação em páginas Web ou num grafo dinâmico que utiliza a biblioteca *TouchGraph*[14].

6 Conclusão

Em resumo, o *Ulisses* representa uma ferramenta com objetivos claros (navegação conceitual em *Topic Maps*), baseada em conceitos de navegação em grafos, a qual utiliza tecnologias bastante difundidas para sua implementação, não o tornando dependente de plataforma ou aplicativo específico. O objetivo do desenvolvimento do *Ulisses* é incentivar o uso de *Topic Maps*, ensinando os princípios básicos do paradigma.

O *Ulisses* fornece uma navegação completa sobre *Topic Maps*, gerados ou não a partir de outras ferramentas. A representação do conhecimento é apresentado de uma forma simples e precisa, formando uma rede semântica baseada em tópicos e associações. Quando se acessa às informações referente a um determinado tópico, visualizam-se suas características (o seu tipo, suas instâncias, seus identificadores de tema, seus nomes e suas ocorrências) e as associações relacionadas com este tópico (incluindo os papéis de associação atuado por ele e os tópicos associados).

Constata-se, na prática, que o *Ulisses* é uma ótima solução para a fase de desenvolvimento, quando o topic map está constantemente a ser alterado. Como o *Ulisses* interpreta a especificação na hora em que o serviço é requisitado, qualquer alteração que tenha sido produzida no arquivo XTM é logo refletida no visualizador. Nessa fase, a navegação proporcionada pelo *Ulisses* pode ser

[11] xSiteable: http://www.shelter.nu/xsiteable/

[12] CSXTM é um formato de representação de *Topic Maps* compacto, simplificado e baseado em XTM. Desenvolvido pelos mesmos criadores do *xSiteable*, utiliza-se folhas de estilos XSL para a conversão de XTM para CSXTM, e vice-versa. Detalhes em http://www.shelter.nu/csxtm.html

[13] TMNav: http://tm4j.org/tmnav.html – TM4J: http://tm4j.org/

[14] TouchGraph: http://touchgraph.sourceforge.net/

utilizada para certas verificações, como por exemplo, a correção dos nomes dos tópicos e dos seus contextos. Porém, numa situação de produção, em que o topic map está estável, pode ser preferível usar uma solução em que o visualizador possa ser aberto por qualquer *browser* sem requerer a presença do processador.

Foi precisamente baseado neste requisito que levou-se a manter a versão inicial do *Ulisses*, a qual pode ser comparada a um compilador, pois gera um conjunto de páginas HTML estáticas. Assim, o *Ulisses* é invocado uma vez e o resultado permite efetuar navegações com qualquer *browser* que o utilizador final escolha, pois resultado são vários arquivos HTML.

No que diz respeito a trabalhos futuros, está a ser estudado um componente que irá permitir ao utilizador especificar o aspecto visual do Web site gerado (por enquanto, apenas há uma configuração da interface). Desta forma, a visualização da ontologia representada no topic map visualizado pode ser apresentada da forma mais propícia.

References

[Biezunsky et al., 1999] Biezunsky, M., Bryan, M., and Newcomb, S. (1999). ISO/IEC 13250 - Topic Maps. ISO/IEC JTC 1/SC34. http://www.y12.doe.gov/sgml/sc34/document/0129.pdf.

[Chandrasekaran, 1999] Chandrasekaran, B. (1999). What Are Ontologies, and Why do We Need Them? In *IEEE Inteligent Systems and their applications*, volume vl 9, n 1. IEEE.

[Gennusa, 2004] Gennusa, P. (2004). Ontopia's Vizigator(tm) - Now you see it! In *XML 2004 Conference and Exposition*, Washington D.C., U.S.A. IDEAlliance. http://www.idealliance.org/proceedings/xml04/papers/311/311.html.

[Godin et al., 1995] Godin, R., Missaoui, R., and Alaoui, H. (1995). Incremental concept formation algorithms based on galois (concept) lattice. In *Computational Intelligence*, volume 11(2).

[Gruber, 1993] Gruber, T. R. (1993). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N. and Poli, R., editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands. Kluwer Academic Publishers.

[Guarino and Giaretta, 1995] Guarino, N. and Giaretta, P. (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. In Mars, N., editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. Ed. Amsterdan: ISO Press.

[Librelotto et al., 2003] Librelotto, G. R., Ramalho, J. C., and Henriques, P. R. (2003). Ontology driven Websites with Topic Maps. In *The International Conference on Web Engineering*, Oviedo, Spain.

[Ogievetsky, 2000] Ogievetsky, N. (2000). XSLT stylesheets for converting ISO 13250 Topic Map documents into XTM 1.0 syntax. http://www.cogx.com/xslt4tm2xtm.html.

[Pepper, 2000] Pepper, S. (2000). The TAO of Topic Maps - finding the way in the age of infoglut. Ontopia. http://www.ontopia.net/topicmaps/materials/tao.html.

[Swartout and Tate, 1999] Swartout, W. and Tate, A. (1999). Ontologies. In *IEEE Inteligent Systems and their applications*, volume vl 14, n 1. IEEE.

Modelação de Workflows com UML e Ferramentas Declarativas

Rui Gamito, Luís Arriaga da Cunha, and Salvador Abreu

LNEC e Universidade de Évora
{rgamito,lac}@lnec.pt, spa@di.uevora.pt

Abstract. A linguagem UML é actualmente uma ferramenta largamente disseminada para a modelação de sistemas de informação. No entanto, entre a esquematização conceptual de um sistema e a sua implementação existe um passo crucial de transformação que nos propomos melhorar. Este artigo apresenta trabalho para o desenvolvimento de um sistema de geração de sistemas aplicacionais, directamente baseados na especificação UML de diagramas de actividade (geração de workflows) e diagramas de classe (geração de bases de dados).
A transposição entre a modelação e a implementação é feita através do XMI gerado por uma ferramenta de modelação UML, servindo de base para a geração do sistema final. Por sua vez, este último suporta-se num *backend* implementado na linguagem de Programação em Lógica ISCO, tirando partido da propriedades declarativas desta linguagem no acesso a bases de dados, e um *frontend* AJAX, facilitando alterações assíncronas que venham a ser realizadas sobre as especificações do sistema, depois de gerado.
Este artigo, sendo relativo a um *work in progress* não propõe ainda conclusões definitivas sobre as implicações desta abordagem, mas já apresenta alguns resultados preliminares relativamente a comparações com outro sistema de gestão de fluxo de dados.

1 Introdução

O uso de uma ferramenta de modelação UML como meio para produzir código de programação é um objectivo muito apetecido, mas não muito conseguido, simplesmente por a UML não ser uma linguagem gráfica de *programação* [10], mas sim de *modelação*. A falha entre a modelação e a produção de código funcional e auto-suficiente faz-se sentir quando atendemos ao facto de muitas ferramentas apenas gerarem XML a partir dos modelos, enquanto outras conseguem realmente gerar código, por exemplo Java, mas apenas para diagramas de classe e componentes, como é o caso do *Rational Rose* [7].

No que toca ao uso disseminado da UML [11], existe uma diferença considerável na aceitação do UML em diversas organizações. No entanto, os autores deste trabalho consideram que a linguagem UML é uma ferramenta que proporciona, entre outros, uma infra-estrutura rigorosa a um nível de abstracção elevado, bem como suporte de interoperabilidade e integração entre vários domínios,

a nível semântico, considerando válida a premissa de que o uso do UML é uma vantagem.

Ao longo deste documento é descrito o procedimento e metodologia usados para a tentativa de construção de um sistema de geração de sistemas aplicacionais a partir directamente da modelação de diagramas UML. O objectivo do trabalho é tornar possível a criação de sistema de controlo de fluxo de dados ou uma base de dados, somente a partir da modelação UML de diagramas de actividades e classes, respectivamente. Adicionalmente, faz também parte do objectivo disponibilizar um mecanismo de controlo, edição e visualização de *workflows* realizado em AJAX [12], permitindo a substituição de um editor UML externo. Tenta-se também que o editor seja simples, embora completo, fugindo um pouco da complexidade de outros *Workflow Management Systems* (WfMS), como por exemplo o *OpenFlow* [1], componente do *ZOPE*, um sistema de gestão de conteúdos.

O trabalho é na sua grande maioria baseado na linguagem ISCO [6], uma linguagem assente no Prolog ampliada sintacticamente para descrever classes, predicados e *goals* ISCO, sequências, restrições de integridade e regras de controlo de acesso.

Este documento está organizado da seguinte forma: na secção 2 é descrita a problemática das escolhas e decisões que influenciaram a evolução do trabalho até ao ponto em que se encontra actualmente; na secção 3 são apresentados os pormenores técnicos relevantes, tais como estruturas de dados, arquitectura da aplicação, regras de utilização e exemplos de código, aplicados quando possível a um exemplo concreto; na secção 4 são apresentados alguns resultados e métricas sobre XMI, XSLT e ISCO; na secção 6 são apresentadas algumas conclusões preliminares sobre o trabalho realizado e são mencionadas as possibilidades de alargamento da aplicação a novas funcionalidades.

2 Procedimentos e Métodos

O editor de UML eleito para realizar a modelação é o ArgoUML [14]. Esta escolha deve-se ao facto de ser *open source*, possuir exportação dos modelos para XMI [2], suportar OCL (*Object Constraint Language*) e possuir geração de código Java (relevante para efeitos de comparação com o que necessita de ser gerado para ISCO). A contribuir para a sua escolha está também o facto de ter uma interface relativamente simples e amigável, bem como o facto de ser implementado em Java, existindo portanto a facilidade de uso em qualquer sistema operativo. Foram analisadas outras ferramentas de desenho e modelação UML, nomeadamente *Dia* [8] e *Umbrello* [13], não sendo objectivo deste trabalho apresentar uma comparação entre estas.

De acordo com os objectivos do trabalho, apenas se têm em conta diagramas que possuam uma "consequência física" directa, isto é, os diagramas que conseguem realizar por si só uma representação completa de algum sistema, que se seja por sua vez passível de possuir uma representação em ISCO. Da totalidade de diagramas oferecidos pelo UML, escolhem-se então dois: diagrama de classes

(possuindo consequências directas sob a forma de base de dados) e diagrama de actividades (com a sua consequência directa sob a forma de *workflows*). Apenas estes dois modelos são considerados, atendendo à sua complexidade individual - especialmente no que diz respeito aos *workflows* - e por se considerar suficiente como representantes de modelos UML.

É importante referir neste ponto que, apesar de se considerarem dois tipos de diagramas, o principal foco de trabalho é na definição de *workflows* a partir dos diagramas de actividade.

Surge então a necessidade de realizar um mapa exaustivo da representação de ambos os diagramas em XMI. A definição deste mapa é extremamente importante para permitir extrair o máximo de informação do XMI.

Para a recolha de informação do XMI a escolha residiu prontamente no uso de XSLT[1] [3] [4] Com o auxílio das transformações XSLT gera-se directamente código ISCO.

As estruturas de dados utilizadas assentam na base *Object-Relational* do ISCO, sob a forma de uma estrutura de classes. Uma vez que se opta por gerar ISCO a partir do XMI, as relações são automaticamente criadas e populadas. Na construção do modelo *Object-Relational* é definida uma separação clara entre o que é informação estática e o que é informação dinâmica, sendo a última correspondente às instâncias de execução dos *workflows* (execução de actividades e eventos, incluindo a chegada de informação de eventos externos).

No que toca ao mecanismo de execução dos *workflows*, este é totalmente construido em ISCO e permite criar as devidas instâncias dos modelos, bem como executá-los, obedecendo à sua lógica definida no UML.Tratando-se de um *work in progress*, apenas alguns padrões de controlo de fluxo são suportados de momento, nomeadamente os padrões *"Sequence"*, *"Parallel Split"*, *"Synchronization"* e *"Exclusive Choice"* [15] [16]. É no entanto objectivo que exista um suporte para a grande maioria.

Para a apresentação amigável da ferramenta é também usada uma das propriedades proeminentes do ISCO: a sua interface cómoda com o PHP. Desta forma temos a prespectiva extremamente aliciante de controlar o *workflow* via internet.

O modelo da aplicação é até este ponto o exemplificado na figura 1

[1] Já que XMI é uma formatação XML e o XSL é perfeito para lidar com esta formatação

Na figura 1 nota-se um problema bem claro de falta de integração entre a modelação e a visualização. É necessário uma ferramenta externa para produzir o modelo e qualquer necessidade de ajuste do mesmo tem que ser realizada pela mesma ferramenta externa. Por outro lado, para visualizar e interagir com as instâncias dos *workflows* é necessário outro suporte, já que a) não existe retro-propagação de informação (ISCO $\Longrightarrow$ ArgoUML) e b) o ArgoUML não possui, obviamente, um mecanismo de execução e controlo de fluxo. Para piorar, qualquer edição do modelo do *workflow* põe em risco possíveis instâncias que já estejam a "correr", dado que a forma de *input* de informação proveniente de um XMI está implementada de forma completamente invasiva.

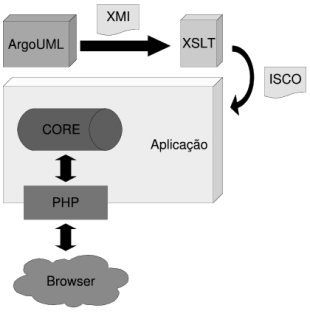

Fig. 1. Layout inicial da aplicação

São consideradas duas opções para resolver este problema:

- Continuar com o ArgoUML para re-edição dos modelos, e capacitar a introdução de dados do lado do ISCO de um mecanismo menos invasivo que se pudesse ajustar às mudanças, bem como preservar a informação possível de quaisquer instâncias activas;
- Desenvolver um editor integrado na aplicação, embora mantendo activa a entrada de XMI de forma genérica e não apenas para o ArgoUML.

A hipótese escolhida é a segunda, permitindo quebrar a dependência de uma ferramenta externa[2] e abrindo ao mesmo tempo a possibilidade de construir uma ferramenta completa de modelação, edição e gestão de *workflows* completamente integrada.

Nesta fase é feito um ajuste aos objectivos do trabalho, agora mais centrado nos *web-services*. Consideram-se hipóteses de interface para o utilizador com PHP simples, *applets* de Java e AJAX, tendo sido o último o escolhido. Após testes com AJAX apresenta-se claro que é a ferramenta ideal já que, aproveitando as suas capacidades de comunicação assíncrona com o servidor, se consegue proporcionar a melhor experiência ao utilizador tendo, por exemplo, a visualização em tempo real dos acontecimentos na execução do *workflow*. O facto da generalidade dos browsers suportarem JavaScript por omissão também ajudou na escolha do AJAX em detrimento das *applets* Java. No que diz respeito ao PHP, não se pode ignorar a vantagem da comunicação com o ISCO, pelo que é também mantido este componente.

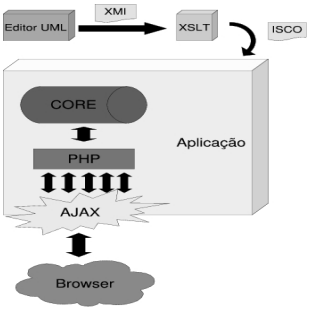

Fig. 2. Layout final da aplicação

A figura 2 representa o estado da arquitectura nesta fase.

[2] Embora mantendo-se o suporte para a entrada de modelos na aplicação a partir da interpretação de XMI

3 Modelo e Representação

Os autores de [9] sustentam que os diagramas de actividade suportam a maioria dos padrões de controlo de fluxo comuns, e mesmo outros que não são tipicamente suportados por WfMS comerciais. No entanto os diagramas de actividade *standard* apresentam algumas falhas de formalização (através de premissas OCL), sendo que a sua interpretação pode ser ambígua quando traduzida para linguagem natural. Embora estas falhas, exploradas ao longo de [9] sejam relativas apenas à *UML Revision 1.4*, podemos ver em [17] que estas não foram completamente resolvidas na revisão 2.0.

3.1 Exemplo

A figura 3 representa uma modelação UML de um diagrama de actividades muito simples. Todos os exemplos de código presentes neste artigo são referentes a esta modelação concreta. Este exemplo apenas exemplifica os padrões possíveis neste momento, de onde resulta a sua simplicidade.

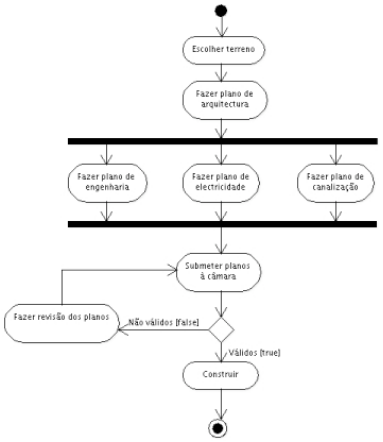

Fig. 3. Exemplo de diagrama de actividades

3.2 Relações e estados

Como foi referido na secção 2, toda a aplicação acenta sobre ISCO, que vai mantendo a persistência dos modelos estáticos dos *workflows* e principalmente da sua respectiva informação dinâmica.

Seguidamente são apresentados os dados estáticos e respectivos pormenores.

Actividades São suportadas as actividades habituais, nomeadamente *"action state"* (apelidada de "simples"), *"join"*, *"fork"*, "inicial", "final", "decisão"/"junção" e "composta". Esta última representa uma actividade que está ela própria definida usando outro diagrama de actividades, correspondendo portanto a outro *workflow*. Suporte para a actividade *"flow final"*, será adicionado posteriormente, uma vez esta fazer parte apenas de padrões de controlo mais avançados. De momento, e para restringir a quantidade de casos iniciais a cobrir, existem algumas "obrigatoriedades" no que toca ao uso das actividades. Destas destacam-se : só poder existir uma actividade final; um *fork* será sempre acompanhado de um *join*; não se processam *forks*/*joins* dentro dos mesmos.

Eventos ou Acções Cada actividade "simples" possui um evento, que é despoletado quando o controlo de fluxo atinge a actividade em questão. Dos tipos existentes de eventos, actualmente apenas é suportado o *"call event"*, responsável pela chamada de uma "função". Posteriormente será introduzido o *"time event"*, que introduzirá uma restrição temporal na execução de determinada actividade. Apenas depois será pensada a inclusão dos restantes tipos de acção (*"signal event"* e *"change event"*), não sendo relevantes de momento.

Transições Cada transição necessita de um nó de partida e um nó de chegada. Como apenas se deseja que as transições ligadas a nós decisores tenham condições de guarda (ver descrição das "condições"), definem-se dois tipos de transição: transição "condicional", possuindo a sua condição de guarda, e a simples transição "incondicional".

Condições Em teoria, cada actividade de decisão possui uma condição, responsável pela escolha do caminho a tomar pelo controlo de fluxo. No entanto, isto implicaria uma estrutura dinâmica para manter as diversas hipóteses de fluxo no nó decisor. Como tal, opta por ter a decisão um passo à frente, isto é, nas transições que partem do nó decisor. Desta forma, aquando da chegada do controlo de fluxo a uma actividade de decisão, todas as condições das transições que dele partem são avaliadas, sendo escolhida a primeira condição avaliada positivamente. Com esta abordagem consegue-se esperar sempre um resultado booleano (verdadeiro ou falso) da avaliação das condições de cada transição, o que facilita o percurso do *workflow*. No presente, esta condição apenas pode ser uma comparação simples, como se vê no exemplo da figura 3, onde se testa apenas o resultado anterior como *true* ou *false*, mas será brevemente incluída a capacidade de realizar pequenas expressões, embora sempre com resultados booleanos.

Workflows Identificadores e nomes de todos os modelos de *workflow* carregados no sistema.

Blocos Um "bloco" é um percurso dentro de um par *fork/join*. Desta forma todas as actividades dentro de um par *fork/join* têm obrigatoriamente que fazer parte de um bloco, e um apenas. Este mecanismo serve para garantir que há isolamento de dados entre os diversos blocos de uma execução paralela, como é o caso da execução a partir de um *fork*. Cada bloco possui um identificador único e a referencia às actividades *fork* e *join* que o englobam. A ligação entre as actividades e os blocos é feita através de outra relação, contendo a referência ao bloco, à actividade e ao *workflow*.

Ao contrário de toda a outra informação estática, os blocos são calculados após a análise do XMI, já sobre a informação mantida relativa do modelo.

São apresentados em seguida os tipos de informação dinâmica.

Execução Uma "execução" é uma instância de um *workflow*. Cada instância possui o identificador do *workflow* a que se refere, uma *flag* de estado de execução (informando se a instância está activa ou já terminou), uma lista de actividades a serem executadas (para auxiliar na execução paralela) e o "traço" da execução, que permitirá saber exactamente o percurso de execução efectuado.

Instâncias de eventos Um evento é despoletado quando o controlo de fluxo chega a uma actividade que o contém. O evento é instanciado para conter informação específica sobre a sua invocação, nomeadamente a execução activa, o bloco que contém a actividade (caso exista) e o resultado da chamada (ver 3.2). Isto é particularmente útil para manter históricos de resultados.

Buffer de entrada Esta relação será responsável por armazenar temporariamente os resultados das chamadas. Este processo é explicado a seguir. É criada uma instância de evento, com o resultado da chamada a *"null"* e é realizada a chamada. Nesta chamada é passada a informação necessária para que a "função" chamada saiba onde colocar a resposta, ou seja, toda a informação necessária para criar um registo no "buffer de entrada".

Um *daemon* externo ao motor de execução de *workflows* vai ser responsável por ver quando a resposta da chamada chega ao buffer e colocá-la no local adequado na relação de instâncias de eventos. Só depois de ter um resultado pode um evento ser considerado acabado, e a actividade que o inclui sinalizada como completa.

3.3 Regras de modelação necessárias

Ao deixar um utilizador usar uma ferramenta completa de modelação UML como o ArgoUML, existe uma boa probabilidade de este vir a utilizar erroneamente alguma das opções de modelação. Como tal existe a necessidade de estabelecer algumas regras no que toca à modelação dos diagramas de actividade, para que a sua "tradução" ocorra sem perdas ou falhas. Essas regras são[3]:

- um nó do tipo *action state* terá exactamente uma acção, que produzirá exactamente um resultado;
- nós do tipo *join* e decisão/junção têm **sempre** um resultado;
- ambos os resultado mencionados anteriormente têm uma duração de apenas uma transição, isto é, apenas o próximo nó (seja ele qual for) pode usar esse resultado;
- o resultado pode ser utilizado fazendo referência ao nome estático "Result" dentro do campo "expression" de uma acção (evento);

[3] Estas regras são as estabelecidas no momento da redacção deste documento, pelo que é natural que o seu número venha a ser aumentado com o suporte de outros padrões de controlo

– é excepção à regra anterior um resultado que seja definido como não-volátil, utilizando o estereotipo "non-ephemeral", podendo ser acedido a qualquer altura dentro do *workflow*;
– as acções dos nós do tipo *action state* são chamadas sob a forma de predicados comuns ISCO, que serão necessariamente codificadas externamente[4] ao ArgoUML;

3.4 Organização em "Units"

Dado que havia a noção clara de estar a trabalhar com partes de código com funções muito especificas, optou-se por organizar o mesmo utilizando os mecanismos de estruturação de programas oferecidos pela programação em lógica contextual [5]. Cada módulo é assim uma *unit* com funções específicas. A organização do código é a seguinte:

Main Esta *unit* é o ponto de união entre todas as outras. Este módulo serve de *router* de comunicação entre os outros, sendo que não há troca directa de informação. Isto ajuda à modularidade e mantém as comunicações mais simples.

PreProcess Este é o módulo responsável pela computação dos "blocos" dentro dos pares *fork-join*.

Process É nesta *unit* que são programadas as acções chamadas dentro do *workflow*. São-lhe passados os seguintes argumentos: identificador da instância de evento; identificador da instância do *workflow*; identificador do bloco (caso exista).

Model Nesta *unit* estão contidos todos os dados referentes a um modelo de *workflow*. Esta *unit* será o resultado directo do XMI processado pelo XSLT.

InterfacePHP Este módulo recebe todos os pedidos provenientes do *browser* e é responsável também pelas respostas.

DataReadWrite Esta *unit* permite ler e escrever na base de dados.

Engine Este módulo é responsável pela execução das instâncias dos *workflows*.

[4] A necessidade da especificação externa das acções prende-se com o facto de ser extremamente difícil garantir que quem faz a modelação consiga programar de facto as acções. Por outro lado, seria necessário estabelecer uma gramática rígida e específica para que as acções pudessem ser inseridas durante a modelação. Desta forma, tendo a codificação fora da modelação permite-se que ambas sejam definidas por utilizadores diferentes, e em tempos diferentes.

3.5 Arquitectura global

A figura4 apresenta o esquema da arquitectura global da aplicação contendo todas as partes que vieram a ser descritas.

Esta arquitectura permite ao utilizador inserir um modelo realizado num editor externo, desde que este possua uma forma de exportar os modelos sob a forma de XMI, bem como construir a modelação completamente a partir de um *browser*, directamente na aplicação. Essa mesma interface permite-lhe ter o controlo de gestão de fluxo do *workflow*.

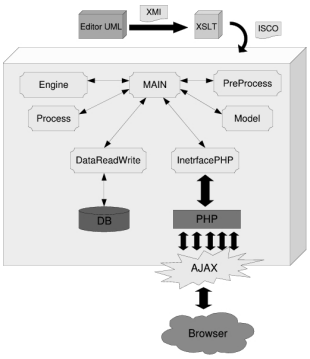

Fig. 4. Esquema da arquitectura global da aplicação

4 Resultados

A seguir são apresentadas algumas métricas caracterizando o que acontece ao longo das diversas fases desde o editor ArgoUML até se ter a representação do modelo em ISCO.

Todo o XMI do ArgoUML é construido fazendo encapsulamento dos componentes em blocos com identificadores, que por sua vez são referenciados noutros blocos. Estas referências cruzadas de blocos ao longo do XMI faz com que sejam necessários mais *templates* no XSLT de modo a conseguir realizar os percursos necessários ao longo XMI. Estes *templates* podem conter ciclos que pesquisam todos os nós de determinado nível da árvore de XMI de modo a conseguir encontrar as informações que procuram, o que aumenta a complexidade de pesquisa para quadrática, sobre o número nós.

A tabela seguinte relaciona o tipo de componente com as médias da quantidade de linhas geradas, linhas de XSLT usado para tratar cada um deles, chamadas a *templates* e ciclos que entram na pesquisa.

Componente	Linhas XMI	Linhas XSLT	N Templates	Ciclos
Join/Fork/Split/Merge	7+In+Out[5]	35	2	1
Act. inicial/final	6	35	2	1
Act. simples	17	63	5	3
Transição simples	8	19	2	1
Transição condicional	20	69	7	3

Para o código ISCO gerado, pegando no exemplo 3, temos aproximadamente 60 linhas de código geradas, com o seguinte aspecto:

```
fact(workflow(w1)).
fact(event_call(ev2,
                fazer_plano_de_arquitectura,
                process(fazer_plano_de_arq,_,_,_))).
...
```

```
fact(activity_start(start, w1, 'Actividade inicial')).
fact(activity_simple(act1, w1, 'Escolher o terreno', ev1)).
...
fact(activity_join(join1, w1, 'Join de todos os planos', 3, 3)).
fact(activity_decision(decision1, w1, 'Decisao sobre os planos')).
...
fact(transition_unconditional(trans1, act1, start)).
...
fact(condition(cond1, (XPTO == true), variable(XPTO))).
fact(condition(cond2, (XPTO == false), variable(XPTO)))
```

Com este código temos modelo pronto a ser interpretado pela aplicação.

5 Comparação com trabalho relacionado

É seguidamente apresentada uma pequena comparação de alguns pontos entre a nossa aplicação (APP) e o sistema de gestão de *workflows OpenFlow* (OF).

OF - A definição das condições das transições é realizada na linguagem TAL (Template Attribute Language), como por exemplo python:instance.some_property=='value'. Os parâmetros da condição são normalmente variáveis de instância.

APP - A definição das condições é realizada sob a forma de OCL. Por omissão qualquer condição é avaliada contra o resultado da actividade anterior. Pode-se no entanto explicitar que o resultado de uma actividade é persistente, podendo ser avaliado em qualquer local do *workflow*, dentro do âmbito da execução.

OF - A definição de eventos (aplications) é feita separadamente, sendo a sua chamada feita a partir da referência a um URL dentro do Zope. Desse URL constará uma qualquer função aplicacional desejada.

APP - A definição dos eventos é feita no sistema, num menu especificamente designado para a função e é implementada em Isco (goals). Os eventos são invocados através da chamada do seu goal.

OF - Falta de capacidade de visualização do modelo de workflow, quer em tempo tempo de definição quer em tempo de execução. Pode-se no entanto usar o OpenFlowEditor para obter visualização e alguma capacidade limitada de interação com o mesmo (criação/remoção de actividades/transições e atribuição de tarefas a utilizadores);

APP - Tudo é feito graficamente sobre o modelo do workflow. A sua construção é feita arrastando e ligando os diversos componentes, e preenchendo os menus de propriedades contextuais. A execução assinala em tempo real o que se passa numa instância do utilizador.

OF - Interface complexa (comandada pelo estilo "industrial" do ZOPE).

APP - Interface simples e intuitiva, altamente reactiva às acções do utilizador. Este artigo não apresenta nenhum *screenshot* da interface porque esta ainda se encontra numa fase de testes, em que se estão a avaliar quais as bibliotecas AJAX a usar.

OF - Exportação dos *workflows* em formato ZEXP.

APP - Exportação dos *workflows* em formato XMI, com formato semelhante ao que pode ser importado.

OF - O *OpenFlow* possui vários *roles*, correspondendo os mesmos aos vários utilizadores intervenientes no desenrolar do *workflow*. As actividades podem ser atribuidas a um utilizador, sendo que este, ao fazer *login*, pode visualizar quais as actividades que tem pendentes.

APP - A noção de utilizadores e grupos de utilização será implementada brevemente.

6 Conclusões e Trabalho Futuro

Para se poder usar o editor integrado em vez do editor externo de UML, é necessário chegar a algumas conclusões sobre o que se deseja do próprio editor. Antes de mais é necessário decidir se nos mantemos puristas, usando unicamente UML nas modelações, ou se disponibilizamos uma ferramenta mais específica e funcionalmente adaptada aos *workflows*. A conclusão é que uma mistura dos dois é o mais adequado. Temos assim uma base sólida de UML, juntamente com flexibilidade suficiente para chegar a padrões de fluxo para os quais a UML não oferece solução. Do ponto de vista de utilização, faz muito mais sentido ter o controlo completo de gestão e edição dentro do mesmo ambiente. Do ponto de vista técnico as vantagens de ter este controlo são bastantes, especialmente porque temos interacção directa com o motor de execução, o que permite realizar alterações em tempo real sobre modelos que já possuam instâncias em execução (em semelhança do que acontece com o *OpenFlow*).

O facto desta aplicação estar baseada em ISCO permite tirar partido das capacidades declarativas para implementar usos "inteligentes" usando pouco mais do que uma chamada de um *goal* ISCO. Exemplo disso são as contas de utilizadores e a respectiva construção de *task lists* relativa a tarefas em instâncias de *workflows*.

São seguidamente apresentados mais dois dos usos que podem ser dados às propriedades declarativas do ISCO para chegar a resultados benéficos ao utilizador.

Construção assistida de workflows Esta ideia baseia-se no pressuposto de colocar a aplicação a ajudar em tempo real o utilizador enquanto este modela o seu *workflow*, através da apresentação de pista visuais e mesmo técnicas sobre as opções a seguir na modelação. Poder-se-à ter inserção directa de padrões de controlo ou mesmo avaliação sobre o modelo de modo a sugerir reorganizações do mesmo. Outro modo de construção assistida poderá ser a simulação automática de execução com base, por exemplo, em resultados pré-estabelecidos das condições ou conjuntos de variação de resultados. Isto permitirá avaliar os diferentes percursos de fluxo e detectar erros de execução.

Google de workflows Baseada num motor de análise de "analogias" de modelos, esta ideia permitiria elaborar buscas sobre "sinónimos" de um modelo ou porç porções do mesmo, apresentando como resultados os modelos que fossem funcional e/ou modularmente idênticos.

References

1. Openflow homepage. http://www.openflow.it/EN/index_html. Homepage.
2. Xml metadata interchange (xmi), v2.1. http://www.omg.org/cgi-bin/doc?formal/2005-09-01. XMI specification.
3. Xslt elements reference. http://www.w3schools.com/xsl/xsl_w3celementref.asp.
4. Xslt reference. http://www.zvon.org/xxl/XSLTreference/Output/index.html.
5. Salvador Abreu and Daniel Diaz. Objective: in Minimum Context. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, pages 128–147. Springer-Verlag, 2003. ISBN 3-540-20642-6.
6. Salvador Abreu and Vítor Nogueira. Using a Logic Programming Language with Persistence and Contexts. In Masanobu Umeda and Armin Wolf, editors, *Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2005)*, Fukuoka, Japan, October 2005. Waseda University.
7. Wendy Boggs and Michael Boggs. *Mastering UML with Rational Rose 2002*. Sybex, 2002.
8. Kevin Breit, Henry House, and Judith Samson. Dia. http://www.gnome.org/projects/dia/doc/dia-manual.pdf. User's Manual.
9. Marlon Dumas and Arthur H.M. ter Hofstede. Uml activity diagrams as a workflow specification language. In *Proceedings of the UML'2001 Conference*, 2001.
10. Ricardo Pereira e Silva. Automatic code generation from uml models.
11. Veronica Fredriksen. Wide gap amongst developers' perception of the importance of uml tools, developereye study reveals. *Express Press Release*, 2005.
12. Jesse James Garrett. Ajax: A new approach to web applications. February 2005. Seminal.
13. Paul Hensgen. Umbrello uml modeller handbook. http://docs.kde.org/stable/en_GB/kdesdk/umbrello/index.html.
14. Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp. Argouml user manual - a tutorial and reference description. http://argouml-stats.tigris.org/documentation/manual-0.22/.
15. Will M.P. van der Aalst, Alistair H.M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns site. http://is.tm.tue.nl/research/patterns/flash_animations.htm, 2001.
16. Bartek Kiepuszewski Will M.P. van der Aalst, Alistair H.M. ter Hofstede and Alistair P. Barros. Workflow patterns. Technical report, Queensland University of Technology, 2002.
17. van der Aalst W. Dumas M. ter Hofstede A. Wohed, P. and N Russell. Pattern-based analysis of uml activity diagrams. In *Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)*, Klagenfurt, Austria. Springer.

XML::TMX– Processamento de Memórias de Tradução de Grandes Dimensões

José João Almeida and Alberto Manuel Simões

Departamento de Informática
Universidade do Minho
{jj|ambs}@di.uminho.pt

Resumo As ferramentas de tradução assistida por computador tentam reutilizar as traduções realizadas pelo tradutor sempre que uma frase semelhante tenha sido já traduzida. Para o intercâmbio destes documentos foi definido um formato denominado TMX (Translation Memory Exchange) baseado em XML.
Este tipo de documento ganha facilmente tamanhos incomportáveis para o seu processamento com métodos tradicionais.
Neste artigo propomos uma metodologia de ordem superior para o processamento de documentos de estrutura repetitiva (em que se inserem as memórias de tradução) com uma abordagem baseada na conjunção de SAX e DOM.
São apresentados vários exemplos de filtros sobre memórias de tradução bem como um conjunto de medidas da sua eficiência.

1 Introdução

A tradução quando feita em ambiente profissional tira proveito de ferramentas de tradução assistida por computador (CAT). Estas ferramentas armazenam as traduções já realizadas em projectos anteriores em bases de dados específicas (memórias de tradução).

Dum modo simplificado as memórias de tradução são grande conjuntos de unidades traduzidas (designadas por unidades de tradução – normalmente frases) e opcionalmente algumas propriedades e anotações associadas.

1.1 Processo de Tradução

A tradução baseada em memórias de tradução tem como principal objectivo a reutilização de traduções já realizadas pelo tradutor. Uma memória de tradução não é mais do que uma base de dados de segmentos traduzidos (unidades de tradução) que permitem ao tradutor[4,6]:

- propagar no texto de destino as traduções de frases que se repetem no texto original;
- reciclar traduções que foram realizadas ao trabalhar noutros textos, reutilizando-as tal como armazenadas na memória de tradução ou depois de alteradas;

– analisar um novo texto original e encontrar *matches* com traduções já armazenadas na memória de tradução, permitindo desta forma reutilizar porções de traduções já realizadas;

O processo de tradução usando memórias de tradução realiza-se da seguinte forma:

1. o programa divide o texto original em segmentos. Esta divisão é feita tendo em conta a pontuação da língua em causa, e a marcação do formato específico em que o documento se encontra;
2. a tradução é realizada para cada segmento do texto de origem pela sua ordem natural, de acordo com os seguintes passos:
   (a) o programa verifica se o próximo segmento a ser traduzido está na memória de tradução, ou se algum segmento razoavelmente semelhante já foi traduzido;
   (b) o tradutor determina se vai usar, editar ou ignorar a tradução que o programa encontrou;
   (c) o programa guarda o segmento da língua de origem e a respectiva tradução na memória de tradução;

São exemplos de ferramentas de tradução assistida por computador o Trados [9], STAR Transit [13], Déjà vu [2] e mesmo o sistema colaborativo de tradução TransBey [3].

1.2 Memórias de Tradução: TMX

O formato TMX [7] foi definido pela associação LISA (The Localization Industry Standards Association) como forma de partilhar memórias de tradução entre sistemas de tradução (que usam internamente formatos proprietários).

Sendo um formato de intercâmbio, é fortemente estruturado. No entanto, bastante simples. É composto por duas partes principais: um cabeçalho de meta informação, e um corpo. Esta segunda parte é a principal destes documentos, composta por pequenas entradas de unidades de tradução.

Segue-se um pequeno exemplo:

```
1   <?xml version='1.0' encoding='ISO-8859-1'?>
2  <!DOCTYPE tmx SYSTEM "tmx11.dtd">
3  <tmx version="version 1.0">
4  <header
5          creationtool="cwb-utils"
6          creationtoolversion="1.0"
7          segtype="sentence"
8          adminlang="EN-US"
9          srclang="fr"
10         o-tmf="CQP-corpora"
11 >
12 </header>
13 <body>
```

```
14 <tu>
15  <tuv lang='pt'>
16     <seg>Praticamente ausente dos mapas de fluxo de dados, a África
17     não contabiliza mais linhas telefónicas do que Tóquio ou Manhattan,
18     nem mais computadores ligados à Internet do que a Lituânia.</seg>
19  </tuv>
20  <tuv lang='fr'>
21     <seg>Quasi absente des cartes de flux de données, l'Afrique ne
22     compte pas plus de lignes téléphoniques que Tokyo ou Manhattan, pas
23     plus d'ordinateurs connectés à Internet que la Lituanie.</seg>
24  </tuv>
25 </tu>

26 <tu>
27  <tuv lang='pt'>
28     <seg>Todavia, o continente não escapa às transformações nas
29     telecomunicações, onde se lêem, mais do que em qualquer outro sítio,
30     as recomposições inéditas impostas pela mundialização.</seg>
31  </tuv>
32  <tuv lang='fr'>
33     <seg>Pourtant, le continent n'échappe pas au bouleversement des
34     télécommunications, dans lequel se donnent à lire, là plus
35     qu'ailleurs, les recompositions inédites qu'impose la
36     mondialisation.</seg>
37  </tuv>
38 </tu>

39 </body>
40 </tmx>
```

2 Processamento de TMX de Grandes Dimensões

Embora seja um formato bastante simples, é muito habitual os documentos TMX chegarem às centenas de MegaBytes. Deste modo, é praticamente impossível processar estes documentos em memória da forma convencional. Uma árvore do DOM (Document Object Model) de um destes documentos ocupa cerca de 15% mais de memória do que o documento original só em estruturas de dados.

Por outro lado, o outro modelo de processamento habitual baseado em SAX não é suficientemente versátil.

Surge então a necessidade de analisar uma metodologia que nos permita por um lado processar documentos de grandes dimensões, e por outro, garantir alguma versatilidade. Para isso decidiu-se processar o corpo da TMX de uma forma incremental, por blocos. Cada memória de tradução pode ser processada independentemente da outra. Por outro lado, cada memória de tradução é um documento XML válido por si só, o que permite que se use um parser para criar o DOM desse bloco.

A implementação deste algoritmo no nosso módulo XML::TMX[10] é bastante simples. A linguagem Perl permite a definição de um separador de registo, separador esse que é usado pelos métodos de leitura de ficheiros (ou *standard input*), para ler porções (ou *chunks*) de informação. Desta forma, é possível definir o separador de registo como sendo a etiqueta de fecho de uma unidade de tradução (`</tu>`) o que permite que cada chunk (com excepção do primeiro e do último) seja uma unidade de tradução completa.

```
<tu>
 <tuv lang='pt'>
    <seg>Praticamente ausente dos mapas de fluxo de dados, a África
    não contabiliza mais linhas telefónicas do que Tóquio ou Manhattan,
    nem mais computadores ligados à Internet do que a Lituânia.</seg>
 </tuv>
 <tuv lang='fr'>
    <seg>Quasi absente des cartes de flux de données, l'Afrique ne
    compte pas plus de lignes téléphoniques que Tokyo ou Manhattan, pas
    plus d'ordinateurs connectés à Internet que la Lituanie.</seg>
 </tuv>
</tu>
<tu>
```

Cada um destes extractos é por si só um documento XML válido (com a excepção da definição de XML) que pode ser interpretado usando o XML::DT [1].

Embora esta abordagem obrigue à inicialização de um *parser* XML para cada uma das unidades de tradução, é escalável: não é necessária a criação da árvore de DOM em memória.

Esta abordagem, embora neste caso específica para as memórias de tradução é facilmente generalizável para outros tipos de documentos como sejam os de anotação de terminologia dicionários TEI e SKOS, subconjuntos de OWL, e outros.

3 Processamento de ordem Superior

Esta secção inicia com uma introdução da abordagem na programação de sistemas de parsing de ordem superior [5], seguida de um conjunto de exemplos da utilidade desta metodologia de programação.

3.1 Abordagem

Dado que o processamento das memórias de tradução não é eficiente como um todo, e por outro lado, ao processar cada unidade de tradução é indiferente a sua vizinhança (não há noção de ordem de memórias de tradução), é possível definir uma função de processamento de memórias de tradução que receba como parâmetro a função que processa cada memória de tradução.

Para além do resultado da função de processamento da unidade de tradução em causa, a função de ordem superior **for_tu**, permite:

- **transformar as unidades de tradução:** $tu \rightarrow tu|0$
  a função de processamento **for_tu** funciona como um *map* funcional que aplica a cada unidade de tradução uma função de processamento que pode devolver a unidade de tradução depois de processada ou produzir efeitos laterais.
- **remover unidades de tradução:**
  no caso da função de processamento devolver um objecto vazio, a unidade é retirada da memória gerada.
- **adicionar e remover propriedades às unidades de tradução:**
  além do texto e respectiva tradução o *standard* TMX permite definir propriedades (etiqueta `prop`) e notas (etiqueta `note`) sobre cada unidade de tradução. A função de processamento recebe não só o texto correspondente à unidade de tradução mas também a lista de propriedades e de notas associadas. A função de processamento permite adicionar e remover propriedades e notas.
- **indicar o ficheiro de saída pretendido:**
  por omissão a função escreve a nova memória de tradução para o *standard output*. No entanto este comportamento pode ser alterado indicando o nome do ficheiro onde a nova memória deve ser escrita.
- **definir um número máximo de unidades de tradução a processar:**
  em algumas ferramentas, como as que funcionam sobre a web, é importante limitar o número de unidades de tradução a processar de forma a aliviar o processamento. Este número pode ser definido ao invocar a função **for_tu**.
- **definir o número máximo de unidades de tradução a escrever:**
  funciona de forma semelhante à anterior, mas em vez de limitar o número de unidades de tradução a processar limita o número de resultados: processa unidades de tradução até que seja retornado o número de unidades de tradução escolhido.
- **indicar um padrão de activação:**
  permite especificar uma expressão regular de pesquisa, de forma a que apenas as unidades de tradução que façam *matching* sejam executadas.

3.2 Leitura de TMX: contar as TUs

Este exemplo trivial tenciona mostrar o quão simples é escrever um processador de memórias de tradução. Neste processador faz-se uma travessia de uma memória de tradução com a finalidade de contar o número de unidades de tradução existentes.

```
1  use  XML::TMX::Reader;
2  my $mem = XML::TMX::Reader->new('sample.tmx');

3  my $count = 0;
4  $mem->for_tu(
5      sub { $count++; }
6    );

7  print $count;
```

linha 1: é carregado o módulo para leitura de TMX;
linha 2: é criado um objecto com a TMX em causa;
linha 4: o método **for_tu** itera todas as memórias de tradução;
linha 5: a função de processamento da memória de tradução limita-se a contar a existência de mais uma ocorrência.

3.3 Transformação de Memórias de Tradução

Considere-se o seguinte processador genérico de memórias de tradução:

```
1  my $mem->for_tu(
2     { output => "nova.tmx",
3       gen_tu => 1000 },
4      \&proc )
```

linha 1: iterar as unidades de tradução;
linha 2: colocar o resultado do processamento no ficheiro "nova.tmx";
linha 3: o processamento termina após produzir 1000 unidades de tradução;
linha 4: a função de processamento a utilizar chama-se **proc**.

Seguem-se várias funções de processamento (**proc**) para o tratamento de unidades de tradução.

Remoção de unidades de tradução Ao criar memórias de tradução automaticamente é habitual existirem maus alinhamentos (unidades de tradução cujo texto não corresponde à tradução correcta).

Se uma unidade de tradução tiver segmentos com tamanhos muito díspares é do nosso interesse removê-la. Neste exemplo considera-se que as línguas em causa são a língua inglesa (en) e a portuguesa (pt).

```
1     sub proc {
2       my $tu = shift;
3       if (length($tu->{en}) > 2 * length($tu->{pt})
4            { return undef }
5       else { return $tu ; }
6     }
```

linha 2: colocar em `$tu` a unidade de tradução;
linha 3: calcular o tamanho do segmento inglês e português (acedidos com `$tu->{en}` e `$tu->{pt}`) e devolver indefinido se o primeiro tiver mais de duas vezes o tamanho do segundo;
linha 5: devolver a unidade de tradução para ser adicionada ao resultado gerado.

Adicionar propriedades Como foi referido anteriormente, cada unidade de tradução pode ter a si associado um conjunto de propriedades e notas. A seguinte função de processamento adiciona uma medida de qualidade de tradução e um domínio.

```
1    sub proc {
2      my $tu = shift;
3      $tu->{-prop}{quality} = calculate_quality($tu->{en},$tu->{pt});
4      $tu->{-prop}{domain}  = infer_domain($tu->{pt});
5      return $tu;
6    }
```

3.4 Limpeza de TMXs

Em todo os trabalhos envolvendo extracção de informação a partir de TMX há naturalmente uma grande dependência da qualidade das unidades de tradução da TMX de partida. Neste exemplo a estratégia usada passa pela contagem e comparação de elementos não textuais (ex, números) que permanecem iguais nas duas línguas: serão rejeitadas as unidades que difiram muito nos elementos não textuais, bem como as que tenham tamanhos excessivamente diferentes. Por último, são rejeitadas as memórias de tradução (inteiras) quando a taxa de unidades de tradução rejeitadas for elevada.

```
1  Para todas a TMX f fazer:
2     perfeitas = 0                 # contador de TU muito boas
3     péssimas = 0                  # contador de TU muito más
4     numbers_pt = {}               # números na parte portuguesa
5     numbers_en = {};              # números na parte inglesa

6     nova.tmx = limpa(f.tmx)
7     se (numbers_pt ≠ numbers_en)
8         apaga(nova.tmx)
9     se (taxa(péssimas) é elevada  )
10        apaga(nova.tmx)
```

onde `novaf.tmx = limpa(f.tmx)` corresponde a:

```
1  $mem->for_tu(
2    {output => "novaf.tmx"} ,
3    \&proctu
4  )
```

sendo a função de processamento das TUs `proctu` acima referida, definida como:

```
1  sub proctu {
2    my $tu = shift;
3    $tu->{pt} = retoca($tu->{pt});
4    $tu->{en} = retoca($tu->{en});

5    ...extrai números pt en
6    ...perfeita  se   números pt = números en ≠ {}
7    ...péssima   se   números pt muito diferentes números en
8    ...péssima   se   tamanhos muito diferentes
9    confiança = f(....)

10   if (péssima) { return undef }
11   $tu->{prop}{conf} = confiança
12   return $tu;
13 }
```

3.5 Remoção de entradas duplicadas

Por vezes ao criar e juntar memórias de tradução acabam por existir unidades de tradução repetidas. Este exemplo mostra uma forma rápida de as remover, usando para isso o valor MD5 da unidade de tradução.

```
1  tie %dic, 'DB_File', "mydbfile.db",
2      O_RDWR|O_CREAT|O_TRUNC , 0640, $DB_BTREE;

3  my $tm = XML::TMX::Reader->new($filename);

4  $tm->for_tu(
5      sub {
6          my $tu = shift;
7          my $digest = md5_hex("$tu->{en},$tu->{pt}");

8          if ($dic{$digest}) {
9                 return undef
10         } else {
11                $dic{$digest} = 1;
12                return {%$tu} ;
13         }
14     }
15 );
```

linha 1: criar uma base de dados de valores MD5 para consulta rápida;
linha 5: iterar todas as memórias de tradução;
linha 7: calcular o valor MD5 da memória de tradução;

linha 8: se o valor MD5 está na base de dados, a unidade é repetida, e portanto, ignorar;

linha 10: se o valor não existe, guardar na base de dados e devolver a unidade de tradução.

4 Considerações referentes a desempenho

A tabela 1 mostra uma comparação de tempos[1] da função **for_tu** implementada com base na construção da árvore DOM e com base no processamento por chunks. Foi construída uma função que apenas conta o número de unidades de tradução (ver secção 3.2) e aplicada a memórias de tradução com diferentes tamanhos.

Note-se que enquanto é possível guardar todo o DOM em memória esta abordagem é mais eficiente. No entanto, assim que deixa de caber em memória passa a não ser exequível (no nosso ambiente de teste a memória de tradução EurLex congela o computador).

Por outro lado, a abordagem por chunks tem um crescimento linear. Embora possa demorar mais tempo consegue dar uma resposta. Note-se que é normal utilizar memórias de tradução com mais de um milhão de unidades de tradução.

			DOM		Chunks	
	TUs	Tamanho	tempo	memória	tempo	memória
NATO	53 562	18 MB	38s	108 MB	50s	10 MB
LMD	68 231	25 MB	41s	145 MB	61s	10 MB
PE1	382 581	83 MB	230s	637 MB	343s	10 MB
EurLex	1 110 163	353 MB	—	—	1003s	10 MB

Tabela 1. Tempos de parsing simples.

Considerando a medida de um exemplo mais complexo, a remoção de unidades de tradução repetidas (ver secção 3.5) numa memória de tradução com 1 784 164 unidades (560MB) demora cerca de 35 minutos e 25 segundos (removendo 714 837 unidades repetidas).

5 Conclusões

As memórias de tradução são recursos bastante úteis, quer para tradutores, linguistas ou investigadores de processamento de linguagem natural (por exemplo, na tradução automática [8]). Deste modo, é importante que existam métodos eficientes para o seu processamento. Embora existam ferramentas que utilizam estas memórias de tradução para ajudar o tradutor no seu trabalho, elas tratam

[1] As medidas apresentadas nesta secção foram obtidas num Pentium IV 3.2GHz, com 2GB de RAM, Linux.

apenas desta utilização específica das memórias de tradução. Há uma grande necessidade e utilidade em dispor de um ambiente de programação versátil e aberto para o tratamento generalizado de memórias de tradução.

A resolução do problema da disponibilização eficiente de memórias de tradução já foi discutida em [12,11], utilizando uma arquitectura distribuída de servidores dedicados. No entanto, esta abordagem não facilita o tratamento preliminar das memórias de tradução.

A abordagem apresentada tem a vantagem de ser escalável a memórias de tradução de grandes dimensões, sem que o seu tamanho influencie a memória necessária para o seu processamento: o tempo cresce linearmente em relação ao número de unidades de tradução.

O facto de se ter optado pelo uso de funções de ordem superior permite que com pouco esforço de programação se façam filtros ou apenas travessias a memórias de tradução. A própria facilidade do uso de módulos externos (como o exemplo da secção 3.5) é crucial no desenvolvimento de ferramentas.

Acreditamos que a metodologia mista de uso SAX (escalabilidade) e DOM (expressividade) é aplicável a uma grande variedade de outro tipos de documentos. Esta abordagem só traz resultados vantajosos quando os documentos têm estrutura repetitiva (nas outras situações funciona mas não tem vantagens) o que acontece na generalidade do uso de XML em documentos de grandes dimensões.

Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação para a Ciência e Tecnologia of Portugal pela bolsa POSI/PLP/43931/2001, e co-financiado pelo POSI, dentro da Linguateca.

Referências

1. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, May 1999.
2. ATRIL. Déjà vu home page and documentation. Translation tool, October 2006. `http://www.atril.com/`.
3. Youcef Bey, Christian Boitet, and Kyo Kageura. The TRANSBey prototype: an online collaborative wiki-based cat environment for volunteer translators. In *LREC-2006: Fifth International Conference on Language Resources and Evaluation. Third International Workshop on Language Resources for Translation Work, Research & Training (LR4Trans-III)*, pages 49–54, Genoa, Italy, 28 May 2006.
4. Claude Bédard. Mémoire de traduction cherche traducteur de phrases (translation memory is looking for sentences translator). *Traduire (Traduire) ISSN 0395-773X*, (186):41–49, 2000.
5. Mark Jason Dominus. *Higher Order Perl.* Morgan Kaufman, 2005.
6. Dorothy Kenny. Translation memories and parallel corpora: Challenges for the translation trainer. In *Inaugural Conference of the International Association for Translation and Intercultural Studies*, Sookmyung Women's University, Seoul, Korea, 12–14 August 2004.

7. Yves Savourel. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association, 1997.
8. Reinhardt Schaeler. Beyond translation memories. In Michael Carl and Andy Way, editors, *Workshop on Example-Based Machine Translation*, pages 49–55, September 2001.
9. SDL International. Sdl trados home page and documentation. Translation tool, October 2006. `http://www.trados.com/`.
10. Paulo Silva, Alberto Simões, and José João Almeida. XML::TMX – Perl extensions for managing TMX files. Perl module, Projecto Natura, Departamento de Informática, Universidade do Minho, 2003. `http://search.cpan.org/dist/XML-TMX-0.14/`.
11. Alberto Simões, José João Almeida, and Xavier Gomez Guinovart. Memórias de tradução distribuídas. In José Carlos Ramalho and Alberto Simões, editors, *XATA2004 - XML, Aplicações e Tecnologias Associadas*, pages 59–68, February 2004.
12. Alberto Simões, Xavier Gómez Guinovart, and José João Almeida. Distributed translation memories implementation using webservices. In *Sociedade Española para el Procesamiento del Lenguaje Natural*, pages 89–94, Jul. 2004.
13. STAR Group. Transit home page and documentation. Translation tool, October 2006. `http://www.star-group.net/`.

VoiceOverM^{2}L - Talk the Math

Daniel Silva, Pedro Abreu, Pedro Mendes, e Vasco Vinhas

Faculdade de Engenharia da Universidade do Porto Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal

Resumo Apesar da facilidade de acesso à informação e simbiótica disseminação, subsistem nichos com barreiras de acessibilidade. Existem igualmente áreas de conhecimento cujo léxico próprio tem necessidades expressivas actualmente sem resposta aceitável. Associando-se-lhes os nichos mencionados, assumem carácter particularmente relevante. A interpretação da informação matemática por invisuais ilustra esta problemática, emergindo a necessidade de ferramentas compatibilizadoras de apresentação e percepção. A sua realização passa pela vocalização da tradução do formato MathML para linguagem natural. Domínio privilegiado para a aplicação desta abordagem é a utilização dos *audio-browsers*, visto permitirem acesso a conteúdos disponibilizados na *web*. Assim, usufruindo das normas existentes, optou-se pela codificação da linguagem natural em SSML. A versatilidade da tecnologia torna-a adaptável a outras áreas, destacando-se ensino e investigação, com aplicações como leitura automática de enunciados, respectivos processos inversos e aprendizagem da simbologia matemática. Isto possibilita a generalização para sistemas de interacção por voz, reformulando paradigmas de interacção pessoa-computador.

1 Introdução

Nas últimas décadas, uma das consequências do desenvolvimento das redes de comunicação de dados foi, indubitavelmente, a proliferação de informação em formato digital. Todavia, se por um lado se verifica um crescimento massivo de conteúdos publicados em formato digital, semelhante evolução não se verifica no que diz respeito aos paradigmas utilizados pelos actuais terminais de acesso. Relativamente a estes, constata-se de facto que os paradigmas nos quais se baseiam se mantêm practicamente inalterados, não sendo mesmo de colocar de parte alguma regressão, se se considerar o abandono das interfaces em linguagem natural (LN) em meados da década de 90[1].

Deste contexto advêm claramente dificuldades acrescidas de acesso à informação por parte de indivíduos com necessidades especiais, pois as actuais interfaces não tomaram em conta, na sua concepção, requisitos básicos de usabilidade nestas condições. A *web*, meio privilegiado para a troca de informação e conteúdos, é um dos casos que melhor ilustra esta problemática. Na realidade, o W3C[2,3] desenvolveu um conjunto de regras de validação de código fonte de forma a proporcionar retrocompatibilidade com *audio-browsers* ou outros dispositivos semelhantes que procuram contornar, na sua maioria, dificuldades a nível

da percepção visual. No entanto, a grande maioria dos sítios *web*, a nível mundial, não segue as normas propostas, impossibilitando a navegação a utilizadores com necessidades especiais.

À medida que aumenta a distância à informação de domínio mais genérico, o problema adensa-se, nomeadamente em áreas de teor científico como as ciências naturais, sociais, engenharias, entre outras. Um ponto comum à grande maioria destas áreas é a matemática, daí que se justifique que uma abordagem em busca de respostas razoáveis a estas questões tenha esta como ponto de partida.

De entre as várias iniciativas existentes, na tentativa de uniformizar a informação contida em fórmulas escritas em notação lógico-matemática, geralmente opta-se pelo uso do MathML[4], em grande medida pelo recurso que este faz à tecnologia XML[5], facilmente interpretada por múltiplas plataformas das mais variadas origens [6].

No trabalho que se apresenta pretende-se fazer face às dificuldades previamente descritas, através de uma aplicação cujo objectivo principal é a transformação de simbologia matemática em formato MathML para LN, interpretável pelos vários dispositivos acima mencionados.

Pretende-se, igualmente, normalizar a vocalização das palavras geradas, tendo sido para este efeito adoptado o SSML[7] como formato padrão para a representação sonora.

Apesar do projecto parecer apenas estar vocacionado para o caso exposto, é também passível de uma adaptação a outras áreas, tais como apoio ao ensino e à investigação, nomeadamente através de aplicações para leitura automática de enunciados, digitalização de porções de manuais escolares ou aprendizagem da simbologia lógico-matemática.

É razoável admitir que no contexto actual de desenvolvimento, e aquando de uma generalização deste tipo de programas, estes poderão evoluir de forma a suportar funcionalidades inversas, tais como registo digital de fórmulas matemáticas tendo como origem a voz humana, reformulando assim todo o paradigma de interacção pessoa-computador.

O presente documento começa por apresentar a planificação delineada no sentido de desenvolver uma solução capaz de responder às problemáticas expostas. Na secção seguinte é realizado um estudo comparativo, de algumas ferramentas neste domínio. Na secção 3 são apresentados os dialectos anotados utilizados no projecto, cujos detalhes mais relevantes a nível da implementação dos processos são mencionados na secção 4. Por fim na secção 5 são discutidos os resultados obtidos, e apresentados possíveis desenvolvimentos futuros.

1.1 Plano

Numa primeira fase, foi contemplada a tradução de MathML em LN anotada. Esta fase envolveu o estudo e compreensão do formato MathML, assim como um estudo pouco aprofundado da língua portuguesa, por forma a gerar expressões correctas, tanto do ponto de vista matemático como do ponto de vista gramatical, sintático e semântico. Após esta primeira fase, o conteúdo produzido é já passível de ser interpretado por uma qualquer ferramenta de *text-to-speech*[8],

tendo mesmo sido incorporadas no projecto tais capacidades de vocalização do texto produzido.

Na fase posterior, foi contemplada a codificação da LN previamente produzida no dialecto SSML. Para tal, foi necessário o estudo do formato, nomeadamente da forma em como este é utilizado para a síntese de voz. Assim, foi possível gerar documentos em formato SSML, a partir da tradução anteriormente gerada, passíveis de serem interpretados pelas ferramentas adequadas.

Por último foi realizado um estudo linguístico por forma a adaptar o conteúdo produzido com as necessidades de percepção do ser humano. Foi igualmente desenvolvida uma interface gráfica que facilita o acesso às várias funcionalidades oferecidas pela ferramenta. De notar que o conteúdo SSML produzido segue as normas propostas pelo W3C para a sua correcta formação.

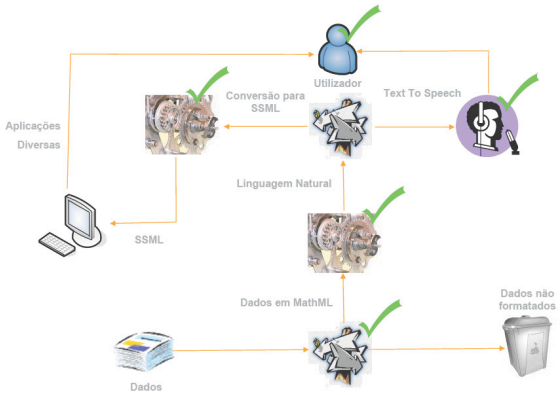

Figura 1. Diagrama Esquemático Representativo das Fases do Projecto

2 Estado da Arte

Existe actualmente uma panóplia de aplicações disponíveis, sendo que em todas elas se encontram aspectos positivos, muito embora na maioria dos casos estes sejam ofuscados pelas limitações inerentes. Verifica-se na maioria dos casos uma certa falta de autonomia, face à necessidade de bibliotecas, serviços do sistema operativo ou software específico.

Dos sistemas com capacidade de interpretar informação matemática, nem todos oferecem funcionalidades no sentido de vocalizar tais conteúdos, e, quando o fazem, o som produzido não é completamente perceptível. Dado que são sistemas orientados à LN, estão estritamente ligados ao idioma em causa, o que obviamente limita a correspondente divulgação noutros idiomas.

Do ponto de vista da introdução de dados, o universo do reconhecimento e processamento de LN não tem conhecido nos últimos anos uma evolução

apreciável, consequência da elevada carga de processamento necessária à obtenção de respostas a pedidos em tempo útil.

2.1 Exemplos Aplicacionais

No contexto exposto, descrevem-se, sucintamente, algumas ferramentas que procuram ilustrar os aspectos referidos.

AudioMath[24] Sendo disponibilizadas ao público apenas capacidades de tradução de páginas e fórmulas matemáticas em LN textual, o AudioMath é inovador na medida em que consegue melhorar a qualidade no que diz respeito à representação, flexibilidade, manipulação e acessibilidade de dados matemáticos, e no que diz respeito à navegação neste tipo de conteúdos, quando comparado com outras aplicações idênticas. Na sua sequência de processamento, começa por gerar uma versão textual do conteúdo matemático por ele interpretável, incluindo abreviaturas, numerais, acrónimos e referências de rede. A partir daí produz um fluxo de dados áudio correspondentes à vocalização do texto em Inglês [9]. Apesar de bastante completa, a aplicação carece de uma maior estruturação, a nível de conteúdos produzidos, na medida em que ao texto produzido não parece estar associada qualquer tipo de anotação normalizada ou controlo fonético/semântico.

Lambda Project[26] Projecto subsidiado pela União Europeia, com o intuito de tornar a matemática acessível a estudantes invisuais, o seu protótipo é actualmente disponibilizado sob a forma de um editor gráfico de fórmulas matemáticas com capacidades de vocalização das mesmas. No entanto a vocalização passa por chamadas a serviços externos e pré-existentes, não sendo garantida a validade lógico-matemática das expressões em causa.

MathPlayer[27] Disponibilizado sob a forma de um *plugin* para Microsoft®Internet Explorer (IE) 6.0 ou superior, revela desde logo uma falha: a sua não universalidade. Apesar do IE ser um dos *browsers* mais utilizados a nível mundial, representando cerca de 62,5% do total de utilizadores em Outubro de 2006, subsistem ainda os restantes 37,5%[17].

De salientar a impossibilidade de salvaguardar, em qualquer tipo de formato, a conversão efectuada aquando da vocalização.

Como funcionalidades chave, o MathPlayer apresenta as fórmulas graficamente, podendo o utilizador comutar entre dois níveis de ampliação do conteúdo, realizando ainda a vocalização, apesar da voz sintetizada ser pouco perceptível.

2.2 Conclusão

Como se pode verificar, as aplicações existentes estão ainda em fase de desenvolvimento, apresentando funcionalidades ainda limitadas, e por vezes complementares entre aplicações.

3 Dialectos

Neste ponto é realizada uma abordagem geral aos dois dialectos centrais da ferramenta desenvolvida e posteriormente expostas algumas particularidades consideradas relevantes para o projecto, sendo igualmente referidos alguns exemplos.

3.1 MathML

Coloca-se desde há muito o problema da representação digital e transmissão de informação referente a fórmulas matemáticas. Até ao aparecimento de outras soluções, tais dados eram transmitidos, na sua maioria, sob a forma de imagens, sendo impossível reaver o conteúdo original, ou ainda em formato textual, não possuindo a forma desejável ou uma legibilidade aceitável.

O MathML surge assim como uma forma de fornecer resposta à problemática descrita, capturando ao mesmo tempo forma e conteúdo, permitindo ainda a transmissão e edição da informação matemática. Trata-se de um dialecto bastante rico, contemplando um vasto leque de operadores lógicos e matemáticos, tornando-se assim extremamente flexível e adaptável, o que lhe valeu uma rápida divulgação e aceitação por parte da comunidade.

Por se focar numa área tão universal como a matemática, existe inerentemente um alargado conjunto de campos de aplicação nos quais o MathML surge como uma imprescindível ferramenta de suporte. Uma das áreas que mais se destaca é a do ensino e investigação, onde assume um papel vital, uma vez que a matemática é usada em variadíssimos campos, como a engenharia e a economia, não esquecendo a medicina e a saúde, ou mesmo as ciências sociais.

Exemplo de Aplicação

Browsers - Permitem a representação de fórmulas matemáticas, com interpretação via extensões ou *plugins*, tendo a vantagem de as fórmulas serem facilmente editáveis, e partilháveis na *web*.

Editores - Permitem a criação e edição de fórmulas matemáticas em formatos facilmente transponíveis e interpretáveis para outras plataformas. São vulgarmente integrados em ambientes de desenvolvimento de mais alto nível, com suporte à integração de DTDs [12], neste caso específico, o MathML.

Computação Científica - Permitem a exportação de resultados de cálculos de teor científico para formato XML facilmente integrável com outras aplicações.

Motores de Rendering - Permitem a transformação da informação formal matemática expressa no dialecto para formatos de imagem ou similar. Nesta gama de aplicações, encontram-se as bibliotecas que, mediante um ficheiro anotado em MathML, produzem um outro em formato de imagem do tipo gif, png, jpeg, bmp, tiff ou semelhante. São também enquadráveis neste âmbito projectos que se baseiam na conversão do formato MathML para SVG [23].

Conversores - Incluem-se neste caso aplicações que realizam a conversão do formato MathML para outros, não necessariamente em XML, nos quais se procura a informação inerente, sendo aceitável considerar que são formatos

concorrentes do dialecto, muito embora na sua generalidade com menor visibilidade ou aceitação, sendo talvez de ressalvar a excepção do LaTeX[10]. A título de exemplo, é de salientar um conjunto de ferramentas que realizam a conversão entre MathML e LaTeX, útil na produção de documentos formais.

Acessibilidade - Neste ponto são enquadráveis o conjunto de aplicações mencionadas anteriormente, nas quais se pretende facilitar o acesso à informação matemática por parte de indivíduos com necessidades especiais.

3.2 VoiceXML

O VoiceXML teve a sua origem em 1995, numa tentativa da AT&T [16] para desenvolver uma ferramenta facilitadora do reconhecimento da fala. Foi posteriormente desenvolvido, de forma a suportar diversas funcionalidades, como síntese e reconhecimento de voz, suporte a ficheiros de áudio, controlo de diálogo e de sessões telefónicas.

É maioritariamente utilizado pela indústria de telecomunicações no fornecimento de serviços informativos e automação de procedimentos de atendimento ao cliente [28]. É também frequentemente um auxiliador na relação entre humano e computador, especialmente por pessoas com necessidades especiais, permitindo um controlo operacional e aplicativo. No ramo da medicina é também utilizado na correcção de dificuldades na fala em crianças, especialmente problemas de dislexia [29], assim como no controlo de instrumentos de auxílio a cirurgia [30]. Para o projecto recorreu-se ao dialecto SSML, base do VoiceXML, pois este contém já todas as características relevantes para o âmbito pretendido.

Exemplo de Aplicação

Reconhecimento de Fala - Objectivo primário na base do desenvolvimento do VoiceXML, aplicações capazes de reconhecimento de fala, o mais independentemente possível de sotaques, ou variações entre indivíduos.

Síntese de Voz - Aplicações baseadas em VoiceXML desenvolvidas com o intuito de melhorar a qualidade de síntese de voz face aos tradicionais sistemas de *text-to-speech*, nomeadamente no que respeita a diferenças regionais de sotaque, entoações, pausas, e outros elementos meta-linguísticos, podendo ainda incluir ficheiros de áudio pré-gravados.

Controlo de Diálogo e Chamadas Telefónicas - Inserem-se aqui ferramentas utilizadas pela indústria de telecomunicações, para atendimento automatizado de clientes, simulando um operador humano, capaz de comunicar com o cliente o mais naturalmente possível.

3.3 Conclusão

Em suma verifica-se que existe já uma panóplia de aplicações que têm como base um dos dois dialectos apresentados. Todavia a diversidade de ferramentas disponíveis dirigidas ao MathML é superior à referente no outro dialecto apresentado, algo que se justifica com as características universais da matemática

aliadas às necessidades emergentes das várias áreas científicas em representar o seu conhecimento intrínseco de forma digital.

4 Aplicação

Nesta secção é descrita de forma mais detalhada a estrutura organizacional da ferramenta e expostos alguns dos seus aspectos críticos em termos da sua arquitectura interna.

4.1 Arquitectura

Na arquitectura modular adoptada para o desenvolvimento do projecto, são identificáveis um conjunto de três camadas distintas – Interface, Lógica de Negócio, Acessos a Ficheiros – conforme ilustrado na Figura 2. Para além desta divisão de alto nível foi igualmente realizada uma divisão lógica das várias classes em pacotes significativos. No sentido de estruturar correctamente o código da aplicação,

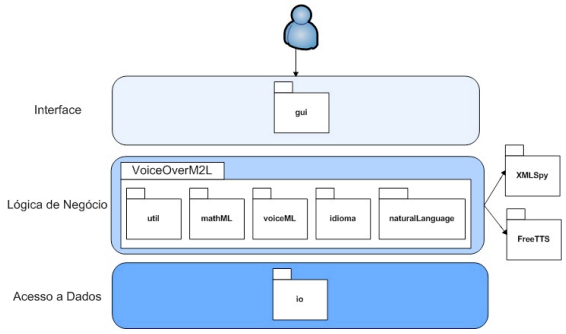

Figura 2. Diagrama de Camadas Ilustrativo da Arquitectura do VoiceOverM^{2}L

o mesmo encontra-se dividido num conjunto definido de unidades funcionais.

VoiceOverM2L Neste pacote encontra-se a classe nuclear de todo o programa, responsável pela coordenação dos múltiplos fluxos de dados entre os restantes pacotes.

GUI Neste pacote encontra-se um conjunto de classes responsável pela geração da interface gráfica e respectivo tratamento de eventos gerados pelos dispositivos de entrada.

MathML Neste pacote encontram-se as classes que geram a conversão de MathML para formato textual, recorrendo para tal aos pacotes de IO e Idioma.

Idioma Neste pacote encontra-se o conjunto de classes que realiza conversão de números e símbolos provenientes do universo restrito do dialecto MathML, convertendo-os para os seus congéneres textuais no idioma pretendido.

NaturalLanguage Neste pacote encontram-se classes responsáveis pelos aspectos fonéticos e conotativos do texto produzido. Assim, é neste ponto que são definíveis as características gerais do orador, a nível de idioma utilizado, sexo e idade, bem como características de elementos textuais específicos, como velocidade de leitura, pausas e timbre utilizado.

VoiceML Neste pacote encontram-se classes responsáveis pela vocalização do texto produzido, utilizando o pacote FreeTTS. Inclui igualmente as classes responsáveis pelo processamento e conversão em formato SSML. É de salientar o facto de características como pausas e entoação não terem ainda os seus valores devidamente afinados, por não se ter realizado ainda o estudo sobre o seu efeito na reprodução final.

Util Pacote de classes destinadas a implementar estruturas de dados cuja finalidade é a agregação dos resultados das chamadas a métodos de outras classes e pacotes.

IO Neste pacote encontram-se as classes responsáveis pela leitura de ficheiros no formato XML, posterior validação segundo *Schema* ou DTD definido e pré-processamento da árvore estrutural. Posteriormente o pacote irá conter classes responsáveis pela conversão em SSML do conteúdo textual produzido.

4.2 Tecnologias

Para o desenvolvimento do projecto foi utilizada uma linguagem de programação orientada a objectos, mais especificamente o Java (versão 1.5) [22], suportado pelo IDE Eclipse 3.1 [21].

Dado que a aplicação envolve manipulação de documentos em formato XML, é utilizada a interface externa do Altova XML Spy 2006 [20] para o processamento dos mesmos, acedida através da biblioteca disponibilizada para o efeito no pacote de instalação. Entre as potencialidades disponibilizadas pela biblioteca, são utilizados os procedimentos de validação de ficheiros segundo um dado *Schema* [13] ou DTD, leitura sequencial de etiquetas, de acordo com a estrutura do documento, e ainda geração de ficheiros para o dialecto SSML.

O projecto inclui ainda duas bibliotecas externas com objectivos específicos:

– O NumericalChameleon [32] é uma aplicação *open source* completa, em Java, que realiza múltiplas conversões numéricas, podendo ser os elementos alvo de conversão de natureza e idioma variados. No desenvolvimento da ferramenta foram utilizados os componentes responsáveis pela conversão de um dado literal para texto nos dois idiomas alvo: Português padrão (PT-PT) e Inglês dos Estados Unidos (EN-US).

– O FreeTTS [31] é um projecto *open source* de síntese de voz, a partir de vários formatos, incluindo também funcionalidades de reconhecimento de voz. No projecto são utilizadas as funcionalidades de reprodução vocal de blocos de texto representando a tradução em LN de fórmulas matemáticas. O pacote está optimizado para inglês, pelo que a reprodução em outros idiomas poderá não ser perfeita, no que concerne a questões de sotaque e entoação.

4.3 Processamento dos Formatos Anotados

Introdução Tendo em conta que todos os dados de entrada a processar se encontram em formato XML, mais concretamente no dialecto MathML, considerou-se útil recorrer a ferramentas de manipulação especializada em dados desta natureza. Para tal efeito, a aplicação de eleição foi o XML Spy da Altova, não só pela capacidade de manipulação e tratamento de informação no referido formato, mas também pela facilidade de geração de código de tratamento auxiliar (Figura 3-A), integração com a plataforma adoptada (Figura 3-B) e disponibilização de bibliotecas de tratamento e acesso a dados em formato XML. A destacar igualmente os bons níveis de desempenho proporcionados pela ferramenta.

Conversão MathML para LN Sendo os dados de entrada documentos normalizados segundo o dialecto MathML e tendo por objectivo último a sua descodificação e consequente tradução para LN, numa primeira instância para a língua portuguesa, possibilitando contudo fácil adaptabilidade a idiomas diversos, a abordagem natural passa por um conjunto estrito de etapas segundo a sequência que seguidamente se apresenta. Inicia-se o processo pela validação dos

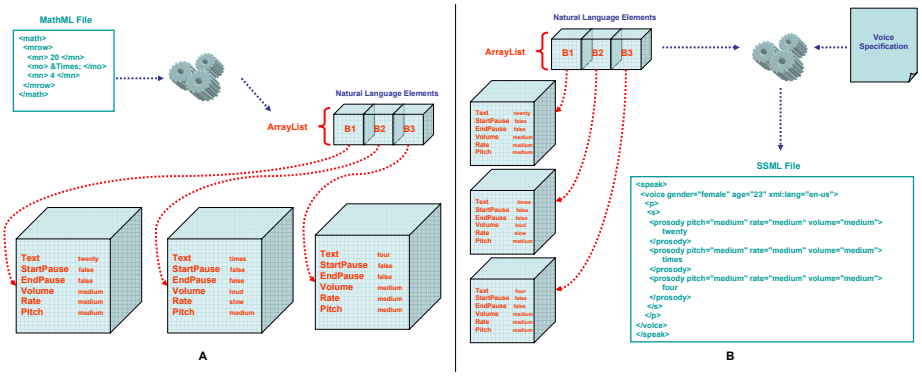

Figura 3. A-Conversão para Linguagem Natural B-Conversão para SSML

dados de entrada, recorrendo-se para tal a bibliotecas auxiliares fornecidas pelo ambiente de desenvolvimento seleccionado, ao que se segue a conversão da estruturação em MathML para uma outra equivalente, estando porém a informação

representada segundo o paradigma orientado a objectos, cumprindo obrigatoriamente a relação entre elementos e respectiva caracterização através dos tipos.

Neste ponto podem-se distinguir três etapas de maior relevo: identificação de elementos, respectivos tipos e sua relação com outros elementos.

No sentido de alcançar com sucesso os pontos descritos, tomam-se em conta os seguintes aspectos:

- Estabelecimento da correspondência entre a componente simbólica e a componente ligada à LN, utilizando para tal, e numa primeira fase, uma matriz de correspondência entre símbolos matemáticos e respectiva tradução.
- Correcta definição das várias características do vocábulo a proferir, não só a nível do termo a empregar na tradução de um dado símbolo matemático, mas também na questão da fonética, tomando em atenção a entoação específica requerida em determinadas situações, bem como informação sobre os tempos e pausas no discurso. Trata-se de um ponto que se reveste de extrema importância para o sucesso da vocalização. Por conseguinte, o esquema de dados resultante do processo contém não só a sequência de termos, como também meta informação fonética.

Conversão de LN para SSML É de salientar o facto do texto produzido em LN ser estruturalmente composto pela sequência de vocábulos, respectivas entoações e pausas. Deste modo, tornou-se necessário proceder a uma análise do esquema de dados que representa esta realidade e respectiva tradução para o formato SSML.

Para tal efeito, recorreu-se uma vez mais às bibliotecas disponibilizadas pelo XML Spy, de forma a produzir documentos válidos em formato SSML. Assim na primeira etapa deste processo é carregada a árvore estrutural de ficheiros deste tipo dada por um ficheiro base no formato em causa. Esse mesmo ficheiro é parte integrante do pacote da aplicação e por defeito não tem permissão de escrita. Posteriormente é criada uma nova árvore conceptualmente semelhante, mas mais extensa, em que as etiquetas produzidas espelham por um lado as opções tomadas pelo utilizador referentes às características fonéticas tais como: idade, sexo, idioma e por outro a especificidade dos elementos textuais correspondentes aos vários elementos provenientes do ficheiro original de entrada em formato MathML. Este processo é descrito na Figura 3-B.

Estes documentos podem ser alvo de processamento por aplicações externas, que se encarregam do respectivo processamento e eficaz reprodução do discurso codificado. A Figura 4 exemplifica como accionar o processamento completo de um ficheiro MathML previamente carregado, através da interface gráfica.

5 Conclusões

5.1 Satisfação de Objectivos

No planeamento inicial do projecto foram delineados vários objectivos, sendo que se considera que todos eles foram satisfatoriamente cumpridos.

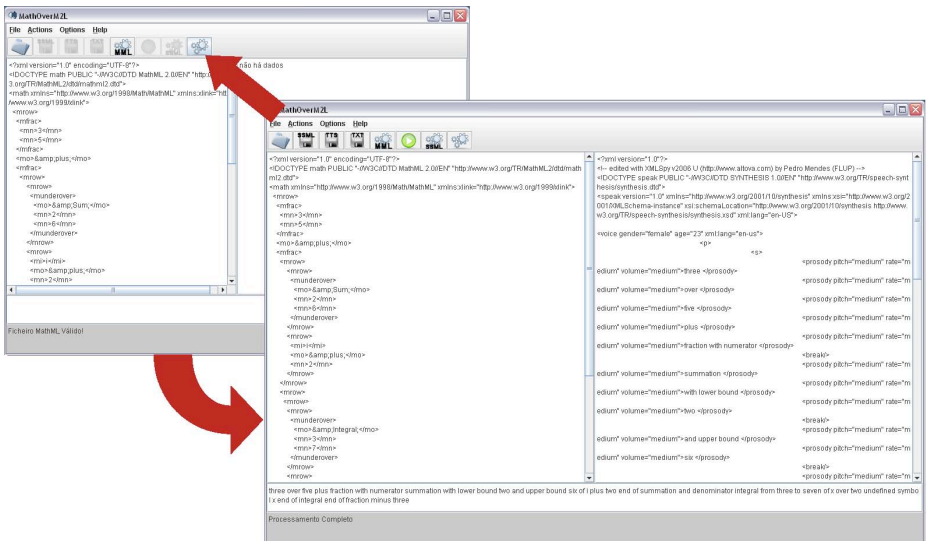

Figura 4. Accionamento do Processamento Completo de um Ficheiro em MathML

A leitura e validação de ficheiros em formato MathML segundo um determinado *Schema* ou DTD encontram-se finalizadas.

O processo de conversão de MathML para LN encontra-se também ele concluído, suportando operações aritméticas simples, operadores complexos tais como produtório e somatório, operador expoente, integrais simples, fracções, raízes, todo o tipo de agregadores,funções trigométricas e ainda identificadores incluindo o alfabeto grego.

A reprodução vocal do texto produzido encontra-se igualmente terminada, sendo que para tal se recorreu ao sintetizador de voz incluído no FreeTTS.

A conversão dos elementos textuais referentes a uma dada fórmula, para formato SSML, encontra-se também concluída suportando, tal como a reprodução vocal, dois idiomas, português e inglês.

Por último, foi igualmente desenvolvida uma interface gráfica, dentro dos actuais padrões de usabilidade, de suporte à aplicação.

5.2 Melhorias Futuras

No desenvolvimento deste tipo de ferramentas, encontram-se facilmente múltiplas extensões e melhoramentos. Dado que o código produzido apresenta elevados nível de modularidade, a expansibilidade para adição de mais idiomas seria uma forma de aumentar o nicho de possíveis utilizadores da ferramenta. Pelos mesmos motivos a expansão do domínio matemático para áreas mais específicas como a lógica ou teoria de conjuntos é igualmente possível. No sentido de afinar os parâmetros que controlam tanto a vocalização do texto como a produção

dos ficheiros em SSML, seria necessário um estudo aprofundado no domínio da linguística para cada um dos idiomas a considerar.

Referências

1. Long,Byron: Natural Language as an Interface Style - http://www.dgp.utoronto.ca/people/byron/papers/nli.html
2. W3C - http://www.w3c.org
3. Web Accessibility Initiative (WAI) - http://www.w3.org/WAI/
4. MathML - http://www.w3.org/TR/2001/REC-MathML2-20010221/
5. XML - http://www.w3.org/XML/
6. Bradley, N.: The XML Companion Third Edition Addison Wesley (2001)
7. SSML Overview- http://www.w3.org/TR/speech-synthesis/
8. Microsoft Text To Speech Engine - http://www.microsoft.com/reader/developers/downloads/tts.asp
9. Ferreira, H., Freitas, D.: AudioMath - using MathML for speaking mathematics (2005)
10. Lamport, L.: A Document Preparation System 2^{nd} Edition Addison Wesley (1994)
11. Mayer,M.: Lógica,Linguagem e Argumentação.Lisboa.Ed.Teorema (1992)
12. DTD - http://www.w3schools.com/dtd/
13. Schema - http://www.w3.org/XML/Schema/
14. History of MathML - http://www.mathmlcentral.com/history.html
15. VoiceXML - http://www.w3.org/Voice/
16. VoiceXML Overview- http://www.w3.org/TR/voicexml20/
17. Browser statistics - http://www.w3schools.com/browsers/browsers_stats.asp
18. Conferência XATA 2005 - http://vecpar.fe.up.pt/xata2005/programme.html
19. XML Spy Code Generation Features - http://www.altova.com/features_code.html
20. Altova XML Spy - http://www.altova.com/products_ide.html
21. Eclipse - http://www.eclipse.org/
22. Linguagem Java - http://java.sun.com/
23. Scalable Vector Graphics - http://www.w3.org/TR/SVG/
24. AudioMath Leitura Automática de Expressões Matemáticas - http://lpf-esi.fe.up.pt/~audiomath/
25. sMarTH online equation editor - http://smarth.sourceforge.net/
26. Lambda Project - http://www.lambdaproject.org/
27. MathPlayer - http://www.dessci.com/en/products/mathplayer/
28. Telisma - http://www.telisma.com/
29. Guide to Good Practice for learning and teaching in Languages, Linguistics and Area Studies - http://www.lang.ltsn.ac.uk/resources/goodpractice.aspx?resourceid=1411
30. VoiceXML Medical Application - http://www.voicexml.org/success_stories/ibm_mch_success.html
31. Free Text To Speech Engine - http://freetts.sourceforge.net/docs/
32. NumericalChameleon - http://www.jonelo.de/java/nc/

Repositório de Testes e Exames para e-Learning

António Lira Fernandes

Universidade do Minho
alf@nortenet.pt

Resumo. Neste artigo apresentamos uma visão geral dos tópicos a atender na elaboração de um sistema de testes e exames. Olhamos para a avaliação nas suas diversas modalidades e vemos o papel que ocupam no e-Learning. Analisamos como decorre o esforço de normalização da estrutura dos testes e exames e que ganhos se procuram obter, centrando a nossa atenção na norma *QTI* da *IMS*. Terminamos apresentando o modelo de implementação de um protótipo de teste e exames segundo as recomendações da *IMS*.

0 Introdução

Qualquer plataforma de e-Learning dispõe de ferramentas de testes e exames, no entanto, são desenvolvidas de forma proprietária, suportadas por bases de dados privadas e muitas vezes centradas no seu criador/proprietário.

O objectivo principal deste trabalho é definir um quadro que permita a reutilização de teste e exames, numa escala que extravasa o sistema onde foram criados. Para tal procuramos apresentar a norma da *IMS*[1] relativa a testes e exames *(QTI*[2]*)* que funciona como referência a este trabalho.

Um protótipo de repositório, compatível com a versão 2.0 do *QTI,* é apresentado neste artigo e foi usado para consolidar as descoberta que resultaram do estuda da norma da *IMS*.

Este documento está estruturado em cinco capítulos. No capítulo 2, apresentamos o e-Learning como resposta às novas necessidades da sociedade actual, abordamos os conceitos relacionados com a avaliação e as vantagens de uma avaliação sistemática dos alunos no processo de avaliação e diagnóstico.

O capítulo 3 apresenta o esforço da *IMS* no sentido de desenhar uma norma que permite a reutilização de testes e exames. Acompanhamos as principais alterações da norma *QTI* da *IMS*, desde a versão 1.2 até a versão 2.1 *public draft*, olhando para o elemento principal da especificação o item de avaliação e para a estrutura dos testes e exames.

No capítulo 4, apresentamos o protótipo de implementação de um sistema de testes e exames de acordo com a norma *QTI* e olhamos com mais detalhe da a estrutura interna do repositório desenvolvido .

Finalmente, são apresentadas algumas conclusões e pistas futuras sobre o tema.

---

[1] IMS Global Learning Consortium: http://www.imsglobal.org/ 2007
[2] IMS Question & Test Interoperability – norma da IMS para testes e exames

1 e-Learning

Por um lado a sociedade moderna tende a reforçar a importância da formação ao longo da vida como resposta à constante evolução tecnológica, à necessidade crescente de especialização de conhecimentos que resultam da maior segmentação do trabalho e à necessária polivalência dos recursos humanos[0].

Por outro, a tecnologia está em constante evolução e é disponibilizada a baixo preço permitindo a sua disseminação em massa, facilitando a execução de diversas tarefas que, sem ela, só podia ser levados a cabo de forma esporádica.

A informação circula em tempo real, obrigando a alterar as nossas práticas e promovendo a mobilidade no trabalho. Voltar à escola é fundamental para ter sucesso neste novo paradigma, mas a gestão do tempo é uma tarefa difícil e que obriga a uma procura por flexibilizar o horário de funcionamento das escolas, dando resposta às necessidades dos "novos" alunos. O e-Learning cresceu para ser a solução, e é hoje apresentado em muitas e variadas formas, em sistemas diversos com soluções mais ou menos caseiras.

Nas várias implementações de e-Learning a componente de avaliação é um elemento chave. Todos os sistemas de e-Learning dispõem de ferramentas de criação e edição de testes que normalmente têm uma estrutura proprietária, o que as torna limitativa em determinados contextos e que complica a sua reutilização em sistemas distintos.

1.0 Avaliação

A avaliação é parte integrante do processo de ensino e de aprendizagem e pode servir como elemento regulador e de orientação quanto ao percurso escolar do aluno, como elemento certificador de conhecimentos, ou como indicador da qualidade de um determinado percurso[1].

Os tipos de avaliação mais comuns são: **Avaliação Diagnóstica** – utilizada para adoptar estratégias de ensino aprendizagem diferenciadas para cada aluno ou grupo de alunos e pode ocorrer em qualquer altura; **Avaliação Formativa** – normalmente é contínua e deve acontecer de forma assídua e persistente, deve ser o mais diversificada possível; **Avaliação Sumativa** – consiste em quantificar de forma global as competências e conhecimentos dos alunos, por vezes é dividida em avaliação sumativa interna ou externa[1][2].

A avaliação é uma componente importante em e-Learning. Num primeiro instante é o instrumento que faz o diagnóstico do aluno, de seguida e durante todo o percurso promove a avaliação formativa e no final confirma o nível de conhecimento numa determinada área (avaliação formativa).

O facto de nos sistemas de e-Learning a produção e a correcção de instrumentos de avaliação poder ocorrer de forma semi-automática[3] torna possível uma avaliação sistemática com registo individual, persistente, o que facilita a avaliação contínua dos alunos e das unidades, disciplinas ou cursos.

---

[3] Automática em alguns aspectos dependendo das opções do professor ou dos mecanismos disponibilizados pelas ferramentas utilizadas.

1.1 Normalização de sistemas de testes e exames

Durante as últimas décadas, as tecnologias de informação e comunicação e o e-Learning tem merecido diversas abordagens que vão sendo materializadas em muitas aplicações Web ou pacotes de software de uso doméstico. Este esforço, por vezes individual, não tem acautelado o prolongamento do tempo de vida dessas iniciativas nem a imunidade às mudanças das tecnologias envolvidas. Se relativamente à forma como os conteúdos são produzidos vai havendo standards que correspondem às vitórias comercias dos produtos *Microsoft*, dos formatos *PDF* e mesmo dos standards Internet, no que diz respeito à catalogação da informação o esforço de standarização está muito mais atrasado.

Uma dificuldade que decorre da normalização é obrigar os fornecedores de soluções a assumirem um custo extra que passa por dominar o standard e implementá-lo nas suas soluções. Os utilizadores também sofrem com a normalização já que têm de alterar práticas e passar a preencher mais informação do que a que efectivamente necessitam.

No próximo capítulo olharemos para um consórcio que se dedica a esse esforço de normalização.

2 IMS Question & Test Interoperability (QTI)

A iniciativa de normalizar testes e exames foi iniciado em Março de 1999, pela *IMS Global Learning Consortium (IMS)* que é uma organização sem fins lucrativos composta, à data, por mais de 50 membros e associados. Dispõem de várias normas, mas neste trabalho vamos centrar a nossa atenção na norma de testes e exames (*QTI*), que está, actualmente, disponível na versão 2.1, em *public draft*[3].

A especificação permite a troca de itens[4], testes de avaliação e resultados entre ferramentas de autor, repositórios de itens, sistemas de aprendizagem, e sistemas de produção de avaliações.

Na versão 1.x a especificação usava a estrutura *ASI*, que significa *Assessment, Sections, Item* e definia uma estrutura em que os itens eram associados em secções que compunham o teste de avaliação[4].

Na versão 2.0 o esforço foi centrado no item que assumiu o papel principal na especificação, passando a ser designado por item de avaliação. Não se tratou unicamente de uma alteração na nomenclatura da pergunta, o item de avaliação (*assessmentItem*) passou a incluir comportamento e formato[5][6].

Na versão 2.1, ainda em *public draft*, foi confirmada a importância do item de avaliação e voltou a surgir uma especificação para a estrutura de testes e exames. Assim surge o teste de avaliação (*assessmentTest*) que tenta definir uma estrutura que englobe um conjunto de itens de avaliação[7]. As ideias que tinham sido avançadas na versão 1.2, que definiam um teste, são em grande medida re-aproveitadas sendo

[4] Ao item corresponde uma pergunta de um teste e incluiu todos os elementos necessários a sua apresentação, regras de comportamento, feedback e resposta. Podemos ver o item como o mais pequeno elemento de um teste que faz sentido por si só.

definida uma "nova estrutura" para o teste, que iremos apresentar num dos subcapítulos seguintes.

Tal como nas versões anteriores a portabilidade dos sistemas é conseguida com a utilização do *XML*[8] que passa a ser o formato de ficheiros a ser trocado entre sistemas. Nesta versão foram definidos cinco sistemas, sete actores e algumas regras. Na **Fig. 1**, é apresentado um modelo abstracto que representa, de forma simplificada, um ambiente de avaliação[3].

A divisão dos sistemas de avaliação em cinco componentes permite desenvolver em paralelo e de forma independente aplicações/sistemas para responderem aos requisitos do modelo.

Os componentes/sistemas representados são: *AuthoringTool* – ferramentas de autoria - sistema usado pelos autores para criar ou modificar um item de avaliação; *ItemBank* – repositório de colecções de itens e avaliações; *AssessmentDeliverySystem* – sistema de apresentação de testes ao candidatos[5]. Dispõem de um motor de entrega, avaliação e cotação automaticamente das respostas dos candidatos; *LearningSystem* – sistema de aprendizagem - que autoriza ou direcciona os alunos para actividades de aprendizagem em coordenação com o tutor; e *testConstructionTool* - ferramenta de construção de testes que usando os itens disponíveis no repositório permite construir instrumentos de avaliação (testes).

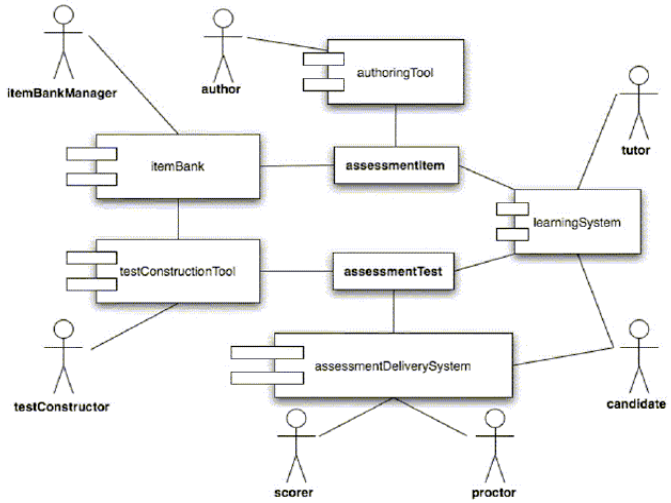

Fig. 1. Os actores das avaliações – modelo abstracto em UML[1]

A norma identifica sete actores do sistema: *Author* – o autor de um item de avaliação; *ItemBankManager* – o responsável pela gestão da colecção de itens de avaliação num repositório; *Proctor* – o responsável por supervisionar a resolução de avaliações e que não tem responsabilidade de correcção; *Scorer* – os correctores das avaliações, que podem não existir já que a correcção das avaliações pode ser

[5] Designação atribuída a quem está sujeito a uma avaliação, no original *learner*.

automática; Tutor – o responsável pela orientação do processo de aprendizagem do aluno; *Candidate* – quem está a ser avaliado; e *TestConstructor* - o construtor de testes.

2.0 Item de Avaliação (*AssessmentItem*)

O item de avaliação é a unidade mais pequena em que podemos dividir uma avaliação, podemos compará-la a uma alínea da avaliação, que inclui mecanismos de correcção, níveis de feedback a dar ao aluno, formato etc.

A estrutura do item de avaliação está dividida em oito elementos, cinco dos quais são facultativos. O elemento ***responseDeclaration*** é usado para definir a forma de resposta à pergunta e as variáveis de resposta associadas. A cotação e as variáveis de resposta são definidas no elemento ***outcomeDeclaration***.

O elemento ***templateDeclaration*** é usado para registar informação sobre um item que é um modelo e serve para criar itens semelhantes através de regras de criação de réplicas do item, possibilitando gerar perguntas aleatórias. Em ***templateProcessing*** é incluído o conjunto de regras que são utilizadas para preencher as variáveis de *Template*[6].

Relativamente ao formato podemos usar o elemento ***stylesheet*** para associar folhas de estilos a um item de avaliação.

O conteúdo da questão é introduzido no elemento ***itemBody*** sendo este um elemento obrigatório e central. Outro elemento obrigatório é o ***responseProcessing*** que serve para definir o que deve acontecer quando o candidato responde à questão.

O elemento ***modalFeedback*** é usado para registar o feedback que deve ser apresentado ao candidato em alternativa ao que pode ser incluído no *itemBody*.

2.1 Teste de Avaliação (*assessmentTest*)

Um teste é um conjunto de perguntas para as quais são definidas regras que determinam o que vemos, por que ordem e a forma como se pode interagir com as perguntas. As regras, podem também determinar a forma pela qual será permitido aos candidatos navegar ao longo do teste, em que instante as respostas são enviadas para o processamento de respostas e o nível de feedback pretendido.

Cada teste é dividido em partes (Fig. 2) para as quais é definida a forma como vamos poder navegar em cada uma. Existem duas forma de navegar numa parte: de forma linear ou de forma não linear. Na forma linear caminhamos ao longo das perguntas sem podermos voltar atrás e numa ordem pré-estabelecida. Na forma não linear podemos saltar perguntas e voltar atrás sempre que desejarmos até que seja feita a submissão das respostas.

---

[6] Trata-se de um tipo especial de item de avaliação que serve para gerar de forma automática instâncias de um tipo de pergunta.

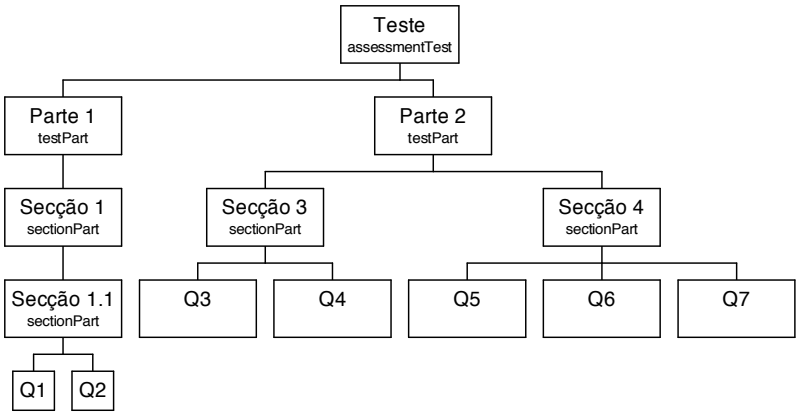

Fig. 2. Exemplo de uma estrutura de teste

Da mesma forma, em cada parte do teste, podemos decidir a regra para o envio das respostas. Assim, existem duas formas de envio de respostas: individual, que pressupõe o envio de uma resposta de cada vez; e o simultâneo que envia todas as respostas de uma vez.

A especificação deixa em aberto a possibilidade de o candidato poder rever o teste depois de corrigido e de ser definido um tipo de feedback para esse instante.

Foi definida a possibilidade de impôr regras que permitam numa qualquer secção só mostrar de forma aleatória algumas perguntas ou alterar a ordem das perguntas numa secção.

2.2 Metadados

Os metadados vão ser fundamentais para estabelecer um catálogo de itens de avaliação que poderá ser reutilizado noutros contextos e por outros utilizadores. A *IMS* utiliza a norma *IEEE LOM*[7] como base e estende-a com mais alguns elementos que são particulares para os itens de avaliação (*qtiMetadata*)[7][9].

A Tabela 1 mostra os metadados específicos do *QTI* e a Tabela 2 os metadados *LOM* usados para catalogar conteúdos genéricos.

Tabela 1. A classe *qtiMetadata* e uma categoria de topo do LOM para descrever uma nova categorias dos metadados QTI

itemTemplate	[0..1]	*Boolean: true* se se tratar de um *template* – altera de forma aleatória o seu aspecto de acordo com factores externos
timeDependent	[0..1]	*Boolean: true* se depende do tempo (tempo limite)
composite	[0..1]	*Boolean: true* se tem mais do que uma interacção

[7] Learning Object Metadata

interactionType	[n]	Tipo de interacção (usa uma lista de tipos de interacção previstas na norma)
feedbackType	[0..1]	Descreve o tipo de feedback (*none, non-adaptive; adaptative*)
solutionAvailable	[0..1]	*Boolean: true* se incluir a solução
toolName	[0..1]	Nome da ferramenta de criação
toolVersion	[0..1]	Versão da ferramenta de criação
toolVendor	[0..1]	Fabricante da ferramenta de criação

Tabela 2. Perfil *IEEE LOM* utilizado na especificação *IMS[10]*

General	Descrição dos itens na generalidade
Identifier; Title; Language; Description; Keyword; e Coverage.	
Life Cycle	Estado actual do objecto
Version; Status; e Contribute.	
Meta-Metadata	Dados sobre os dados (esquema)
Identifier; Contribute; Metadata Schema; e Language	
Technical	Descrição das características técnicas do curso/objecto/item
Format; Size; Location; e Other Platform Requirements	
Educational	Descrição das características educativas do curso
Learning Resource Type; Context; Typical Learning Time; Description; e Language	
Rights	Direitos de autor
Cost; Copyright and other restrictions; e Description	
Annotation	Anotações
Classification	Classificação do curso/conteúdo

3 Modelo de Implementação

Um sistema de testes e exames comporta um conjunto de componentes que interagem de forma autónoma prestando serviços a outros componentes ou utilizadores finais.

Na figura 3 está representado um sistema de testes e exames. Na base do sistema podem existir vários *itemBank* (pelo menos um) que podem estar distribuídos e que servem de repositórios de itens, respondendo a interrogações colocadas por outros repositórios ou por outros componentes. A troca de ficheiros é realizada através de ficheiros em formato *XML*[11] em conformidade com a estrutura definida para os testes e exames de acordo com o *QTI*, para garantir a interoperabilidade dos sistemas[8].

Os Sistemas de Aprendizagem *(Learning System)* acedem ao repositório utilizando uma *API* de interrogações e recebem ficheiros *XML* e outros necessários a apresentação do item em pacotes *PIF*[9]. Às Ferramentas de Criação *(Authoring Tools)*

[8] Cada componente pode ser de um fornecedor diferente, desde que implemente a norma

[9] Sempre que é necessário transportar ficheiros entre sistemas heterogéneos usamos um Ficheiro do Pacote de Intercâmbio - *Package Interchange File (PIF)*. Assim criamos um

é permitido enviar itens para os repositórios, usando a API disponível, que deve garantir a correcta catalogação do item de avaliação.

Os Sistemas de Apresentação de Testes *(Assessment Delivery System)* usam os itens os testes e fornecem ficheiros *XML* com estatísticas e resultados dos testes (esta funcionalidade ainda não está definida em detalhe na especificação da *IMS*).

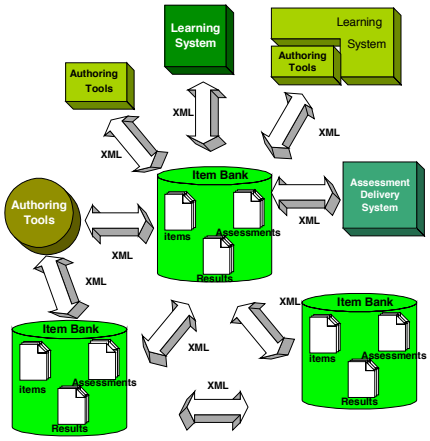

Fig. 3. Modelo Ambiental do Sistema

Para compreender este quadro foi desenvolvido um protótipo que incluiu dois componentes: um repositório e um mini sistema de apresentação de perguntas (limita-se a apresentar as perguntas sem implementar a interacção com o utilizador).

3.0 Repositório

O repositório faz o armazenamento dos itens em sistema de ficheiros sendo os meta dados armazenados em base de dados, permitindo transferir a complexidade da distribuição e das pesquisas para o motor de base de dados escolhido.

A base de dados relacional vai fornecer a meta-informação, acerca dos itens de avaliação permitindo aos utilizadores registados no sistema, efectuarem pesquisas sobre determinados tipos de itens. O tipo de pesquisa tem por base tanto os *LOM* como a extensão de metadados específicos do *QTI* e resulta na possibilidade de fazer o download de um *PIF* com o(s) item(s) de avaliação para ser usado num outro sistema compatível com o *QTI*.

O protótipo foi desenhado num modelo com três camadas. Na camada de ligação com os utilizadores existem funcionalidades para pesquisa e para armazenamento de novas questões, baseadas numa interface web.

---

ficheiro ZIP com todos os ficheiros que queremos transportar e adicionamos informação sobre o pacote, num ficheiro *XML* designado por *imsmanifest.xml* [12][13].

O núcleo foi desenvolvido em *PHP* e tem uma hierarquia de classe com funções de pesquisa, e funções para armazenamento e catalogação de itens.

3.1 Mini sistema de apresentação de perguntas

Um outro componente, do protótipo consiste num sistema que permita apresentar itens mediante a leitura de ficheiros *XML* estruturados de acordo com a norma *QTI*. Com esse propósito foi implementada uma hierarquia de classes em *PHP* que suportam o modelo definido na especificação.

Este sistema não implementa todos os tipos de perguntas definidas na especificação *QTI*. No entanto funciona, associado ao repositório permitindo visualizar os itens de avaliação ajudando na selecção das perguntas disponíveis no repositório.

3.2 Arquitectura da Base de Dados

O sistema dispõe de mecanismos de armazenamento e pesquisa que são suportados por uma base de dados[10]. Dividimos as informações do manifesto em três estruturas de base de dados. Uma geral, que guarda informação sobre o *PIF* e sobre o caminho no sistema de ficheiros do repositório. Uma segunda estrutura com a informação do *LOM* e uma terceira com os metadados específicos do *QTI*. Desta forma, podemos usar o repositório não só para questões mas também para conteúdos.

O registo dos manifestos no repositório é feito numa única tabela sendo que todos os campos resultam dos atributos do manifesto[11] excepto um campo (*IdentifierVersion*) que será um número único automático que servirá de chave primária da tabela.

A arquitectura da base de dados para guardar os *LOM* resulta de uma selecção de um conjunta da informação que será mais frequentemente objecto de pesquisa pelos utilizadores.

Para cada pacote registamos os metadados do *PIF* e os metadados de cada ficheiro incluído no pacote. Não utilizamos nenhum tipo de mecanismos para garantir que os ficheiros enviados para o repositório são únicos. Esse esforço foi unicamente dirigido para os pacotes.

A arquitectura da base de dados dos metadados específicos do *QTI* é a apresentada na Fig. 4. Neste caso incluímos toda a informação por se tratar da informação que é mais específica do propósito deste repositório.

---

[10] A base de dados utilizada foi o *MySQL*®, no entanto o sistema suporta qualquer tipo de base de dados desde que seja fornecida uma classe específica que implemente todos métodos definidos na classe genérica bd.

[11] Os campos da tabela manifest são: *identifier; imsmd; version; base; dir; e IdentifierVersion*

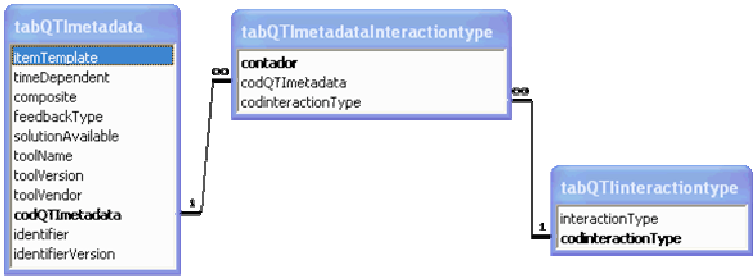

Fig. 4. A arquitectura do *qtiMetadata*

3.3 Sistema de ficheiros e arquitectura do site

Os pacotes são guardados em sistema de ficheiros, numa pasta *assessmentpack* na pasta de dados típica do *Moodle*[12]. Sempre que é recepcionado um pacote é criada um pasta com uma estampa temporal em *assessmentpack*, na qual são descompactados todos os ficheiros que compõem o *PIF* para fazer as validações necessárias, obter a meta-informação e ser possível ao sistema de apresentação de perguntas realizar a sua função.

Na Fig. 5 é apresentada a estrutura do *site* do repositório. As configurações são guardadas num ficheiro de texto (`config.php`) e é usada uma folha de estilo para definir o aspecto gráfico das páginas web (`formatacoes.css`).

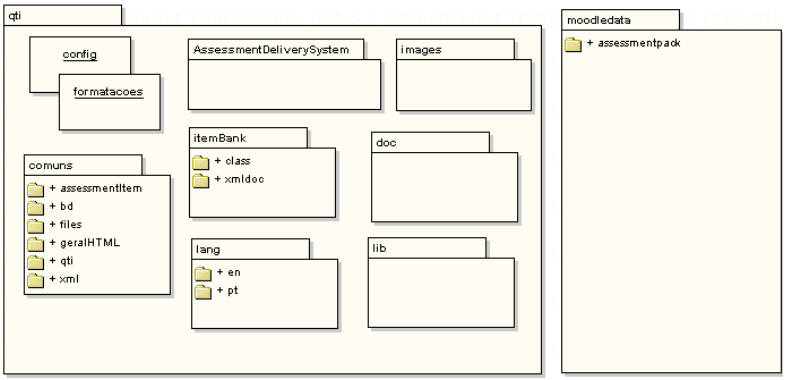

Fig. 5. Estrutura do Site

Na pasta *AssessmentDeliverySystem* existem dois *script PHP* que carregam a classe *assessmentItem* e que servem para apresentar e receber as respostas a uma

[12] *Moodle* é o acrónimo de *Modular Object-Oriented Dynamic Learning Environment* e é um sistema gestor de cursos livre, sob licença *Open Source*[14].

pergunta. Estes *scripts* podem ser usados, de forma independente, para apresentar uma pergunta lendo um ficheiro *XML* conforme com o *QTI IMS* versão 2.0.

Na pasta `itemBank` existe um conjunto de classes que permite criar um repositório. Existe também um conjunto de documentos *XML* que pode ser usado como exemplo de teste do sistema e que são fornecidos na especificação da *IMS*.

Na pasta `images` existem algumas imagens usadas no site. Na pasta `doc` existe uma descrição do site. Na pasta `lib` existem algumas funções do *Moodle* usadas para integrar o repositório com a plataforma (a integração com o *Moodle* ainda não está concluída).

Na pasta *comuns* existem várias classes usadas pelo repositório como prestadores de serviços especializados. Na Tabela 3, é apresentada uma breve descrição sobre a pasta *comuns*.

Tabela 3. Sub pastas da pasta comuns

Pasta	Funções dos *Scripts* e Classe
assesmentItem	Hierarquia de classes comuns que descreve um pergunta em conformidade com a *QTI* da *IMS* e que é usadas pelo repositório e pelo mini sistema de apresentação
bd	Conjunto de classes que permitem manipular uma classe genérica para acesso a uma base de dados. Neste protótipo suporta *Microsoft Access* e *MySQL*.
files	Uma classe que manipula ficheiros num sistema de ficheiros.
geralHTML	Classes que permitem manipular páginas e formulários *HTML*. Dispõem também de uma classe para armazenar utilizadores e disponibiliza mecanismos de autenticação.
qti	Algumas classe que definem tipos básicos de dados e que enumeram elementos em conformidade com a norma *QTI*.
xml	Classes que permitem ler ficheiros *XML* e que criam classes *PHP* usando um *DTD*.

4 Conclusão

No universo do e-Learning a oferta de sistemas é vasta e baseada em modelos proprietários, que foram crescendo como resultado das necessidades e não tanto baseados em normas internacionais. Os principais fornecedores vão integrando as normas nos seus sistemas, permitindo a interoperabilidade e a reutilização dos conteúdos, muitas vezes recorrendo à mera possibilidade de exportar dados para *XML*. Mas o que se procura é uma maior integração baseada na adopção de estruturas compatíveis com as normas relativas ao e-Learning.

No subgrupo dos testes e exames existem também muitos produtos que permitem criar testes e exames mas todos com estruturas e granularidade distintas. Pretendemos neste estudo dar alguma contribuição na compreensão e aplicação das normas *QTI* da *IMS*. Não tem havido muitos estudos sobre a frequência da reutilização de testes pelos próprios autores e por outros e sem a adopção de uma norma de armazenamento essa partilha é extremamente difícil.

Este artigo resultou do esforço inicial do nosso estudo sobre os testes e exames para e-Learning. O protótipo desenvolvido está em conformidade com a versão 2.0 da norma, mas não inclui a estrutura dos testes que só foi especificada na versão 2.1.

A importância da avaliação tem sido crescente e é comum pretendermos avaliar para diagnosticar, avaliar para certificar, avaliar para adoptar as melhores estratégias de aprendizagem, e sendo assim, o investimento em ferramentas que permitam avaliar de forma contínua e sistemática podem ajudar na qualificação dos recursos humanos e na melhoria das organizações.

Referências

[0]. Lima, J. R.; Capitão Z.: e-Learning e e-Conteúdo. Centro Atlântico (2003)

[1]. Ministério da Educação: Avaliação dos alunos Ensino Básico: http://www.min-edu.pt/Scripts/ASP/novidades_det.asp?newsID=316. (2005)

[2]. Bonniol, J., Vial, M, "Modelos de avaliação. Textos fundamentais com comentários", Porto Alegre, Artmed Editora, (2001)

[3]. IMS Global Learning Consortium: "*IMS Question & Test Interoperability: Overview*". Version 2.1 Public Draft Specification (2006)

[4]. IMS Global Learning Consortium: "*IMS Question & Test Interoperability: An Overview*". Final Specification Version 1.2 (2002)

[5]. IMS Global Learning Consortium: "*IMS Question & Test Interoperability: Item Overview*", Version 2.0 Public Draft (2004)

[6]. IMS Global Learning Consortium: "*IMS Question & Test Interoperability: Migration Guide*". Version 2.0 Final Specification (2005)

[7]. IMS, "*IMS Question & Test Interoperability: Information Model*", Version 2.1 Public Draft Specification (2006)

[8]. Ramalho, J.C., Henrique, P., "*XML & XSL* da teoria à prática", FCA (2002)

[9]. IMS Global Learning Consortium: "*IMS Question & Test Interoperability: Item Meta-data and Usage Data*". Version 2.0 Final Specification (2005)

[10]. Information School - University of Washington, "*IEEE 1484 Learning Objects Metadata (IEEE LOM)*" (2006)

[11]. Tarasov, V.A., Tarasov, V.V., Kyurshunov, A.S.; "*Using xml and the IMS QTI standard for the development of assessment tools*", Proceedings of the Sixth Inter-Karelian Conference Sortavala, Russia (2003)

[12]. IMS, "*Using IMS Content Packaging to Package Instances of LIP and Other IMS Specifications*", Version 1.0 Implementation Handbook (2001)

[13]. Becta, "*Specification of Content Delivery Services*" (2003)

[14]. Moodle, 2006; http://www.moodle.org

MITRA: Uma Solução para Serviços de Pesquisa em *Intranets*

Jorge Machado[1], José Borbinha[2]

[1]Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Portalegre, Lugar da Abadessa, Apartado 148
7301-901 Portalegre, Portugal
jmachado@ext.bn.pt
[2]INESC-ID – Instituto de Engenharia de Sistemas e Computadores, Rua Alves Redol 9
Apartado 13069, 1000-029 Lisboa, Portugal
jlb@ist.utl.pt

Resumo. Este artigo descreve o sistema MITRA, uma solução para indexação de conteúdos em linha e metadados descritivos complementares codificados em qualquer esquema XML. Esta capacidade torna este sistema uma solução ideal para serviços especializados de pesquisa em intranets. O MITRA baseia-se numa arquitectura com cinco camadas. A primeira camada é a de recolha de conteúdos que pode ser implementada por sistemas externos ou sistemas especializados de transferência de recursos, como por exemplo arquivos locais estruturados. A segunda camada cria índices invertidos dos conteúdos e dos metadados recolhidos (usando o sistema LUCENE). A terceira camada gere as relações semânticas e as associações dos metadados aos recursos. . Uma quarta camada muito recente permite a implementação de uma metodologia de análise do domínio. A última camada, a de apresentação, permite receber pesquisas estruturadas em pedidos HTTP e responder em XML ou HTML conforme sejam ou não utilizadas XSL's. O esquema de representação interna dos metadados é o Dublin Core, o qual permite ao MITRA fornecer naturalmente uma interface de SRU/SRW, mas outros esquemas podem ser também configurados. O MITRA combina assim o poder da indexação livre de conteúdos com o poder do processamento de metadados estruturados, oferecendo o melhor dos dois mundos. Esta solução é usada como suporte a vários serviços efectivos, reportados no texto.

Introdução

É normal uma organização publicar hoje em dia conteúdos através de múltiplos serviços e portais. Neste artigo descreve-se o MITRA, uma ferramenta para indexar espaços dessa natureza e fornecer serviços de pesquisa e analise desse mesmo espaço ou domínio. Este sistema pode indexar conteúdos publicados em linha de forma livre ("sítios" HTML) ou conteúdos estruturados aos quais podem ser associados quaisquer esquemas de metadados codificados em XML. Esta solução é usada como suporte a vários serviços efectivos no contexto da Biblioteca Nacional.

Este artigo prossegue com uma explicação do que se entende por recursos e colecções no contexto do MITRA, e que implicações isso traz na criação dos índices.

O artigo prossegue com a descrição da arquitectura do MITRA. De seguida descrevemos as técnicas usadas para optimizar as pesquisas e as apresentações de resultados, assim como a forma como os metadados são utilizados para melhorar essas apresentações. Em sequência explica-se que serviços remotos são fornecidos pelo MITRA para facilitar a interoperabilidade com outros sistemas.

Por fim propomos uma metodologia suportada pelo MITRA para análise de contextos, onde apresentamos um caso de estudo da revista DLIB e a forma como se está a desencadear a análise da mesma. Esta capacidade é uma forma inovadora de usar estes sistemas, podendo suportar processos de extracção de conhecimento em Intranets ou colecções de conteúdos concretas.

Colecções e Recursos no MITRA

O MITRA destina-se a indexar sítios perfeitamente delimitados e estruturados. Para o MITRA cada sítio é um recurso, que pode ser associado não só com metadados descritivos ou estruturais de codificados em XML, só mas também com colecções.

No MITRA uma colecção é um conjunto de recursos que pertencem a um espaço virtual bem definido. As colecções no MITRA não são no entanto factor restritivo em nenhum aspecto. Em termos de pesquisa o agrupamento de recursos que se faz com uma colecção pode ser feito com qualquer outro índice vindo ou não de um esquema XML. No entanto pensamos ser importante introduzir as colecções para marcar uma posição mais forte que um mero índice que pode ser invocado numa pesquisa. As colecções têm a associados vários dados que de alguma forma podem vir a influênciar a arquitectura futura do MITRA. Exemplos disso são a localização do INDICE, sendo que no futuro o MITRA poderá vir a ter múltiplas camadas de indexação, usando assim sistemas de indexação diferentes para cada colecção. A colecção está integrada no esquema de pesquisa e resposta do MITRA sem ter que estar especificamente dentro da *query* efectuada, etc. Cada recurso é identificado por um URN e pode estar associado a várias descrições na forma de metadados. Quando uma pesquisa é lançada o MITRA retorna os resultados organizados em grupos de recursos (*clusters*) nos quais se podem identificar os metadados, a colecção, a informação de estado e a informação básica acerca do resultado principal dentro de um recurso (Figura 1).

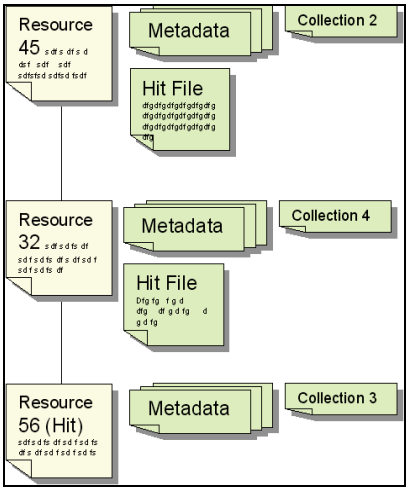

Figura 1 – Resposta estruturada em XML

Um exemplo poderá ser visto no serviço de Busca da Biblioteca Nacional [2], bastando para isso aceder à seguinte ligação (como neste caso nenhuma opção XSL é especificada, o resultado será retornado em XML):

http://busca.bn.pt/Handler?verb=search&query=lusiadas

As colecções permitem ao MITRA dividir os seus alvos de trabalho por áreas distintas o que lhe confere um ainda maior poder de pesquisa. Assim como nos restantes casos é criado um índice especial para identificar os documentos de cada colecção sendo que facilmente se filtra uma pesquisa.

Arquitectura do MITRA

O sistema MITRA tem uma arquitectura de cinco camadas codificadas em JAVA, tal como se mostra na Figura 2.

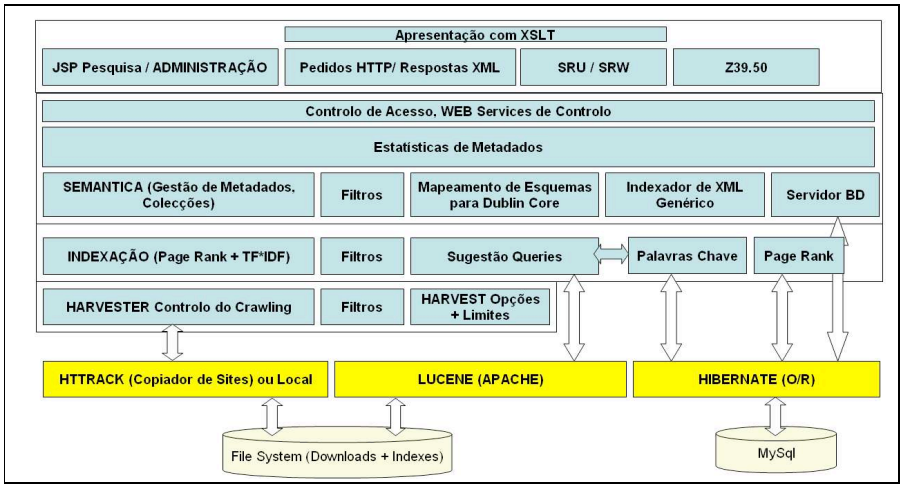

Figura 2 – Arquitectura do MITRA.

Neste sistema reutilizam-se vários componentes e soluções já existentes, conforme se pode constatar na descrição da arquitectura no ponto seguinte. Por exemplo, a camada de indexação usa o LUCENE [10] para a criação de índices, enquanto que a camada de recolha pode usar o HTTRACK [8] para recolha de recursos em linha. Podem ser desenhados ainda componentes de interfaces especializados para arquivos locais de conteúdos estruturados. No caso do serviço BUSCA [2] da Biblioteca Nacional, as descrições bibliográficas codificadas em XML são recuperadas da PORBASE [18], em esquemas como o UNIMARC [27], que é depois convertido em Dublin Core [25]. Isto permite, por exemplo, pesquisar a Biblioteca Nacional Digital [1] usando não apenas pesquisa no texto livre mas também sobre índices estruturados construídos a partir das descrições bibliográfica, tais como autor, assunto, título, data de publicação, resumo, etc.

O sistema compreende ainda uma camada semântica para a gestão dos metadados que cada serviço possa fornecer ao MITRA.

Fluxo de trabalho do MITRA

A Figura 3 ilustra a sequência de processos no fluxo de trabalho do MITRA.

Quando um pedido de indexação é feito pela primeira vez, a camada de controlo de acesso valida o pedido, que é depois passado à camada semântica para esta criar, para o recurso a indexar, um identificador na forma de um URN. Com esse identificador um cliente poderá invocar o MITRA sempre o desejar, com pedidos de reindexação ou, por exemplo, de adição de metadados descritivos.

Os pedidos de indexação pendentes são passados, de forma assíncrona, à camada de Indexação, que por sua vez os passa à camada de *Harvesting*.

Mais acima, o primeiro passo da camada semântica será fazer o carregamento de todos os metadados existentes em XML (descrições bibliográficas ou outras relevantes). Este fluxo permite a alguns dos filtros de *crawling* usar esses metadados para decidir aquilo que irá ser copiado. É de salientar que estes metadados podem conter informação de *broker* para outros metadados. Todo este processo é determinado por filtros configuráveis, que podem ser criados à medida de cada caso. depois de ter sido feito o download de todos os "recursos" ou "pedidos" para o sistema local, começa concretamente a indexação.

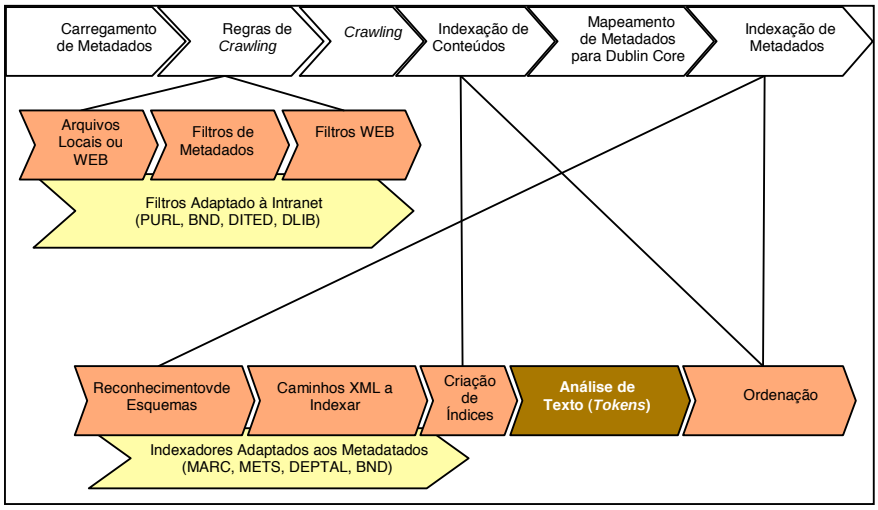

Figura 3 – Fluxo de trabalho da indexação no MITRA

A indexação tem três fases. A indexação de conteúdos passa pelo processo simples de criação de índices de termos, limpeza de *stop words,* que são palavras muito utilizadas e por isso redundantes, plurais, *stemming*, que são algoritmos para redução de uma palavra ao um prefixo máximo comum com outras do seu tipo, como por exemplo carro e carrinho, em que carro seria o prefixo máximo comum, conversões de maiúsculas em minúsculas, etc. A indexação de metadados passa por uma primeira fase de possível transformação de alguns esquemas para Dublin Core, e por fim ocorre a indexação dos mesmos.

A indexação de metadados em qualquer esquema é uma opção, podendo-se usar para isso um módulo de indexação genérica de XML. No entanto o uso dessa opção por omissão foi rejeitada por não fazer sentido sobrecarregar o LUCENE com índices que na prática não irão servir. Optou-se assim por criar uma série de componentes em JAVA que, dependendo do esquema, extraem de cada XML uma lista de pares (nome_do_indice, texto) que depois são passados ao LUCENE. Os nomes dos índices são criados com base na *expressão* XPATH que identifica o caminho para obter o texto em causa. Por exemplo, no caso do campo *creator* do formato Dublin Core, o nome do índice respectivo seria *dc.creator*. No caso da indexação genérica de XML

também teríamos de usar o *namespace* no nome do índice para garantir que não haveria nomes de índices repetidos.

Saindo um pouco da camada de indexação seria interessante de referir que o MITRA permite a adição de metadados em qualquer altura do ciclo de vida de um recurso. Quando isso é feito a indexação de metadados é chamada assincronamente ao resto do fluxo de indexação uma vez que é completamente independente.

Voltando à camada de indexação estes índices criados com base nos metadados permitem a construção de interfaces de pesquisa avançada Figura 4 como podemos ver na Figura 4 em relação ao serviço BUSCA da Biblioteca Nacional Digital.

Desta forma a camada de indexação coloca todos os conteúdos, estruturados ou livres, em vários índices sendo alguns deles muito conhecidos e usados noutros motores de busca, como por exemplo o *site,* que guarda todos os URL's indexados, a data de última modificação, etc.

Processo de Interrogação no MITRA

Todos os índices podem ser usados na construção de perguntas ao MITRA, tendo a linguagem de interrogação (*query language),* que é muito semelhante à do Google, a seguinte sintaxe que vem por defeito com o LUCENE:

Indice1:([NOT] <texto a encontrar>) [[AND|OR|NOT] [IndiceN:(<texto a encontrar>)]]

Apresentamos de seguida alguns exemplos do índice URN e de possíveis perguntas, todas equivalentes, aos índices:

urn:MITRA.bn.pt:1 – http://site1/index.html, http://site1/public/copy1/i.htm, … *urn:MITRA.bn.pt:2 – http://site2/index.html, http://site2/public/copy1/i.htm, …*

Pergunta: *URN:urn:MITRA.bn.pt:1 AND (contents:software for xml)* Pergunta: *site:http://site1 (contents:software for xml)* Pergunta: *site:http://site1 software for xml* Pergunta: *dc.creator:Saraiva AND mitologia*

O MITRA usa diversos índices que resultam da interpretação daquilo que se está a indexar, sendo que muitos deles resultam de *parsing* de HTML e outros da estrutura do material indexado. Exemplos disso são os índices site, *sumary, comments, image, imagesize,* e todos os índices resultantes da indexação de metadados.

Finalmente, na ordenação dos resultados podem-se utilizar várias formas de medição de qualificação de resultados face a uma pesquisa algoritmo de analise de ligações usado no Google, o *Page Rank* [17], ou o cálculo de *rank tfidf* [Salton88] para descobrir o peso de um documento face a uma pesquisa.

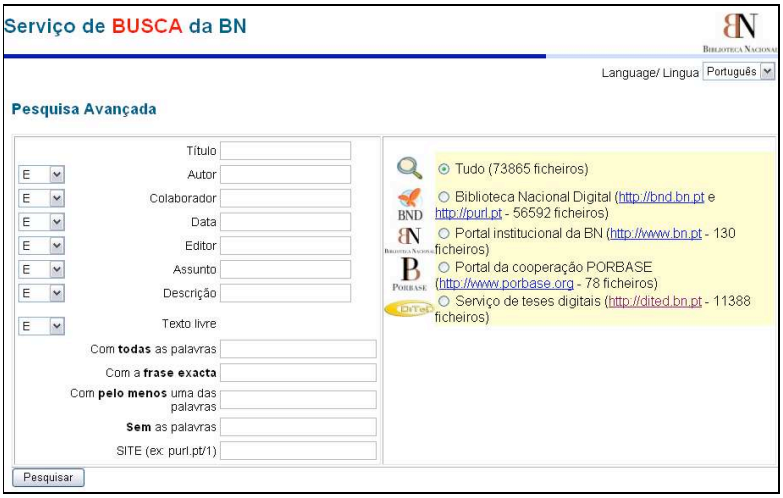

Figura 4 – Interface de pesquisa avançada recorrendo aos diferentes índices.

Serviços WEB, Interoperabilidade e Pesquisa

Quer a interface administrativa, quer a interface de pesquisa foram desenvolvidas para permitir pedidos humanos e de outras máquinas. Assim o MITRA pode ser totalmente administrado remotamente por outros sistemas como é o caso do serviço DiTeD [5], o repositório de Teses e Dissertações Digitais da Biblioteca Nacional Digital, que o usa como motor de busca interno para indexar as suas colecções. Essa interface é baseada em pedidos POST e respostas em XML, para as quais foi desenvolvido um cliente JAVA para distribuição. Além desta interface o MITRA apresenta outras interfaces standard como a SRU ou SRW [21] e [22], sendo que a SRU/SRW, que significa Search and Retrieve via URL ou via *WEB Service* é um *standard* muito utilizado para executar *queries* CQL (*Common Query Language*) [3] e responder num esquema bastante simples contendo normalmente uma lista de registos Dublin Core como resultados mas não obrigatoriamente (Figura 5).

As interfaces WEB simulam todos os serviços do MITRA, públicos e administrativos, para facilitar a interacção com outros sistemas.

No caso da interface pública de pesquisa o MITRA define uma XSLT pela qual os resultados são passados produzindo HTML. Neste passo é tirado grande partido dos metadados para garantir que os resultados são explícitos, tal como mostrado na Figura 7. Na Figura 8 mostra-se uma comparação com a mesma pesquisa dentro da BND mas agora com o sistema Google.

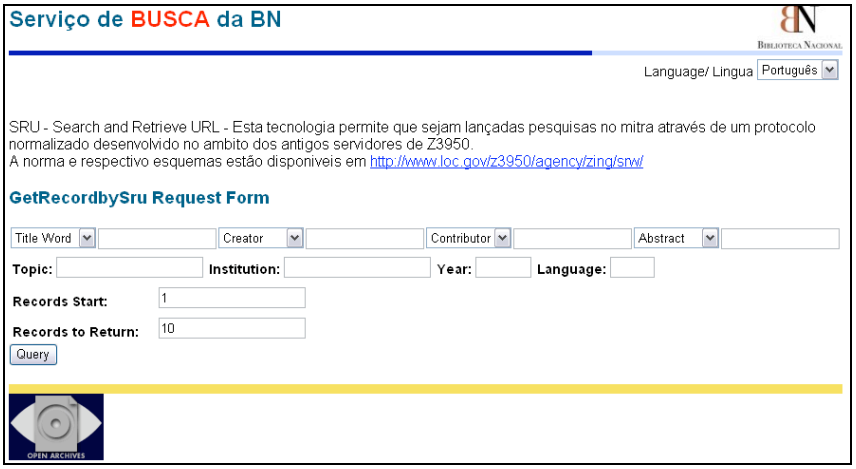

Figura 5 – Interface WEB para o servidor de SRU do serviço BUSCA (http://busca.bn.pt/jsp/sru.jsp)

Figura 6 – Interface WEB para os Serviços

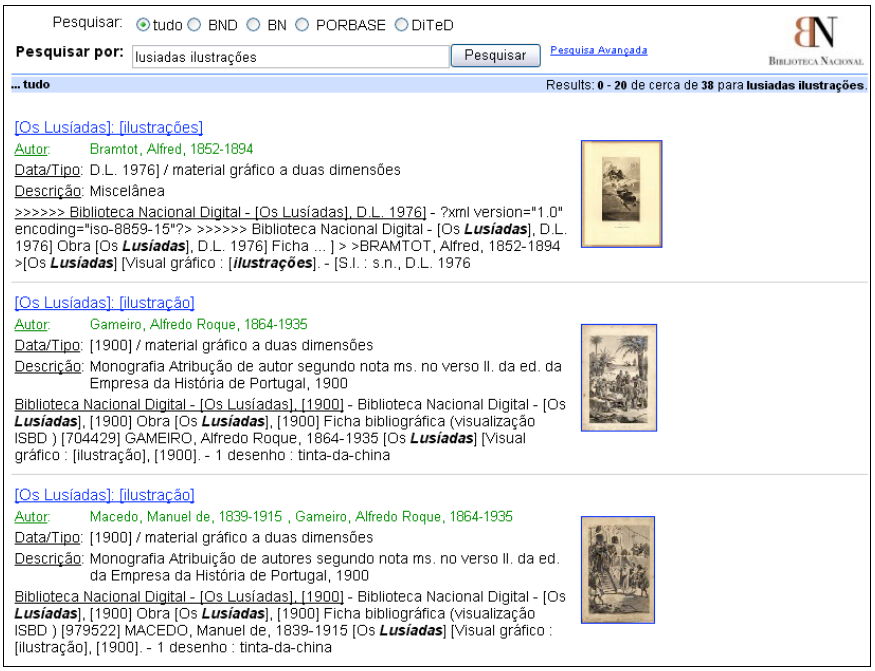

Figura 7 – Pesquisa por "lusíadas ilustrações" no BUSCA

Figura 8 – Pesquisa por "site:purl.pt lusíadas ilustrações" no Google

Suporte ao estudo de ambientes controlados

Ainda relativamente à Figura 3, o processo "Análise de Texto (*tokens*)" está destacado porque pretendemos chamar a atenção para a sua especial importância para a metodologia apresentada de seguida.

A quarta camada do MITRA dá suporte à análise dos espaços indexados. Se esta for correcta, permite a extracção de conhecimento que poderá ser uma mais valia para outros processos de análise ou de decisão. O problema que se coloca é como analisar o domínio que estamos a indexar. Os conteúdos digitais de uma organização são uma fonte rica de informação que bem trabalhada poderá fazer surgir conhecimento escondido. O grande problema é que os sistemas de informação apenas extraem conhecimento de dados estruturados e muitas vezes apenas em bases de dados relacionais. Pedir mais que isto é ainda hoje pedir muito. O método apresentado de seguida para a análise de domínios com o suporte do MITRA está a ser testado pela primeira vez num estudo aos conteúdos da revista D-LIB [6], a qual tem acessíveis todos os artigos desde 1996, estruturados por mês e ano. Achamos no entanto que este método pode ser generalizado a qualquer domínio.

Como fazê-lo?

O grande objectivo do processo é encontrar os *termos chave* do domínio, compostos por uma ou mais palavras. Para isto, problemas como por exemplo plurais ou sinónimos têm de ser considerados. Para lidar com esta problemática de identificação de palavras-chave criámos no MITRA o conceito *keyword* que pode ser composto por um ou mais *termos chave*, podendo estes conter uma ou mais palavras. Por outro lado temos o índice de palavras existentes no domínio. O objectivo final será maximizar a lista de *termos chave* e minimizar o índice de palavras soltas do índice de palavras. Desta forma vamos identificar todos os objectos importantes do nosso universo de estudo.

Para atingir esse objectivo o sistema de suporte existem mais duas listas que podem ser definidas no MITRA, as s*top-words* e as palavras *irrelevantes*. As palavras *irrelevantes* são palavras soltas que estão numa situação intermédia em que não se tem a certeza se não poderão vir a ser *keyword*. As *stop words* são as palavras descartadas que já não conseguem formar qualquer *keyword*.

Para gerir este processo foi desenvolvido um serviço com a interface que se mostra na Figura 9.

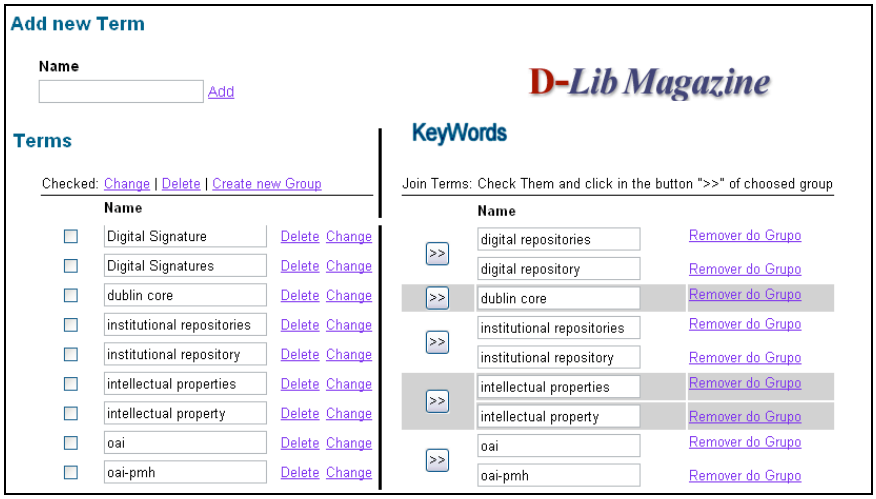

Figura 9 – Interface para análise de texto.

Método de Análise

A análise do domínio terá de ser feita de forma interactiva e com intervenção humana de especialistas. O MITRA disponibiliza no entanto algumas rotinas que permitem a criação de listas de possíveis candidatos a *keywords* combinando as palavras do índice duas a duas e três a três, ordenando-as pelo número de referências ao termo composto.

Assim a cada interacção descobrem-se mais *keywords* e reindexa-se o domínio. Nesse processo o analisador de texto vai descartando as *stopwords* e o universo de palavras vai diminuindo e facilitando cada vez mais o trabalho de análise. O Analisador de Texto (Figura 3) é uma peça chave que se utiliza em todo lado na camada de indexação, incluído na analise das *queries* para as pesquisas. Ele limpa o texto de *stop words*, letras maiúsculas, diacríticos, faz o *stemming* e neste caso normaliza também as *keywords* substituindo as ocorrências dos termos pelo código da *keyword* respectiva.

Recuperação da Informação na D-LIB

Construída a base de *keywords* cada domínio terá os seus requisitos de conhecimentos, os quais irão requerer módulos de recuperação diferentes.

No caso da D-LIB todos os conteúdos estão organizados de forma cronológica. Para auxiliar o processo foram criados dinamicamente descrições de metadados, onde entre outras coisas, está descrito o Mês e o Ano de cada artigo. Essas descrições especialmente criadas para descrever uma estrutura que se evidenciava numa

disposição de arquivos locais, foram associadas aos registos, juntamente com as descrições em Dublin Core.

Combinado o índice de *keywords* com os índices de informação estruturada como os autores, assuntos ou títulos, poderemos chegar a informação bastante relevante para a análise das bibliotecas digitais e sua evolução. Um exemplo são as datas de publicação que nos podem ajudar a descobrir quando é que determinadas tecnologias ou assuntos foram mais abordados, quem foram os autores que os abordaram em primeira instância, ou aqueles que mais abordaram determinados temas.

A parte desta informação se considerarmos fazer dos nomes de autores *keywords* onde criamos um conjunto de todas as assinaturas de determinada pessoa podemos ainda extrair listas de artigos por autor, por tema, por data, etc.

Por observação gráfica do número de referências a certas *keywords* podemos por exemplo perspectivar os temas que irão ser mais abordados nos próximos anos e quais estão em decadência como se pode visualizar na Figura 10.

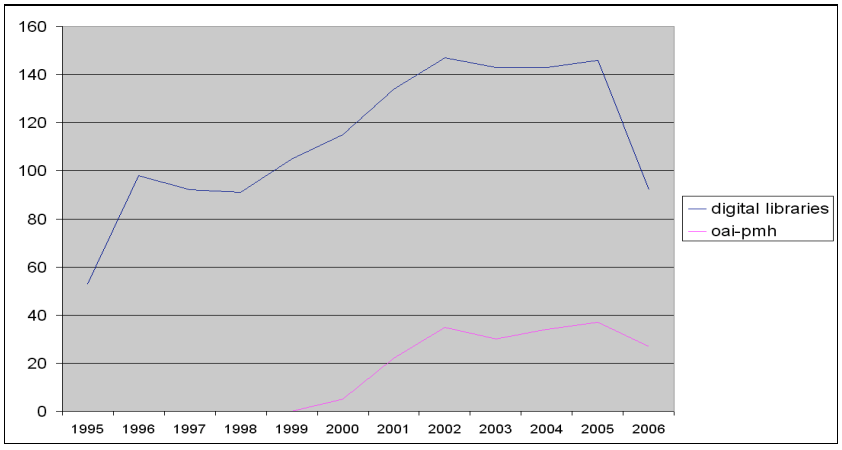

Figura 10 – Exemplo da contagem de referências a duas keywords na revista D-LIB

Para além de tudo isto poderemos ainda filtrar resultados usando apenas determinados autores em casos que se verifique quem são os melhores em determinados assuntos de forma a melhorar os resultados, etc.

Conclusões e Trabalho Futuro

Este artigo descreve um simples e efectivo motor de busca desenvolvido para gerir colecções de recursos estruturados em linha. Esta solução é usada como suporte ao serviço de pesquisa genérico na Biblioteca Nacional (http://busca.bn.pt) e ainda nos serviços de pesquisa especializados do Repositório de Teses da Biblioteca Nacional (http://dited.bn.pt). Foi criado ainda um serviço experimental de indexação da revista

DLIB (recurso localizado em http://www.dlib.org, e serviço em http://MITRA.bn.pt/dlib).

O serviço na BN reconhece neste momento esquemas de metadados, os quais são usados em módulos de filtragem semântica, melhoria da apresentação dos resultados, e suporte à pesquisa avançada. Entre esses esquemas destacam-se os formatos MARCXML, Dublin Core, METS e DEPTAL. Com as descrições em METS, por exemplo, conseguimos descobrir por exemplo os objectos representando obras digitalizadas que possuem conteúdos resultantes de processos de OCR automáticos (texto em ficheiros) e associar esses conteúdos às respectivas imagens digitalizadas, permitindo assim oferecer um efeito de "pesquisa de texto em imagens". Finalmente, as funções de análise de domínios controlados estão a ser usadas em projectos de caracterização de colecções e espaços, o que se espera venha a produzir resultados bastante interessantes, a reportar no futuro.

Referências

[1] BND. Biblioteca Nacional Digital. <http://bnd.bn.pt>
[2] BUSCA. <http://busca.bn.pt>
[3] CQL (Common Query Language) <http://www.loc.gov/standards/sru/cql/index.html>
[4] DEPTAL. Repositório Institucional. <http://deptal.bn.pt>
[5] DiTeD. Digital Thesis and Dissertations. <http://dited.bn.pt>
[6] D-LIB. <http://www.dlib.org>
[7] HIBERNATE. <http://www.hibernate.org/>
[8] HTTRACK. <http://www.httrack.com>
[9] LDAP. <http://www.openldap.org/>
[10] LUCENE. Apache LUCENE. <http://lucene.apache.org/>
[11] MARCXML. MARC 21 XML Schema. <http://www.loc.gov/standards/marcxml/>
[12] METS. Metadada Encoding and Transmission Standard. <http://www.loc.gov/standards/mets>
[13] MITRA. <http://mitra.bn.pt>
[14] MYSQL < http://www.mysql.com/>
[15] Nyna Plat, Search Engines for Intranets <http://www.llrx.com/features/nina.htm>
[16] OAI-PMH. Open Archives Initiative - Protocol for Metadata Harvesting <http://www.openarchives.org/>
[17] Page Rank <http://www.google.com/technology/>
[18] PORBASE. Base Nacional de Dados Bibliográficos. <http://www.porbase.org>
[19] PURL sistema de gestão dos recursos digitais da BND. <http://purl.bn.pt>
[20] SCORM. The Sharable Content Object Reference Model. <http://www.adlnet.org/index.cfm?fuseaction=scormabt>
[21] SRU Search and Retrieve URL. <http://www.loc.gov/z3950/agency/zing/srw>
[22] SRU/SRW <http://www.loc.gov/standards/sru/>
[23] SRU/SRW schema <http://www.loc.gov/standards/sru/xml-files/srw-types.xsd>
[24] STRUTS. The Apache STRUTS Web Application Framework. <http://struts.apache.org/>
[25] The Dublin Core Metadata Initiative <http://dublincore.org>
[26] The Google Pagerank Algorithm and How It Works, Ian Rogers, IPR Computing Ltd. <http://www.iprcom.com/papers/pagerank/>
[27] UNIMARC <http://www.unimarc.info>
[28] Z39.50. Z39.50 Maintenance Agency. <http://www.loc.gov/z3950/agency/>

XMLaligner: Exploração de Corpora Paralelos

Ivo Anjo and David Martins de Matos

L^2F – Laboratório de Sistemas de Língua Falada
INESC-ID Lisboa/Instituto Superior Técnico/Universidade Técnica de Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{ivo.anjo,david.matos}@l2f.inesc-id.pt
http://www.l2f.inesc-id.pt/

Resumo Apresenta-se a aplicação XMLaligner, que permite a consulta e manipulação de corpora multilingues estruturados, sendo o seu principal objectivo o de fornecer uma forma simples, mas poderosa, de navegação num corpus paralelo descrito a múltiplos níveis. Tendo em consideração que um corpus pode ser constituído por milhares de documentos, a aplicação, concebida como um "browser" gráfico, permite, além da consulta simultânea e alinhada de documentos do corpus, para cada par de línguas, efectuar pesquisas no contexto desses ficheiros. A ferramenta potencia a utilização de corpora paralelos em aplicações como tradução automática e projecção de características entre vários conjuntos de línguas.

1 Introdução

No processamento das línguas humanas é necessário o recurso a colecções de textos – corpora – que providenciam material observável, a partir do qual se derivam regras e modelos estatísticos que permitem construir aplicações como processadores sintácticos ou sistemas de tradução automática (que podem ser treinados a partir de corpora paralelos). Os corpora multilingues podem ainda ser úteis em aplicações de lexicografia e na projecção de características, descritas para uma dada língua, para outra onde elas não estão ainda descritas, o que os torna úteis na aquisição de informação linguística sobre a língua alvo.

Uma colecção de corpora paralelos é composta pelos documentos nas várias línguas e pela descrição dos alinhamentos entre as línguas. Apesar de ser possível a codificação destas colecções numa grande variedade de formatos, desde bases de dados até ficheiros textuais, são as representações hierárquicas as que melhor combinam a flexibilidade com a simplicidade. É neste ponto que normas como o SGML [1] e, mais recentemente, o XML [2] assumem um papel importante na codificação de documentos.

A utilização de corpora para os fins expostos pressupõe, contudo, a capacidade de extrair informação relevante, seja automática ou manualmente. Ainda que na análise automática existam processos eficazes para sintetizar vários tipos de informação, na análise manual assumem importância não trivial as limitações humanas, seja na manipulação das grandes quantidades de informação não agregada, que é a natureza habitual de um corpus, seja na gestão dos múltiplos componentes (ficheiros).

1.1 Objectivos

Com vista a facilitar a interacção com colecções paralelas, definiu-se o objectivo de construir uma ferramenta que permitisse não só a gestão de ficheiros das colecções alinhadas, mas também a navegação paralela e a execução de consultas no contexto dos textos alinhados. Um outro objectivo foi o de conseguir fornecer a funcionalidade através de uma interface simples de utilizar.

1.2 Estrutura do documento

Considerando a motivação e objectivos expostos, foi criada a aplicação XMLaligner. A secção 2 descreve o contexto de trabalho previsto para a aplicação, em particular, no que respeita ao processamento computacional da língua baseado em corpora, focando-se os corpora estruturados baseados em XML (secção 2.1) e alguns casos concretos (secção 2.2). A secção 3 apresenta a arquitectura e o desenvolvimento do protótipo da aplicação. Finalmente, na secção 4 apresentam-se algumas conclusões e direcções para evolução.

2 Contexto

A motivação para o trabalho advém das necessidades sentidas na aplicação ao português de algoritmos e dados de análise de discurso da língua inglesa [3]. A falta de recursos linguísticos equivalentes torna impossível a tarefa, pelo que surgiu a ideia de projectar a informação disponível para o inglês através de um corpus paralelo. Este corpus possibilita a extracção de informação linguística para processamento aos níveis da semântica e do discurso, permitindo a aplicação final do algoritmo ao português.

2.1 Corpora XML alinhados

Considerando que a informação estrutural dos textos é importante para a análise de discurso, foi definido como requisito que o corpus de trabalho fosse estruturado e que, se possível, admitisse marcações adicionais sem perturbações na estrutura existente. Embora estes requisitos pudessem ser satisfeitos por vários formatos, a utilização de XML é suficientemente simples e flexível para a tornar a mais atractiva: é possível manter a mesma estrutura e enriquecê-la progressivamente, à medida que são aplicados novos algoritmos ao material de trabalho (e.g., marcação de estruturas retóricas).

Além do requisito básico de utilizar XML, optou-se pela escolha de um formato normalizado, potenciando a reutilização. Foi escolhido o formato TEI[1] [4]. A escolha foi em parte motivada pela prévia utilização de corpora neste formato, mas o processo é aplicável a outros modelos com fins semelhantes, dos quais se pode realçar o XCES[2] [5].

A definição dos alinhamentos, possíveis a vários níveis, e.g. ao parágrafo, ao segmento, ou mesmo à palavra, está dependente das necessidades específicas das aplicações

[1] Text Encoding Iniciative.
[2] Corpus Encoding Standard for XML.

em estudo. Os casos de interesse, no contexto em que este trabalho se insere, contemplam desde alinhamentos ao segmento, mapeando segmentos de discurso entre duas línguas, até alinhamentos à palavra, em que são estabelecidas relações entre palavras ou grupos de palavras das duas línguas.

2.2 O Corpus JRC-Acquis

O corpus JRC-Acquis é constituído por um conjunto de textos de legislação comunitária da União Europeia, escritos entre 1950 e 2005. Esta colecção de textos legais é composta por aproximadamente 8000 documentos e cobre uma gama variada de domínios. O corpus está disponível nas 20 línguas oficiais da União (2005)[3]. Os países que passaram a integrar a União a partir de 2007 (Bulgária e Roménia), assim como outros que estão em processo de adesão (Croácia), já iniciaram a tradução da legislação, embora as traduções nas novas línguas ainda não façam parte do corpus.

Da introdução dada acima aos corpora paralelos, é clara a utilidade de um corpus desta natureza para aplicações de investigação em linguística. Se se considerar a dimensão em número de documentos e línguas disponíveis, este é um dos maiores corpora paralelos de acesso público (considerando o número de línguas): o texto base do corpus, assim como outra legislação comunitária, estão disponíveis nos servidores web da Comissão Europeia. Este material foi convertido para XML e alinhado à frase, através de relações n-para-n. Os alinhamentos são específicos para cada par de línguas alinhadas, tendo sido produzidas para todos os pares de combinações possíveis (em lugar de se utilizar uma língua pivot), embora existam algumas excepções. O corpus não foi corrigido manualmente [6,7].

3 XMLaligner

Considerando que o requisito base da aplicação era a visualização simultânea de documentos XML, foi necessário desenvolver toda a base de processamento XML da aplicação. Esta funcionalidade básica foi depois reutilizada na implementação de outras funções. Para além do requisito base, foram identificados outros aspectos de interesse, como a definição de modos de visualização – apenas parágrafos alinhados, parágrafos alinhados e seus parágrafos vizinhos, todo o documento visível – e capacidades de procura básica nos documentos actuais.

Foi ainda identificado um requisito importante em termos de interface gráfica: teria de ser facilmente utilizável por diferentes tipos de utilizadores, i.e., não poderia exigir conhecimento da estrutura da aplicação ou do formato XML do corpus.

3.1 Arquitectura

A arquitectura da aplicação possui três níveis (figura 1): apresentação e interacção, controlo e processamento do corpus. O primeiro nível corresponde à interface (gráfica)

[3] Checo, Dinamarquês, Alemão, Grego, Inglês, Espanhol (Castelhano), Estónio, Finlandês, Francês, Húngaro, Italiano, Lituano, Letão, Maltês, Holandês, Polaco, Português, Eslovaco, Esloveno e Sueco.

com o utilizador, o segundo controla o acesso aos documentos de forma alinhada e o terceiro contém os processadores XML de alinhamentos e de documentos do corpus.

As classes correspondentes aos processadores podem ser substituídas sem afectar adversamente o funcionamento geral da aplicação. Desta forma, é possível a extensão a outros formatos de corpora.

Além do requisito de apresentação dos dois conjuntos de parágrafos/documentos alinhados, a interface gráfica apresenta também a lista de documentos para o conjunto de línguas seleccionado e listas com as línguas disponíveis. É também revelada na interface alguma da estrutura dos ficheiros de alinhamentos, permitindo ver directamente a correspondência dos parágrafos, tal como é lida pelo processador de XML.

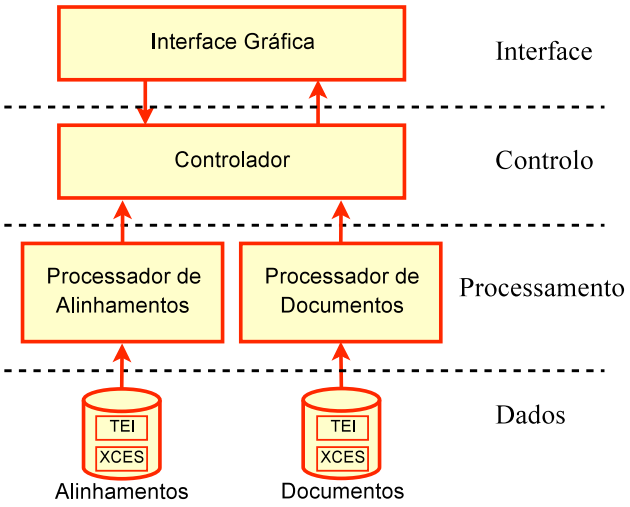

Figura 1. Arquitectura da aplicação.

As figuras 2 e 3 mostram exemplos, respectivamente, do ficheiro de alinhamentos e de um documento do corpus (ambos segundo o formato TEI).

O ficheiro de alinhamentos possui, para cada documento, uma secção `div` identificada com o código desse documento, e em que são listados o tipo de um alinhamento e os seus parágrafos-alvo. O atributo `type` indica quantos parágrafo(s) do documento na primeira língua correspondem ao(s) do documento na segunda língua; o atributo `xtargets` indica quais os números dos parágrafos referidos. A aplicação utiliza estes atributos para identificar os parágrafos que deverá mostrar, além de disponibilizar também esta informação ao utilizador.

A estrutura principal de um documento do corpus é um conjunto de parágrafos numerados, os quais são referidos nos alinhamentos. Os estádios de processamento

```
<TEI.2 id="jrc-en-pt" select="en pt">
  <teiHeader lang="en" date.created="2006-05-14">
  ...
  </teiHeader>
  <text id="jrc-en-pt." select="en pt">
    <body>
      <div type="body" n="21970A0720(01)" select="en pt">
        <p>35 paragraph links:</p>
        ...
        <link type="1-1" xtargets="21;18"/>
        <link type="2-2" xtargets="22 23;19 20"/>
        <link type="2-1" xtargets="24 25;21"/>
        ...
      </div>
    </body>
  </text>
</TEI.2>
```

Figura 2. Estrutura básica de um ficheiro de alinhamentos.

```
<TEI.2 id="jrc21970A0720_01-pt" n="21970A0720(01)" lang="pt">
  <teiHeader lang="en" date.created="2006-05-14">
  ...
  </teiHeader>
  <text id="jrc21970A0720_01-pt." lang="pt">
    <body>
      <p n="2">ACORDO COMPLEMENTAR ao ≪Acordo relativo aos
        Produtos de Relojoaria entre a Comunidade Económica
        Europeia e os seus Estados-membros e a Confederação
        Suíça≫ (1)</p>
      <p n="3">O CONSELHO DAS COMUNIDADES EUROPEIAS,</p>
      <p n="4">por um lado,</p>>
      <p n="5">O CONSELHO FEDERAL SUÍÇO,</p>
      ...
    </body>
  </text>
</TEI.2>
```

Figura 3. Estrutura básica de um documento do corpus.

permitem aceder a todos ou apenas a alguns dos parágrafos do documento, indexados pelo seu conteúdo ou número.

3.2 Construção do protótipo

O desenvolvimento inicial dos níveis de processamento começou com uma implementação SAX (Simple API for XML) do processador de ficheiros de alinhamentos. Esta implementação utilizava directamente estruturas nativas da biblioteca de suporte (`QTreeWidget` e `QTreeWidgetItem`), permitindo simultaneamente guardar os dados ao serem processados e a sua visualização directa. Foi utilizado este tipo de processamento, pois permitia um melhor controlo sobre a análise dos dados de entrada, além de possuir uma API mais directa e simples que a alternativa DOM (Document Object Model). O inconveniente da abordagem baseada em SAX era obrigar a aplicação a gerir e manter estruturas de dados para guardar a informação lida dos ficheiros XML: esta informação tinha de ser estruturada e a sua hierarquia facilmente navegável, algo que era necessário implementar explicitamente na solução SAX.

Para avaliar as capacidades do DOM, realizou-se directamente em DOM a primeira versão do processador dos documentos do corpus, em lugar de se produzir uma versão SAX, como tinha acontecido no caso dos alinhamentos. Algumas das vantagens encontradas no DOM para o caso em estudo foram:

- Os dados lidos serem automaticamente organizados numa estrutura em árvore;
- Classes DOM gerirem a criação da árvore em memória e disponibilizarem uma API para a consultar e alterar, não sendo necessário que a aplicação implemente as suas próprias estruturas para fornecer comportamento equivalente;
- As possibilidades de expansão futuras para a edição de documentos, além de consulta, sendo que o DOM permite a edição da árvore em memória, assim como a sua escrita directa para um ficheiro.

Apesar dos aspectos positivos, é, contudo, de referir que o facto de ser necessário construir e manter toda a árvore em memória pode ser uma desvantagem: ficheiros como os de alinhamentos, com um elevado número de elementos (normalmente cerca de 300.000 linhas), podem causar problemas de ocupação de memória; no caso do SAX, tem-se um maior controlo do que guardar em memória ou simplesmente descartar, o que o torna potencialmente mais eficiente, em termos de tamanho dos dados de trabalho. Para a aplicação XMLaligner, as vantagens do DOM compensam as desvantagens, pelo que todo o processamento é baseado nesta API, tendo sido reimplementada a versão inicial (SAX).

A aplicação permite ter carregado um ficheiro de alinhamentos de cada vez, podendo ser carregado outro em qualquer altura. Os documentos do corpus são carregados em memória apenas quando são consultados pelo utilizador, e descarregados assim que é escolhido um novo par. Mesmo assim, uma instância habitual desta aplicação consome cerca de 220MB de memória, o que seria expectável, tendo em conta que todos os ficheiros XML são processados com DOM e mantidos inteiramente em memória.

A pesquisa, que funciona apenas dentro do par de documentos carregado actualmente, permite a procura utilizando múltiplas palavras ou expressões regulares.

Face a outras aplicações semelhantes que correm online a partir do browser, as vantagens desta aplicação/protótipo são o trabalhar directamente sobre os dados, e permitir um nível de interactividade que seria mais difícil de obter numa aplicação baseada na web.

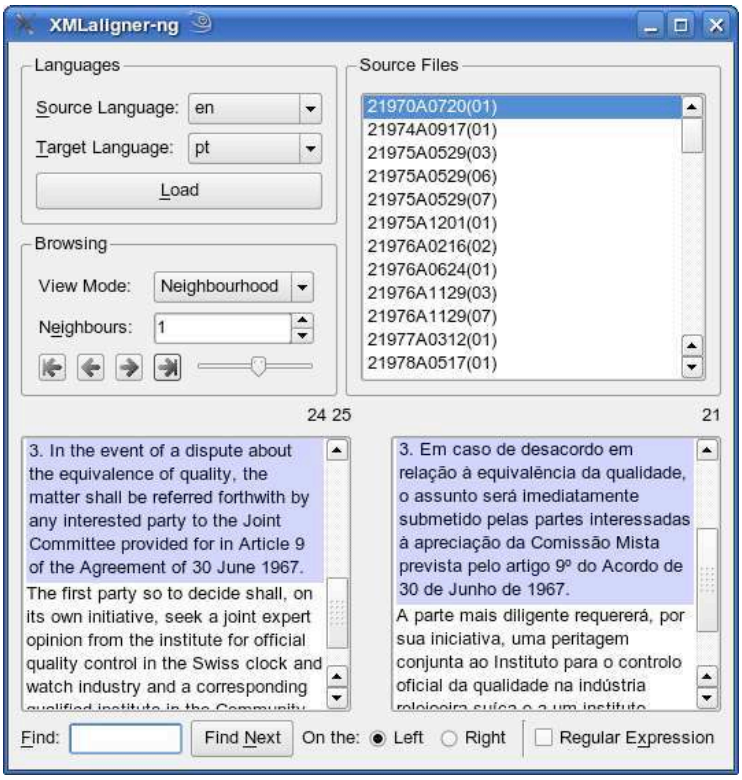

Figura 4. Protótipo apresentando alinhamentos entre as versões inglesa e portuguesa de um documento do corpus.

Para a realização da aplicação foi escolhida a biblioteca Qt 4 [8]. Este pacote providencia um conjunto extenso de componentes que permite o rápido desenvolvimento de aplicações gráficas multi-plataforma (Linux/Unix, Windows, MacOS X), que são utilizados na interface gráfica, para controlo da aplicação e apresentação de dados ao utilizador. O módulo `QtXml` contém implementações em C++ de SAX 2 e DOM (nível 2), usadas como base para os vários processadores, facilitando o seu desenvolvimento.

4 Conclusões

4.1 Protótipo

O protótipo, no seu estado actual, permite toda a navegação gráfica no corpus, incluindo escolha de alinhamentos e de documentos, procura e modos de visualização. Apesar de ainda ser apenas um protótipo, com a funcionalidade actual, já permite satisfazer parcialmente alguns dos requisitos para permitir trabalho linguístico.

4.2 Evolução

Na continuação do desenvolvimento do protótipo actual, além de melhoramentos estruturais, está planeado (i) o suporte a outros formatos de corpora, além do actual TEI; e (ii) visualização gráfica da estrutura XML dos documentos e respectiva edição.

Para desenvolvimentos futuros, além da funcionalidade presente no protótipo actual, consideram-se várias opções. A primeira é a implementação de alinhamentos a múltiplos níveis (e.g., parágrafos, segmentos e palavras), quando os corpora os possuirem, e alinhamentos de marcas, e.g., categorizações. A segunda direcção de evolução é a de providenciar métodos de busca mais sofisticados (e.g., categorias, buscas combinadas e concordâncias): os actuais métodos apenas permitem pesquisas num dos documentos de cada vez e apenas numa das línguas de cada vez. Finalmente, seria interessante considerar a ligação da aplicação de visualização a algoritmos para modificação do corpus e produção da correspondente informação de alinhamento.

Agradecimentos

Este trabalho foi parcialmente financiado pelo Projecto NLE-GRID: Natural Language Engineering on a Computational Grid (POSC/PLP/60663/2004).

Referências

1. ISO (International Organization for Standardization): ISO 8879:1986 – Information processing – Text and office systems – Standard Generalized Markup Language (SGML) (1986)
2. World Wide Web Consortium: Extensible Markup Language (XML) (2006) `http://www.w3.org/XML/`.
3. Marcu, D.: The Rhetorical Parsing of Unrestricted Texts: A Surface-Based Approach. Computational Linguistics **26**(3) (2000) 395–448
4. TEI Consortium: TEI – The Text Encoding Initiative (2006) `http://www.tei-c.org/`.
5. XCES: Corpus Encoding Standard for XML (2006) `http://www.xml-ces.org/`.
6. European Commission Joint Research Centre (JRC): JRC-ACQUIS Multilingual Parallel Corpus (2006) `http://wt.jrc.it/lt/Acquis/`.
7. Pouliquen, B., Steinberger, R.: The Acquis Communautaire Corpus. In: JRC Enlargement and Integration Workshop: "Exploiting parallel corpora in up to 20 languages", Arona, Italy, European Commission Joint Research Centre (JRC) (2005)
8. Trolltech A.S.A.: Qt – Cross-Platform C++ Development Framework (2006) `http://www.trolltech.com/products/qt/`.

Ferramentas para a Construção de Arquivos Digitais de História Oral*

Silvestre Lacerda[1]
lacerda@iantt.pt
Norberto Lopes[2], Nelma Moreira[2], Rogério Reis[2]
{nml,nam,rvr}@ncc.up.pt

[1] Instituto dos Arquivos Nacionais / Torre do Tombo
[2] DCC-FC& LIACC, Universidade do Porto

Resumo Neste trabalho, apresentamos um conjunto de ferramentas, baseadas em tecnologias `XML`, para a produção, indexação, classificação e pesquisa de documentos multimédia associados a entrevistas. Estas ferramentas foram desenvolvidas no âmbito do projecto de história oral da Universidade Popular do Porto cujo objectivo é a preservação e divulgação da memória social e laboral do Porto no séc. XX.

Palavras chave: especificações de linguagens `XML`, indexação, editores de `XML`, *metadados*, *thesaurus*

1 Introdução

A produção de documentos estruturados e anotados é essencial para permitir a disponibilização e pesquisa de informação em formatos digitais e em particular na `Web`. Para tal é necessário ter definidas especificações de cada tipo de documento e ter disponíveis editores que facilitem a sua criação e modificação.

Neste trabalho, apresentamos um conjunto de ferramentas, baseadas em tecnologias `XML`, para a construção, indexação, classificação e pesquisa de documentos multimédia associados a entrevistas. Estas ferramentas foram desenvolvidas no âmbito do projecto de história oral da Universidade Popular do Porto (**UPP**) cujo objectivo é a preservação e divulgação da memória social e laboral do Porto no séc. XX.

A cada documento foi associada metainformação, usando especificações do *standard Dublin Core* adaptadas para o tratamento dos vários tipos de documentos relacionados com uma mesma entrevista: áudio, vídeo, transcrição, resumo, etc.

A indexação e classificação dos documentos, com recurso a palavras chave ou a palavras no texto é habitualmente feita com auxílio de vocabulários controlados, e em especial de *thesauri*. Neste contexto, foi também desenvolvida uma

* Trabalho parcialmente financiado pela Fundação para a Ciência e Tecnologia (FCT) e Programa POSI.

especificação XML para *thesauri*, assim como um conjunto de ferramentas para a sua consulta e pesquisa.

Este trabalho está organizado do seguinte modo. Na secção seguinte é apresentado o Centro de Documentação e Informação da Universidade Popular do Porto, no âmbito do qual este trabalho se insere. São aí também descritos alguns dos objectivos a atingir para a construção de uma arquivo digital de acesso livre. Na Secção 3 são apresentadas as especificações das linguagens XML usadas para alguns dos documentos associados às entrevistas. Na Secção 4 são descritos os editores de documentos que foram implementados e, na Secção 5, um interface de pesquisa nos documentos. Na Secção 6 são descritas ferramentas para a consulta e pesquisa a *thesauri*. Finalmente, na Secção 7 é indicado algum trabalho em curso e futuro.

2 Centro de Documentação e Informação sobre o Movimento Operário e Popular do Porto

O Centro de Documentação e Informação sobre o Movimento Operário e Popular do Porto (**CDI**) da **UPP** [19,20] foi criado em 2001 (com o apoio da Sociedade Porto 2001 e em colaboração com a União de Sindicatos do Porto) e tem como principais objectivos:

- contribuir para a preservação da memória e da história oral e social do Porto, valorizando o seu património social e as suas identidades;
- coligir, tratar e difundir informação sobre o movimento popular e de trabalhadores do Porto e apoiar e estimular o estudo sobre ele;
- identificar e conhecer o património arquivístico de sindicatos e outras organizações de trabalhadores do Porto, através do levantamento, diagnóstico e inventário dos seus arquivos, incluindo o levantamento da informação sobre núcleos documentais custodiados por outras instituições públicas ou privadas;
- recolher, em suporte áudio e vídeo, testemunhos e histórias de vida de pessoas que protagonizaram e/ou vivenciaram acontecimentos sociais representativos da vida social e laboral da cidade ao longo do século XX;
- disponibilizar informação relevante sobre condições de trabalho, lutas sociais, associações de trabalhadores e organizações populares, vivências das ilhas e dos bairros sociais, práticas culturais mais relevantes da "cidade do trabalho"a partir de depoimentos dos entrevistados;

O projecto "Memórias do trabalho - testemunhos do Porto laboral no século XX"enquadra o trabalho desenvolvido para preservação da memória da história oral. Existem actualmente 80 entrevistas realizadas, com uma duração total superior a 260 horas, de trabalhadores de diferentes profissões e com diferentes experiências de intervenção social, muitos deles dirigentes sindicais, activistas de associações locais, activistas políticos e presos políticos antes do 25 de Abril de 1974.

As narrativas de vida recolhidas constituem um acervo documental ímpar e de grande importância para a investigação do movimento dos trabalhadores, sobre as suas lutas, práticas culturais e condições de trabalho, no Porto no séc. XX. Cada narrativa foi registada em formatos áudio e vídeo, transcrita e elaborado um resumo que pode ser disponibilizado na `Web`. Embora alguns dos documentos multimédia possam ser de acesso restrito é importante a sua catalogação. Para os restantes e, em especial os documentos textuais. é essencial a sua estruturação e anotação para o acesso, pesquisa e classificação. Desde a criação do **CDI**, em 2001, foi muito clara a opção de utilizar documentos em formatos abertos e baseados em especificações `XML`. Quer as transcrições, quer os resumos e quer os restantes documentos disponibilizados na *página* `Web` do **CDI** referentes ao cadastro dos arquivos das organizações, são anotados e podem ser pesquisados por texto ou por palavras pertencentes a uma indexação pré-definida.

Para a análise e investigação do acervo, pretende-se agora melhorar o acesso à informação já disponibilizada e criar ferramentas que ajudem a construção e edição dos documentos e dos arquivos digitais associados.

Alguns dos objectivos são:

- Permitir pesquisas mais elaboradas, pelo uso de vocabulários controlados, em especial *thesauri* na área das ciências socais e políticas.
- Permitir resultados de pesquisa mais complexos. Por exemplo: extracção de todas as referências a indústrias têxteis numa dada zona e período temporal.
- Permitir a introdução de comentários associados a segmentos de texto, num sistema aberto a qualquer interessado. Os comentários deverão ser *guardados* numa camada diferente, de modo a não alterar o texto original.
- Organizar os documentos associados a uma dada entrevista pela sua catalogação num sistema semelhante ao usado pelo ISLE (International Standard for Language Engineering) [9], para recursos multimédia para estudos linguísticos.
- Construção de um arquivo digital de acesso livre, de acordo com os princípios da OAI (Open Archives Initiative) [10], pelo uso de *metadados* e pela implementação de protocolos de recolha de informação (OAI-PHM).

De salientar ainda a utilização e o desenvolvimento de ferramentas informáticas em *software* livre. Sendo assumido que a informação contida nos documentos produzidos se pretende mais durável do que a duração habitual dos suportes de *hardware* e *software*, só com a possibilidade de ter os códigos fonte (e os poder alterar) é garantido o acesso a este espólio no futuro.

3 Especificações de Linguagens `XML`

Nesta secção descrevemos um conjunto de especificações de linguagens `XML` para diversos documentos. Todos os documentos, inclusive os multimédia, devem ter um registo de metainformação que permita a sua catalogação. A sua especificação é apresentada na Secção 3.1. Sobre cada documento de texto é possível fazer anotações que serão usadas na pesquisa ou classificação, ou constituírem

comentários que poderão ser úteis na análise. Na Secção 3.2 é descrita uma especificação para as anotações. Nas secções seguintes são apresentadas as especificações para as transcrições e os resumos das entrevistas.

O esquema de `XML` usado para definir cada linguagem foi o `Relax NG` [21]. Para além do seu poder expressivo tem uma sintaxe muito elegante e é baseado numa formalização teórica de autómatos de árvore que permite implementações claras e eficientes. Este esquema manipula elementos, atributos e texto ao mesmo nível, permitindo um uso flexível de padrões.

3.1 Descrição de *metadados* usando o *standard Dublin Core*

Para a descrição de *metadados* foram usados os 15 elementos base do *standard Dublin Core* [11], nomeadamente:

Título (`title`)
Autor (`creator`)
Editor (`publisher`)
Assunto (`subject`) Cada valor deve ser uma palavra-chave obtida por consulta dum *thesaurus*.
Colaborador (`contributor`) Para cada colaborador deve ser indicada a sua função.
Data (`date`) As datas serão associadas à criação, modificação e disponibilização do documento.
Descrição (`description`) Explicação ou resumo do conteúdo do documento.
Tipo (`type`) No caso dos resumos e transcrições será `texto`.
Formato (`format`) Uma descrição do tipo de média.
Idioma (`language`)
Fonte (`source`)
Relação (`relation`) Para além das relações habituais de pertença (`isPartOf`, `hasPart`, `isRequired`, etc.), dever-se-á relacionar cada documento com a entrevista que lhe deu origem.
Direitos (`rights`)
Âmbito (`coverage`)

Cada um destes elementos pode ser opcional ou ser repetido. Para a especialização dos elementos e para o preenchimento do conteúdo de cada elemento, optou-se pelo uso de atributos e a utilização de vocabulários controlados, em geral com especificação `XML`. Para os atributos, seguiu-se o método proposto pela comunidade OLAC (Open Language Archives Community) [16,3] de usar atributos correspondentes às qualificações `refinamento`, `codificação` e `idioma`, respectivamente `refine`, `code` e `lang`.

3.2 Anotações

Para a indexação e pesquisa nos documentos de texto, é conveniente anotar, de modo manual ou semi-automático, segmentos de texto.

As anotações não são inseridas como *elementos* `XML`, mas sim são referências a segmentos de texto. Deste modo é possível ter vários níveis de anotações e múltiplas anotações num mesmo segmento.

Foi seleccionada uma tipologia de termos que é actualmente usada para essas anotações. A classificação actual incluí: *evento*, *toponímia*, *data*, *acontecimento*, *cargo*, *nome próprio* e *nome de organização*. Cada anotação é caracterizada por um destes tipos, um comentário e marcas de início e fim do segmento de texto. O esquema da linguagem `XML` associada é apresentado na Figura 1.

```
Annotations = element Annotations {
   attribute marks { text },
   Annotation+
}
Annotation = element Annotation {
   attribute startchar { text },
   attribute endchar { text },
   attribute type {"evento" | "toponimia" | "data" |
          "acontecimento" | "cargo" |
           token "nome proprio" | token "nome organizacao" },
   attribute normtext { text },
 text}
```

Figura 1. Esquema `Relax NG` para as anotações.

Pretendemos estender o sistema, de modo a permitir anotações associadas a termos de um *thesaurus* ou outros vocabulários controlados. Note-se que este tipo de anotações poderá também ser feita para documentos de áudio ou vídeo.

3.3 Transcrições

Uma da tarefas mais laboriosas no tratamento de entrevistas (ou outros registos de fala) é a transcrição do áudio para texto, e poucas ferramentas existem em *software* livre. No decurso do projecto utilizou-se inicialmente uma ferramenta desenvolvida na Universidade do Minho, `Escriba` e mais recentemente o `transcriber` [14,1], uma aplicação que é usada pela comunidade linguística internacional mas é especialmente vocacionada para transcrições de emissões radiofónicas. Actualmente, está em curso a construção duma aplicação de transcrição, que se adapte melhor aos requisitos deste projecto.

Numa transcrição, será necessário ter informação sobre cada interveniente e a divisão da entrevista em diversas conversações. Na Figura 2 apresentamos o esquema da especificação para transcrições. De notar a ausência de informação sobre a qualidade ou características da gravação áudio, mas tal informação não nos parece essencial para este tipo de transcrição e poder ser difícil de especificar por um transcritor não especialista em áudio.

```
include "annotations.rnc"
include "metadata.rnc"

start = TranScript
TranScript = element Transcript {FileData, SecTion+, Annotations?}
FileData = element Filedata { MetaData, PerSon+ }
PerSon = element Person { attribute id { text },
  attribute name { text },
  attribute email { text }?,
  attribute type {"interviewer" |
                      "interviewee" | "other" }}
SecTion = element Section { attribute title { text },
    attribute desc { text },
    ConverSation+ }
ConverSation = element Conversation {attribute starttime { text },
 attribute endtime { text },
 attribute title { text },
 PerSon,
 text}
```

Figura 2. Esquema `Relax NG` para as transcrições.

Na literatura não é do nosso conhecimento nenhum *standard* para especificações de documentos deste tipo, havendo caracterizações no âmbito do projecto TEI [5] e de alguns projectos de história oral [6].

3.4 Resumos

Os resumos das entrevistas são pequenos textos de acesso livre que sumarizam as narrativas. Presentemente são a maior fonte de informação para quem pretende consultar o arquivo. Os resumos podem ser anotados e estão divididos em secções, que designamos por `histórias`. O esquema apresentado na Figura 3 é baseado no já utilizado pelos resumos que estão disponíveis na *página* `Web` do **CDI**. As principais alterações foram a separação da metainformação e o facto das anotações não corresponderem a elementos `XML` inseridos no texto.

4 Editores

Embora existam editores genéricos para documentos `XML`, estes normalmente obrigam a conhecimentos mínimos da sua sintaxe, assim como do esquema específico associado a cada documento. Ambientes gráficos que escondam pormenores técnicos para o utilizador não especializado facilitam e aceleram a produção e modificação de documentos estruturados. Exemplos de editores de anotações `XML` são o UXO [15] para anotações linguísticas e que permite acesso a base de dados, e o Cadixe [4], desenvolvido no âmbito de um projecto de bioinformática.

```
include "metadata.rnc"
include "annotations.rnc"

start = Resume
Resume = element Resume { MetaData, Stories, Annotations? }
Stories = element Stories {
   attribute title { text },
   attribute desc { text },
   Story+
}
Story = element Story {
   attribute startchar { text },
   attribute endchar { text },
   attribute title { text },
   attribute desc { text },
   text
}
```

Figura 3. Esquema `Relax NG` para os resumos.

Por outro lado, há actualmente a tendência para que os editores sejam integrados em navegadores `Web`. No entanto não optamos por esta alternativa, pois não nos parece que os benefícios desse tipo de ferramentas – não ser necessária instalação e acesso directo a arquivos remotos, por exemplo – compensem as suas desvantagens, nomeadamente de maior custo de desenvolvimento e limitações dado envolverem diferentes tecnologias `Web`, para além da obrigatoriedade do utilizador ter de estar *on-line* para a produção dos documentos.

Assim, optou-se pelo desenvolvimento de aplicações gráficas que contudo devem ser de instalação fácil em qualquer plataforma. O desenvolvimento do sistema informático do **CDI** é essencialmente baseado na linguagem de programação `Python` [22]. Os interfaces gráficos utilizamos a `API` gráfica `wxWidgets` [18], devido à sua portabilidade e vasto `toolkit` disponível. Em particular, a construção dos editores de texto baseia-se na classe `wxStyledTextCrl` que implementa a poderosa componente de edição de texto `Scintilla` [8].

A manipulação de documentos `XML` é feita essencialmente através da `API elementtree` [12], mas usando a implementação `lxml` [13] que é baseada na biblioteca `libxml2` [17] e tem um maior suporte de `XSLT` e `XPath`. As principais vantagens desta `API` são a sua proximidade com a estrutura do `Python`, a sua maior velocidade e pouco gasto de memória: cerca de 15 vezes mais rápida e 5 vezes menos consumidora de memória do que a mais recente biblioteca de `DOM` do `Python`, e cerca de 2 vezes mais rápida que bibliotecas de `SAX`.

4.1 Editor para metainformação

O editor de *metadados* pode ser usado integrado no editor de documentos de texto ou ser usado isoladamente para a catalogação de outros documentos. Em

relação à especificação referida na Secção 3.1, tem algumas limitações, por exemplo apenas permite que os documentos tenham só um título. Utiliza diversos vocabulários controlados. Para além dos habituais *standard*s para idiomas, tipos e formatos de documentos, etc., tem vocabulários específicos, por exemplo para funções dos colaboradores (entrevistadores, transcritores, etc.). De futuro, pretende-se que estes vocabulários possam ser escolhidos/configurados para cada elemento. A possibilidade de inserção de qualquer número de elementos de cada tipo será também revista, mas de modo a que o editor continue simples e de uso intuitivo.

4.2 Editor para a anotação e classificação de documentos

O editor de resumos permite a criação e modificação de documentos que seguem a especificação dada na Secção 3.4. Tem contudo algumas restrições conceptuais. Foi desenhado não como um editor genérico de texto, mas sim como um classificador e anotador de textos. Assim, após a anotação de um segmento de texto não é possível a edição de texto da secção correspondente (neste caso, duma *história*). Deste modo é possível, garantir que quem anota não altera o conteúdo do que está escrito. Poderá contudo, ser desejável um sistema mais flexível mas em que a opção de não edição continue disponível. A inserção de histórias está actualmente a ser generalizada para a classificação de textos segundo uma hierarquia de categorias. Deste modo, pretende-se desenvolver um editor de classificação (e anotação) de documentos mais geral e que será usado para a análise de conteúdo das narrativas. Na Figura 4 apresentamos uma imagem tanto do editor de *metadados* como do de resumos.

O editor de transcrições deve ser integrado numa ferramenta que permita o acesso e manipulação de ficheiros áudio e, embora se possa aproveitar o esquema geral do que foi desenvolvido para os resumos não foi ainda implementado na sua totalidade.

5 Pesquisa nos documentos

Foi implementado um protótipo de uma *página* `Web` para o acesso e pesquisa nos documentos produzidos com as novas ferramentas, em particular, nos resumos de entrevistas. A pesquisa é feita com recurso ao `XPath`. Ainda não está implementado nenhum sistema de indexação que optimize a pesquisa e permita pesquisas e resultados mais avançados como os descritos na Secção 2.

6 Manipulação de um *thesaurus*

Os *thesauri* são hierarquias semânticas de termos, que se situam entre a mera classificação e ontologias mais complexas. Cada termo pode ter associados outros termos por diversas relações, em particular hierárquica e equivalência. Os termos *descritores* são definidores de um conceito e devem ser usados em detrimento de outros que sejam considerados no *thesaurus* como equivalentes.

Figura 4. Editores de resumos e de *metadados*.

Existem disponíveis *thesauri* em diversas áreas temáticas e daí ser importante o acesso à sua manipulação digital. O `Zthes` [23] é um *standard* para a representação, acesso e navegação em *thesauri*. Quando este trabalho começou a ser desenvolvido, no início de 2006, havia apenas disponível um esquema em `DTD` para a versão 0.5 do `Zthes` e que foi adaptado para o esquema em `Relax NG` que se apresenta na Figura 6. Mais recentemente, a *página* `Web Zthes` foi actualizada e disponibiliza diversos esquemas para uma nova versão do `Zthes`, a 1.0.

Tendo a **UPP** acesso a um *thesauri* pluridisciplinar, foi feita a sua conversão para a linguagem `Zthes`. Este *thesaurus* é constituído por cerca de 10000 termos e organizado em 25 áreas temáticas.

A navegação no *thesaurus* pode ser por:

- listagem alfabética de todos os termos
- listagem hierárquica organizada por *microthesauri* dos termos descritores

Para optimizar a listagem hierárquica de termos, foi implementada uma indexação dos termos num dicionário `Python`.

A pesquisa de termos utilizando o `XPath`, pode ser exacta ou aproximada, e permitir múltiplas palavras. O *thesaurus* pode ser consultado através de um interface `Web`, como o exemplificado na Figura 7. Dada a relativa dimensão do *thesaurus* que utilizamos, o desempenho dos procedimentos de pesquisa e navegação, baseados quase exclusivamente em manipulação de `XML`, é bastante satisfatório.

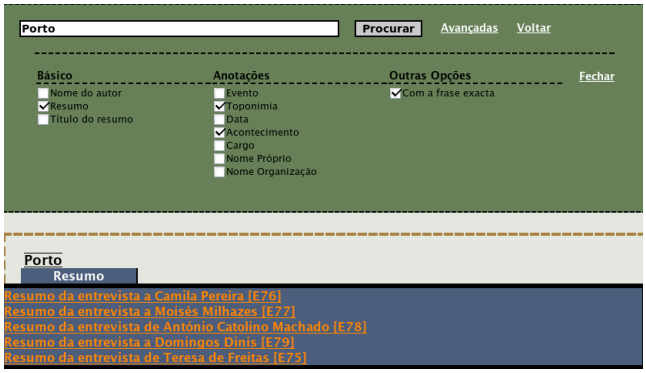

Figura 5. Pesquisa nos resumos.

7 Conclusões e Trabalho futuro

As ferramentas descritas apesar de estarem ainda em fase de desenvolvimento, já podem ser utilizadas. É necessário melhorar os editores de modo a serem mais gerais e mais flexíveis; permitirem a classificação e anotação de textos genéricos, pela definição duma hierárquica de categorias; e integrarem o acesso a *thesauri*. Em especial, pretendemos analisar a possível utilização do modelo de grafos de anotação [2,7] que permite uma maior independência entre o interface com utilizador e os formatos dos documentos. Num grafo de anotação a informação é representada por um conjunto de nós e arcos que podem ser anotados com os mais variados atributos. Note-se que a estrutura usada para implementação de anotações segue já este tipo de formalismo.

Durante 2006 decorreu o projecto *Memórias, Vivências e Identidade dos Trabalhadores Têxteis do Porto - Percursos de Vida e de Trabalho na Indústria Têxtil*, no âmbito do Concurso *Investigação Científica na Pré-graduação*, da Universidade do Porto. Dos materiais produzidos, incluí-se o tratamento de seis entrevistas e algumas das ferramentas informáticas aqui descritas, e que foram já utilizadas no decurso desse projecto.

Os documentos existentes no **CDI** irão ser convertidos para os novos formatos e será desenvolvida uma nova versão da *página* `Web` do **CDI**, segundo os objectivos definidos na Secção 2.

8 Agradecimentos

O trabalho aqui descrito teve contribuições dos vários membros da equipa do projecto *Memórias do Trabalho- Testemunhos do Porto Laboral no Século XX*, nomeadamente Manuel Loff, Teresa Medina, Cristina Nogueira e Eloy Rodrigues. A selecção, recolha e tratamento das narrativas envolveu e envolve um grande número de pessoas em especial alunos de licenciaturas da Universidade do Porto. Entre 2000 e 2004, Luís Pessoa colaborou no desenvolvimento e manutenção da actual *página* `Web` e ferramentas informáticas associadas.

```
start = Zthes
Zthes = element Zthes {Term+}
Term = element term {terment,
   termnote?,
   admin?,
   relation*}
terment  = attribute termId {text},
        element termName {text},
        element termQualifier {text}?,
        attribute termType {text}?,
        attribute termLanguage {text}?
admin = attribute termCreatedDate {text}?,
        attribute termCreatedBy {text}?,
        attribute termModifiedDate {text}?,
        attribute termModifiedBy {text}?
termnote = element termNote {text,
   attribute type { "SN" | "NE"}}
relation = element relation {
   attribute relationType {text},
   element sourceDb {text}?,
   attribute termId {text},
   attribute termName {text}
}
```

Figura 6. Esquema `Relax NG` para *thesauri*.

Referências

1. Claude Barras, Edouard Geoffrois, Zhibiao Wu, and Mark Liberman. Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, 33(1–2):5–2, 2001.
2. Steven Bird and Mark Liberman. A formal framework for linguistic annotation. *Speech Communication*, 33(1–2):23–60, 2001.
3. Steven Bird and Gary Simons. The OLAC metadata set and controlled vocabularies. In *Proceedings of the ACL 2001 Workshop on Human Language Technology and Knowledge Management*, pages 7–18, Morristown, NJ, USA, 2001. Association for Computational Linguistics.
4. Gilles Bisson. *Cadixe user manual: why and what.* CADERIGE Project, 2005.
5. The TEI Consortium. Tei: The Text Encoding Initiative. `http://www.tei-c.org/`, 2006.
6. Janet Crum. Oral history markup language. `http://www.ohsu.edu/library/staff/crumj/oml`, 1999.
7. Edouard Geoffrois, Claude Barras, Steve Bird, and Zhibiao Wu. Transcribing with annotation graphs. In *Second International Conference on Language Resources and Evaluation (LREC)*, pages 1517–1521, 2000.
8. Scintilla Project Group. Scintilla. `http://www.scintilla.org/`, 2006.
9. IMDI Team. IMDI Metadata Elements for Session Descriptions. Technical report, MPI Nijmegen, October 2003.

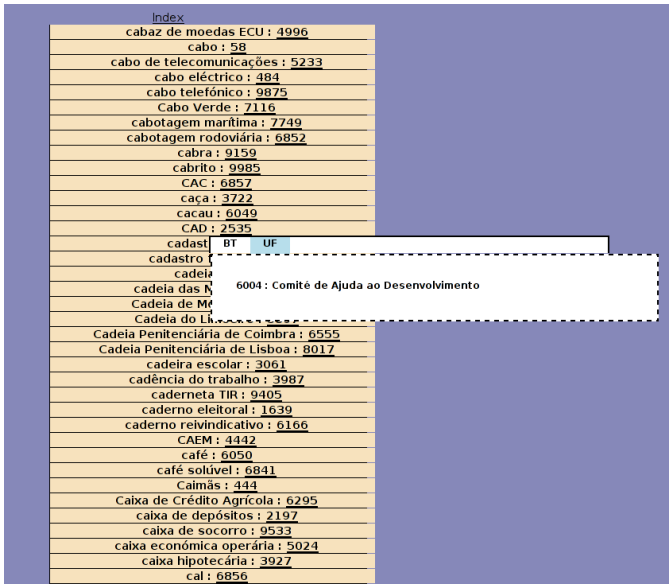

Figura 7. Pesquisa alfabética num *theasaurus*

10. Open Archives Initiative. OAI-PMH Core Resources. `http://www.openarchives.org/`, 2006.
11. The Dublin Core Metadata Initiative. Dublin Core Metadata Terms. `http://dublincore.org/`, 2006.
12. Fredrik Lundh. ElementTree API. `http://effbot.org/zone/element-index.html`, 2006.
13. lxml development team. lxml. `http://codespeak.net/lxml/`, 2006.
14. Sylvain Galliano Mathieu Manta, Fabien Antoine and Claude Barras. Transcriber. `http://trans.sourceforge.net/`, 2006.
15. Jan-Torsten Milde. Uxo: An xml-based extensible annotation editor. In *Proccedings of the GLDV-Spring Meeting*, pages 151–159, Giessen University, 2001.
16. Open Language Archives Community. Olac metadata. `http://www.language-archives.org/OLAC`, 2006.
17. The GNOME Project. The XML library for Gnome. `http://xmlsoft.org/`, 2006.
18. Julian Smart, Robert Roebling, Vadim Zeitlin, and Robin Dunn. *wxWidgets 2.6.3: A portable C++ and Python GUI toolkit*.
19. Universidade Popular do Porto. Centro de Documentação e Informação sobre o Movimento Operário e Popular do Porto. `http://cdi.upp.pt/`.
20. Universidade Popular do Porto. O Centro de Documentação e Informação sobre o Movimento Operário e Popular do Porto da Universidade Popular do Porto. In Manuel Loff e Maria da Conceição Meireles Pereira, editor, *PORTUGAL: 30 ANOS DE DEMOCRACIA (1974-2004)*, pages 287–294. Universidade do Porto, 2006.
21. Eric van der Vlist. *RELAX NG*. O'Reilly, 2003.
22. Guido van Rossum. *Python Library Reference*, 2.4.2 edition, 2005.
23. The Zthes working group. The Zthes specifications for thesaurus representation, access and navigation. `http://zthes.z3950.org/`, 2006.

Automating XML Pipelines Through Rules

Rui Lopes and Luís Carriço

LaSIGE and Department of Informatics
University of Lisbon
{rlopes,lmc}@di.fc.ul.pt

Abstract. This paper presents XPR - XML Pipeline Rules, a novel approach on defining, maintaining, and using XML pipelines for multiple-step document format conversion and automated detection. XPR introduces a template-based approach for reusable blocks of processing operations, as well as an abstraction for detecting document formats based either on XPath or schema validation. With XPR, there is no need to hand code different pipelines for each required document conversion. Therefore, it helps developers on maintenance and extensibility scenarios. A use case is presented, illustrating the drawbacks of current XML pipeline technologies, and how XPR can be used complementary to these by overcoming their limitations.

1 Introduction

XML processing with pipelines is an emerging topic in the XML scene. Pipelines have raised the level of abstraction on defining XML-based applications, discarding the complexity from using traditional API-based XML technologies in programming languages. Thus, it has shifted away from a procedural approach towards a purely declarative one. As such, XML pipelines are defined as a flow of operations to be performed over XML documents (e.g., applying an XSLT stylesheet, validating a document against a schema, etc.), according to some given business logic. Furthermore, pipelines improve on the maintenance and reuse of operations, as users typically tend to develop blackboxed solutions for performing required tasks (e.g., one XSLT stylesheet *vs.* several stylesheets plus something to glue them all).

Albeit being an emerging topic, different XML pipeline technologies have been developed in the last years. Such technologies tried to solve different problems, therefore ended on having different capabilities. Cocoon [1] and XPL [2] were defined with a flavour towards creating full-fledged XML-based web applications; MT pipeline [3] and smallx [4] shared a common goal of providing highly efficient XML processing for large datasets; SXPipe [5], XPDL [6], or even Ant scripts [7] provided simpler (therefore less powerful) ways for specifying XML pipelines. XProc [8] is the W3C's proposal on standardizing these technologies, to cope with their different needs. Despite the differences on purposes and specification languages, all these technologies share a common concept: pipelines are defined explicitly, i.e., a developer specifies the flow of operations to be performed over a given input, delivering an expected output. If the logic behind

this process has to be changed or adapted to different circumstances, the developer has to modify the pipeline accordingly (e.g., adding support for a different XML format at the pipeline's input).

However, the level of abstraction introduced with XML pipeline technologies may be insufficient when scaling or maintaining pipeline-based applications. Imagine the following scenario: a developer has to come up with a solution for transforming different document formats into a normalized publishing format (XSL-FO) to be further processed in consonance to a required business logic. Moreover, this process should be fully automated, to discard the manual selection of appropriate transformations based on document formats. Furthermore, after evaluating the problem, the developer realized that direct transformations to XSL-FO were not available to some document formats (such as one-to-one XSLT stylesheets). The cost of developing them is highly prohibitive, but sequences of off-the-shelf transformations were able to perform those tasks. Based on these requirements and available tecnhologies for document format conversion, the following graph of transformations was identified (Figure 1):

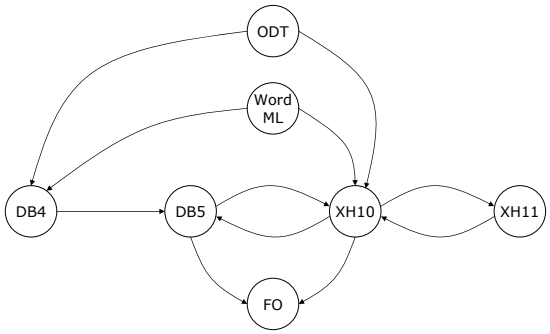

Fig. 1. Graph of transformations

Each node in the graph represents a particular document format that must be supported (Open Document for word processing, *ODT*; Microsoft Word 2003, *WordML*; DocBook v.4, *DB4*; DocBook v.5, *DB5*; XHTML 1.0 Strict, *XH10*; XHTML 1.1, *XH11*; XSL-FO, *FO*), while each arc represents a transformation from its source node to its destination node. In this scenario, any path between a chosen node and the *FO* node is an XML pipeline that can be constructed and used. However, the maintenance cost of creating this type of applications increases as new requirements appear (e.g., supporting a new document format, changing a document transformation operation to a more robust version, support different normalized document formats, etc.), therefore being reflected on pipelines specifications - either by modifying existing ones, fully rewritting them, or even creating new sets of pipelines.

Two concepts can be leveraged from these type of scenarios: identifying the transformations that are available (and how they can be concatenated), and de-

tecting automatically XML document formats. From these high-level concepts, any XML pipeline can be derived according to required transformation semantics. This paper presents *XPR - XML Pipeline Rules*, a high-level language for the specification of these two concepts, named *routing graphs* and *sensors* for document formats, respectively. These rules are defined and transformed into corresponding XML pipelines based on the syntax proposed by the latest XProc Working Draft. Next, the way routing graphs are specified is presented.

2 Defining a routing graph

A routing graph describes what operations have to be used to transform any XML input format into any XML output format. Therefore, each node of a routing graph is, at least, a source or a destination of one processing operation. This means that, for instance, from the scenario described previously, the routing graph is defined by the set of arcs of Figure 1. Mapping such abstractions into a concrete XML syntax is done through *XPR templates rules*, as seen on Figure 2 (a complete solution is presented in Appendix A).

```
<xpr:rules xmlns:xpr="http://www.rlopes.net/xml/xpr"
           xmlns:p="http://www.w3.org/2006/XProc">

  <xpr:template in="odt" out="xh10">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="odt-to-xh10.xsl"/>
    </p:step>
  </xpr:template>

  <xpr:template in="xh10" out="fo">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="xh10-to-fo.xsl"/>
    </p:step>
  </xpr:template>

</xpr:rules>
```

Fig. 2. XPR templates example

This example presents the basic language constructs usage for defining a routing graph, through the specification of a template that converts *ODT* into *XHTML 1.0*, and another one from *XHTML 1.0* to *XSL-FO*. At the top level, the `xpr:rules` element agreggates templates defined with `xpr:template` elements. Each template defines its input and output identifiers (e.g., document formats) with the `in` and `out` attributes, respectively.

Inside each template, an XProc-based syntax of a flow of operations has to be defined. Each template may be a lot more sofisticated as compared to the example, such as combining several XProc steps and constructs, calling other pipelines, etc. This is typically done when the intermediate results from the operations are not intended to be exposed on the routing graph. In order to link each template when computing XProc-based pipelines, three bindings have to be used inside the templates. These bindings are `$name` (the name of the flow of operations to be referenced inside pipelines), `$from` (the operation upon which the template is dependent from), and `$source` (the specific source to be grabbed from the operation). The presented templates will be used as operations on other pipelines that will be also generated: converting *WordML* and *XHTML 1.1* to *XSL-FO*, both will use the conversion between *XHTML 1.0* and *XSL-FO* defined in the second template. Even further, if a different conversion target is chosen for the same routing graph (i.e., replacing XSL-FO), both templates can be used seamlessly, therefore leveraging extensibility and maintenance scenarios.

After the establishment of the desired routing graph, the abstraction layer for the automatic detection of document formats is defined through a set of *sensors*, as presented in the next section.

3 Attaching sensors

As explained earlier, an XML pipeline is typically defined to support a specific document format as its input. Adding support for other document formats may be implemented through XPath-based testing with `choose/when/otherwise` constructs (similarly to XSLT's). More advanced scenarios may opt for schema validation as the way to detect document formats and trigger appropriate pipelines to perform desired operations, through `try/catch` mechanisms. Both approaches are similar in their semantics, as they perform some type of detection of document formats. However, as they are reflected differently on XML pipeline syntax, it poses increasing difficulties on extensibility and maintenance scenarios (i.e., typically results on deep trees of nested `choose` and `try` blocks). Moreover, when a different conversion target is required to be supported, these mechanisms have to be adapted accordingly in a manual fashion.

Consequently, XPR introduces the notion of *XPR sensor rules* for document formats to allow the automation of selecting an approriate XML conversion pipeline. Each sensor describes its semantics either through an XPath expression or a validation schema, and links itself to a specific document format. Figure 3 presents an example for the scenario described earlier (a complete solution is presented in Appendix A).

This example shows the syntax for describing sensors, both XPath-based and schema-based. The XPath expression tests loosely for an *XHTML 1.0* document, while a RELAX NG schema identifies an *ODT* document. In addition to `xpr:template` elements, the `xpr:rules` element also wraps `xpr:sensor` elements, the way sensors are described. Each sensor defines its associated document format through the `format` attribute. In the case of XPath-based sensors,

```
<xpr:rules xmlns:xpr="http://www.rlopes.net/xml/xpr"
           xmlns:html="http://www.w3.org/1999/xhtml">

  <xpr:sensor format="xh10" test="/html:html and not(//html:ruby)"/>

  <xpr:sensor format="odt" schema-href="odf.rng" schema-type="relaxng"/>

</xpr:rules>
```

Fig. 3. XPR sensors example

the `test` attribute must be used for defining the document format identification expression. On the other hand, when creating a schema-based sensor, the `schema-href` and `schema-type` have to be used to identify the schema location and its type (e.g., `xmlschema`, `relaxng`, etc.), respectively. In the case of having more than one sensor for the same document format, sensor ordering inside the XPR document decides which one to be applied.

After the rules for the routing graph and sensors have been described, they must be applied in the appropriate way, as explained in the next section.

4 Applying the rules

With the routing graph described, and the sensors defined accordingly, these rules are applied in the following way:

- A node in the routing graph is selected as the target document format;
- Each route between every node and this node is traced;
- For each route, a corresponding pipeline is created;
- The appropriate sensor is attached to each pipeline.

However, this algorithm may yeld more than one valid route between starting and ending nodes. In such cases, the shortest path method must be applied accordingly, as less transformations will be used (typically resulting on a better performance). If, after this, more than one route can still be selected, template ordering inside an XPR document dictates the decision (earlier templates will be chosen over latter ones).

Another feature on applying XPR rules bases itself on the lifespan of sensors. Their main goal is to automatically detect a document format and select an appropriate pipeline. However, they may be used also as validation mechanisms between the operations defined by templates, therefore enforcing that valid documents are fed to and produced by each processing operation.

The concrete application of XPR rules on existing or newly-created XProc-based pipelines can be done in different scenarios, depending on its goals and execution environment, as follows:

1. *Offline rules unwinding:* this type of XPR rules application is based on transforming a set of rules into a concrete XProc library consisting on the set of pipelines for each document format to be supported, and a pipeline that combines the specified sensors with their corresponding pipelines. This type of application is particularly suitable for command line or script-based generation of pipelines;
2. *Double pass pipeline execution:* in this case, an XProc step for triggering XPR is used inside XProc pipelines (like any other XProc step). However, the execution environment preprocesses the XPR step by unwinding it into concrete pipeline libraries (like in the previous case), and replacing the XPR step with a call to the sensors pipeline. This case enables the direct usage of XPR inside XProc pipelines without having to implement and/or adapt an XPR engine to an XProc processor;
3. *XProc component:* syntactically, this case is used exactly in the same way as the double pass pipeline execution case. However, by implementing a native component for XProc, there is no need for pre-processing a pipeline document. Therefore, pipelines can be executed directly. Another benefit from this approach comes from performance optimization issues, as the implementation may pre-compile and cache the generated pipelines.

As said, for the second and third cases, XPR is used directly inside XProc pipeline documents. This is done seamlessly, through the definition of the a special purpose processing component, according to the XProc step signature presented in Figure 4 (namespaces omitted for simplification):

```
<p:declare-step-type type="xpr:rules">
  <p:input port="document"/>
  <p:input port="rules"/>
  <p:output port="result"/>
  <p:parameter name="format" required="yes" />
  <p:parameter name="sensors-everywhere" />
</p:declare-step-type>
```

Fig. 4. XProc step signature for XPR

This step declaration for XPR requires two inputs (the `document` to be processed, and the `rules` document), produces one `result` document, and requires two parameters, the document `format` that will be chosen to be delivered by generated pipelines, and an optional `sensors-everywhere` flag to signal the insertion of sensors after every routing graph nodes (increasing their lifespan during pipeline execution).

From the scenario described earlier in this paper, a real XProc pipeline that uses XPR according to this step signature is presented in Figure 5.

Next section discusses briefly the way an XPR processor prototype was implemented.

```
<p:pipeline xmlns:p="http://www.w3.org/2006/XProc"
            xmlns:xpr="http://www.rlopes.net/xml/xpr"
            name="publisher">

  <p:input port="document"/>
  <p:output port="result" step="branding" source="result"/>

  <p:step name="normalize" type="xpr:rules">
    <p:input name="document" step="publisher" source="document"/>
    <p:input name="rules" href="rules.xml"/>
    <p:parameter name="format" value="fo"/>
  </p:step>

  <p:step name="branding" type="p:xslt">
    <p:input name="document" step="normalize" source="result"/>
    <p:input name="stylesheet" href="branding.xsl"/>
  </p:step>

</p:pipeline>
```

Fig. 5. XProc pipeline example using an XPR-based step

5 Implementation details

As a proof-of-concept, an XPR processor was implemented in XSLT 2.0 in consonance with the first and second scenarios described on the previous section. This implementation transformed both sensor and routing rules into concrete XProc pipelines, according to XPR semantics described earlier. Three main aspects can be distinguished on implementing XPR: detecting which routes are valid according to the selected target document format; generate the sensor pipeline; and generate each concrete route's pipeline.

The first step on implementing XPR in XSLT is to detect which routes are valid, i.e., which input document formats have a route to the target document format. Due to XSLT's functional nature, the prototype XPR processor implemented route detection through a naïve recursive breadth-first algorithm. First, an initial route set is chosen based on all input document formats (i.e., available templates's `in` and `out` attributes). Afterwards, each route's ending node is expanded to all available subsequent nodes in the routing graph. This operation is repeated for each route, until one of two conditions is verified: a loop has been detected, or no more expansion can be performed. After all valid routes have been found, the shortest ones are filtered for each available input document format, therefore forming the routing set that will be used subsequently.

Based on the routing set calculated previously, the sensor pipeline can now be created. Each sensor is transformed into specific XProc constructs, depending on its type (XPath or schema-based). For succeding sensor tests, a corresponding

transformation pipeline is called. Each sensor failure is recusively transformed into the next route's sensor detection.

More concretely, XPR sensor based on XPath expressions are transformed into `choose/when/otherwise` XProc constructs (see Figure 6 for an example): `when` the testing for the sensor's XPath expression succeeds, it means that a document has been fed that matches the rules specified by the sensor, therefore the appropriate pipeline is called; `otherwise`, iterate to the next sensor in the list.

```
<p:choose name="generated-sensor-name">

  <p:when test="sensor's XPath expression"/>
    <!-- call the corresponding pipeline -->
  </p:when>

  <p:otherwise>
    <!-- iterate to the next sensor -->
  </p:otherwise>

</p:choose>
```

Fig. 6. XProc skeleton for an XPR XPath-based sensor

In the case of schema-based sensors, their semantics are transformed into `try/catch` XProc constructs (see Figure 7 for an example): first, `try` to validate the input document according to the specified schema and execute the corresponding pipeline; if something goes wrong, `catch` the validation error and iterate to the next sensor in the list.

These XProc skeletons for XPR sensors further emphasize the (probable) inherent complexity of specifying directly a sensor pipeline in XProc, as the pipeline's XML tree deepness grows proportionally to the number of sensors that must be used. Once again, this issue may pose severe problems on scaling and maintaining this type of pipelines in a manual fashion, therefore enforcing the need for the abstraction layer provided by XPR.

The last step on transforming XPR into concrete XProc pipelines relates to the creation of the transformation pipelines themselves. As in the previous case, the routing set calculated initially serves as the ground basis for pipeline construction. Therefore, each route arc is transformed into the corresponding step defined inside XPR template rules, and all bindings (`$name`, `$from`, and `$source`) are adapted accordingly.

The result from all these transformations is an XProc library consisting of a sensor pipeline, and a set of pipelines based on the calculated routes. From the scenarios presented in the previous section, this prototype can be used standalone (unwinding mode) or used in conjunction with a translator of XProc-based XPR

```
<p:try name="generated-sensor-name">

  <p:group name="generated-group-name">
    <p:step name="generated-step-name" type="p:schema-type">
      <p:input port="document" ... />
      <p:input port="schema" href="schema-location" />
    </p:step>
    <!-- call the corresponding pipeline -->
  </p:group>

  <p:catch>
    <!-- iterate to the next sensor -->
  </p:catch>

</p:try>
```

Fig. 7. XProc skeleton for an XPR schema-based sensor

steps into concrete pipeline calls. Both cases are supported in the prototype through simple Ant scripts, thus out-of-the-box functionality is available.

6 Concluding remarks

This paper presented XPR - XML Pipeline Rules, a solution for simplifying the creation of XML pipelines for document format conversion based on partitioned flows of operations, and a way to ease the automation on document format detection based either on XPath expressions or full schema validation. By defining a set of template rules and a target document format, no pipelines have to be specified by hand, reducing human errors and maintenance costs on improving, modifying, and extending pipeline-based document transformations. Moreover, focusing solely on specifying rules for document format detection, developers do not have to tinker complex document detection blocks to be able to apply the same pipeline-based business logic to different document formats automatically.

As the XML pipeline syntax followed by XPR is XProc, some details on XPR's syntax and implementation may be modified in the future, side-by-side with the evolution of XProc until reaching *Technical Recommendation* status. Moreover, XPR will evolve also as a native XProc processing component to be used out-of-the-box in XProc implementations (according to the third scenario described on the rules application section).

Lastly, the current way XPR template rules are specified and used is based solely on matching in and out document formats. This idea can be further extended to more complex annotations (e.g., semantic-based) upon which sofisticated inference engines can select more appropriate routes and generate corresponding pipelines. Such annotations can be used to, for instance, identify priorities on choosing which route is more appropriate for transforming a given

document format to the desired target document format (e.g., based on the sofistication/accuracy of the transformations specified inside each XPR template), or even composing several annotations as a way to describe each template's acceptable XML data (i.e, using logic-based operators).

References

1. Apache Foundation: Apache Cocoon (1999-2006) http://cocoon.apache.org.
2. Bruchez, E., Vernet, A.: XML Pipeline Language (XPL) Version 1.0 (Draft). W3C Member Submission (2005) http://www.w3.org/Submission/xpl.
3. Markup Technologies: Mt pipeline (2006) http://www.markup.co.uk.
4. Milowski, A.: smallx (2006) https://smallx.dev.java.net.
5. Walsh, N.: SXPipe - Simple XML Pipelines (2004) https://sxpipe.dev.java.net/nonav/specs/sxpipe.html.
6. Walsh, N., Maler, E.: XML Pipeline Definition Language Version 1.0. W3C Note (2002) http://www.w3.org/TR/2002/NOTE-xml-pipeline-20020228.
7. Apache Foundation: Apache Ant (2000-2006) http://ant.apache.org.
8. Walsh, N., Milowski, A.: XProc: An XML Pipeline Language. W3C Working Draft (2006) http://www.w3.org/TR/xproc.

A Scenario solution

For purely informative purposes, the scenario described in the paper's introductory section is presented in its full form, as follows:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xpr:rules xmlns:xpr="http://www.rlopes.net/xml/xpr"
           xmlns:p="http://www.w3.org/2006/XProc"
           xmlns:html="http://www.w3.org/1999/xhtml"
           xmlns:w="http://schemas.microsoft.com/office/word/2003/2/wordml">


  <!-- sensor rules -->
  <xpr:sensor format="xh10"   test="/html:html and not(//html:ruby)"/>
  <xpr:sensor format="xh11"   test="/html:html and //html:ruby"/>
  <xpr:sensor format="wordml" test="/w:wordDocument" />

  <xpr:sensor format="odt"    schema-href="odf.rng"      schema-type="relaxng"/>
  <xpr:sensor format="fo"     schema-href="fo.rng"       schema-type="relaxng"/>
  <xpr:sensor format="db4"    schema-href="docbook4.rng" schema-type="relaxng"/>
  <xpr:sensor format="db5"    schema-href="docbook5.rng" schema-type="relaxng"/>

  <!-- template rules -->
  <xpr:template in="odt" out="db4">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="odt-to-db4.xsl"/>
```

```
  </p:step>
</xpr:template>

<xpr:template in="odt" out="xh10">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="odt-to-xh10.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="wordml" out="db4">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="wordml-to-db4.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="wordml" out="xh10">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="wordml-to-xh10.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="db4" out="db5">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="db4-to-db5.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="db5" out="fo">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="db5-to-fo.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="db5" out="xh10">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="db5-to-xh10.xsl"/>
  </p:step>
</xpr:template>

<xpr:template in="xh10" out="db5">
  <p:step name="{$name}" type="p:xslt">
    <p:input port="document" step="{$from}" source="{$source}"/>
    <p:input port="stylesheet" href="xh10-to-db5.xsl"/>
  </p:step>
```

```
  </xpr:template>

  <xpr:template in="xh10" out="fo">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="xh10-to-fo.xsl"/>
    </p:step>
  </xpr:template>

  <xpr:template in="xh10" out="xh11">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="xh10-to-xh11.xsl"/>
    </p:step>
  </xpr:template>

  <xpr:template in="xh11" out="xh10">
    <p:step name="{$name}" type="p:xslt">
      <p:input port="document" step="{$from}" source="{$source}"/>
      <p:input port="stylesheet" href="xh11-to-xh10.xsl"/>
    </p:step>
  </xpr:template>

</xpr:rules>
```

eXPLoit
Um Sistema de Processamento em Pipeline de XML

Nuno Teixeira[1] e José Paulo Leal[2]

[1] FCUP, Universidade do Porto
[2] DCC-FC & LIACC, Universidade do Porto

Resumo Um *pipeline* de processamento *XML* é uma sequência de transformações de documentos em que a saída de uma transformação alimenta a entrada da transformação seguinte. A vantagem da utilização de *pipelines* é a flexibilização do uso das transformações de *XML*, permitindo a sua reutilização em diferentes *pipelines*.
As linguagens de *pipeline* de processamento *XML* permitem definir e controlar o encadeamento das transformações. Têm sido propostas várias linguagens deste tipo, entre as quais o *XPL - XML Pipeline Language* - submetido ao W3C para discussão pela *Orbeon Inc.*.
Este artigo descreve o *eXPLoit*, um sistema de processamento em *pipeline* de dados *XML*, que inclui um interpretador da linguagem *XPL* e um conjunto de adaptadores que facilitam a sua ligação a diferentes contextos, nomeadamente: a linha de comandos, aplicações *web* e *webservices*. Os adaptadores existentes no *eXPLoit* convertem os dados relativos a cada ambiente em fluxos de *XML* que são processados pelo interpretador. Exemplos desses dados são a informação contida num pedido *HTTP* para uma aplicação *web* ou a mensagem *SOAP* enviada para um *Web Service*. O interpretador do *eXPLoit* avalia um documento *XPL* que controla a invocação de um conjunto de processadores *XML*. O *eXPLoit* inclui um biblioteca básica de processadores, tais como transformações *XSL*, conversões de *CSV* em *XML*, consultas a *webservices*, entre outros. É também possível a adição de processadores ao *eXPLoit*. O interpretador possibilita o uso das estratégias de avaliação *lazy* e *greedy*.

1 Introdução

Com a crescente preponderância do *XML* na transferência de dados entre sistemas heterogéneos, são cada vez mais frequentes os componentes que usam este formalismo. De facto, o *XML* já extravasou há muito a formatação de páginas *web* para o qual foi desenhado. É a peça fundamental na implementação de *web services*, quer na formatação das mensagens, quer no protocolo que as envolve, quer na especificação dos próprios serviços. O *XML* é também cada vez mais usado na formatação de documentos de aplicações, garantindo um formato aberto, independente de plataformas e dos sistemas que os produziram.

O conceito de *pipeline XML* foi proposto para gerir o crescente número de componentes desenvolvidos para processar dados formatados em *XML*. Um *pipeline XML* liga um conjunto de componentes que processam *XML*, conectando

os fluxos de saída dum processador aos fluxos de entrada do seguinte. Esta abordagem permite a reutilização de processadores de *XML* pela sua participação em diferentes *pipelines*. Permite também uma nova forma de programação, baseada na articulação de um conjunto de componentes genéricos que processam *XML*.

A gestão de *pipelines XML* é feita por motores de *pipeline*, a partir de definições em linguagens *XML* especializadas. A generalidade dos motores de *pipelines* actualmente existentes estão associados a sistemas específicos, dos quais se destacam os gestores de conteúdos, como o *Cocoon*[1] ou o *Orbeon Forms* (anteriormente chamado *Orbeon Presentation Server*)[2]. No entanto, um motor de *pipeline* genérico, utilizável em múltiplos contextos, poderá aumentar a capacidade de reutilização dos processadores de *XML*.

Um motor de pipeline que possa ser usado indistintamente a partir de um contentor de *servlets*, de um motor *SOAP*, ou mesmo da linha de comandos permitirá a reutilzação dos mesmos *pipelines* como parte duma aplicação *web*, como operações de um *web service*, ou como comandos *shell*. O objectivo do trabalho apresentado neste artigo é, portanto, desenvolver um motor de *pipeline XML* com possibilidade de utilização em múltiplos contextos, e explorar as possibilidades deste tipo de abordagem.

O motor de *pipeline* desenvolvido -o *eXPLoit* - deve o seu nome à linguagem de *pipeline* em que se baseia, o *XPL*. O *eXPLoit* é constituído por um interpretador desta linguagem, com uma *API* neutral que permite a sua utilização em múltiplos contextos. O *eXPLoit* contém ainda um conjunto de adaptadores que facilitam a utilização do interpretador em alguns contextos, como sejam os pedidos e resposta *HTTP*, as mensagens *SOAP*, o sistema de ficheiros e o ambiente de execução.

Neste artigo começaremos por apresentar as linguagens de *pipeline* em geral e o *XPL* em particular, bem como alguns dos sistemas que integram motores de *pipeline*. Seguidamente descreveremos o sistema *eXPLoit* e terminaremos com um exemplo da sua utilização em diferentes contextos.

2 Estado de Arte

Nesta secção são apresentados alguns conceitos sobre o processamento *pipeline*, começando por definir o próprio conceito. São também apresentadas as principais linguagens de definição de *pipeline*. Dos vários motores de *pipeline* presentemente em utilização, como a plataforma *Orbeon Forms*, destacamos o gestor de conteúdos *Cocoon*, cuja linguagem de definição de *pipeline* tem um papel central na configuração da *framework*. Nesta secção é também dada uma ideia geral da linguagem XProc.

2.1 Definição de *XML Pipeline*

Um pipeline de processamento *XML* corresponde à interligação de um conjunto de processamentos *XML*, normalmente chamadas transformações *XML*.[3]

Dadas duas transformações T_1 e T_2, estas podem ser ligadas de forma a que um documento *XML* transformado por T_1 seja posteriormente usado por T_2. Estes *pipelines* são chamados lineares, pois um documento dado como *input* percorre uma série de transformações pré-fixadas para produzir um documento como *output*. *Pipelines* não lineares podem incluir:

- Múltiplos *inputs* e *outputs* – possibilidade de receber *input* de várias fontes e a sua aplicação resultar em diversos *output*s;
- Testes – onde uma *pipeline* é executada se se verificar uma dada condição;
- Iteração – a transformação é aplicada a um conjunto de fragmentos *XML* seleccionados de um documento;
- Agregações – múltiplos documentos (ou fragmentos *XML*) são agregados num único.

2.2 Linguagens *XML Pipeline*

As linguagens *XML Pipeline* são usadas para definir *pipelines*. Um programa escrito numa linguagem deste tipo é interpretado por um componente denominado *motor de XML pipeline*. Este é responsável por criar processamentos, ligá-los entre si e, por fim, processá-los. Existem várias linguagens deste tipo, tais como:

- *W3C XML Pipeline Language (XPL) Version 1.0 (Draft)*[4] especificada numa *W3C Submission*. A *Orbeon Inc.* efectuou uma implementação de uma versão anterior a esta linguagem. Esta é a linguagem que é interpretada pelo sistema que este artigo pretende ilustrar;
- *smallx XML Pipelines*[5] é usada pelo projecto *smallx*;
- *ServingXML*[6] define um vocabulário para exprimir transformações $flat \rightarrow XML$, $XML \rightarrow flat$, $flat \rightarrow flat$, e $XML \rightarrow XML$ em *pipelines*;
- *XProc: An XML Pipeline Language*[7] é a linguagem mais recente cuja especificação começou a ser formulada em finais de Dezembro 2005 e continua muito activa. Esta linguagem surge como uma tentativa de se criar uma linguagem *standard* para processamento *XML*.

XProc

Esta linguagem aborda um *pipeline* como sendo um conjunto de componentes interligados. Quer os componentes quer o *pipeline* aceitam como *input* um conjunto (vazio ou não) de documentos *XML* e produzem, ou não, um conjunto de documentos *XML* como *output*. O *input* de um componente pode ser um documento *web*, ser o *input* do *pipeline* ou o *output* de outro componente. A *XProc* classifica os componentes em dois tipos: *steps* e *constructs*. Um *step* consiste num conjunto de operações, nomeadamente processamentos *XML*. Por seu lado, um *construct* engloba um conjunto de componentes. Um exemplo de um *construct* é o próprio pipeline. A linguagem *XProc* encaixa-se no conjunto de linguagens de *pipelines* não lineares, pois suporta múltiplos inputs e outputs, testes, iterações.

Além disso, inclui um conjunto de funcionalidades tais como: execução de transformações usando o sistema *try/catch*, cópias de documentos *XML* em que determinadas áreas desse documento são transformadas, importação e execução de *sub-pipelines*.

2.3 Motores de linguagens *XML Pipeline*

Neste secção são enunciados alguns dos motores de linguagens *XML* Pipeline existentes no momento. É indicada a forma como estes motores interagem com os programas que estão escritos na linguagem *XML* que é usada na sua especificação.

Orbeon Forms
A *Orbeon* desenvolveu uma plataforma *open source* que permite criar aplicações *web* tendo como base o uso de *XForms* e documentos *XML*. Esta plataforma é composta pelos seguintes componentes: *XForms* para a criação de formulários *web* complexos; *PFC (Page Flow Controller)* é o centro de qualquer aplicação nesta plataforma, pois define as páginas da aplicação e como é feita a navegação entre elas; *XPL* é a linguagem usada para definir os *pipelines* de processamento *XML* que podem ser usados na aplicação.

O motor que manipula os formulários é baseado em *Ajax*, usando a biblioteca fornecida pela *Yahoo!*: *Yahoo! User Interface library*. O *PFC* é o componente responsável para controlar o fluxo de execução de uma aplicação implementada na plataforma. Este componente direcciona os pedidos que vêm por parte do cliente para as páginas que fazem parte da aplicação. Estas páginas devem obedecer a uma arquitectura *MVC* (*Model-view-controller*). A analogia é a seguinte:

- *Model* - a sua função é obter informação para ser mostrada pelo *View*. Normalmente o *Model* está associado ao um pipeline *XPL*;
- *View* - produz documentos *XHTML* e *XForms*, mas pode produzir outros formatos tais como: *XSL-FO* ou *RSS*. Um exemplo de um *View* é documento no formato *XSL* que efectua *XSLT* sobre o resultado fornecido pelo pipeline;
- *Controller* - é precisamente esta a função ligada ao *PFC*. Envia o pedido que chega de um cliente, como um *browser*, aos respectivos *Model* e *View*. Por isso indica a ligação entre o *Model* e o *View*.

Esta plataforma possui um conjunto de processadores, entre os quais se destacam: processadores que efectuam transformações *XSL*, acesso a base de dados *SQL*, serialização de documentos *XML* para o disco, comunicação com *webservices*. É possível uma integração com a linha de comandos e com webservices.

Cocoon
O *Cocoon* é uma plataforma de publicação *XML* na web. Permite definir documentos *XML* e aplicar transformações para outros formatos, como o *HTML*, *PDF*, ou texto. O *Cocoon* permite controlo das transformações, possibilitando, desta forma, *pipelines* não lineares.

Os processadores de *XML* do *Cocoon*, controlados pela sua linguagem de definição de *pipelines* são os seguintes: *Matchers*, *Generators*, *Transformers*, *Serializer*, *Selectors*, *Views*, *Readers* e *Actions*.

Uma *pipeline* no *Cocoon* deve ser formada por uma sequência dos seguintes componentes: um *generator* que fornece os dados a serem trabalhados (como por exemplo, o *FileGenerator* que lê um ficheiro e o converte em eventos *SAX*); um *transformer* que recebe eventos *SAX* e gera outro *XML* como eventos *SAX* (como por exemplo o *XalanTransformer* que aplica *XSL*); e finalmente um *serializer* que recebe eventos *SAX* e converte-os para outro formato (como o *XMLSerializer* - transforma eventos *SAX* num ficheiro *XML*).

3 XML Pipeline Language (XPL)

O *XPL* é linguagem de definição de *pipelines* proposta à W3C pela *Orbeon Inc.*. Esta linguagem é utilizada no *Orbeon Forms*, uma plataforma desenvolvida pela *Orbeon*, e é também implementada pela *Oracle* no *Oracle XML Developer's Kit*[8]. Esta linguagem de *pipeline* foi a escolhida para ser incluída porque quando se iniciou o estudo para a implementação do *eXPLoit* era a que possuia uma definição mais estável. Chegou-se a colocar a linguagem *XProc* como hipótese mas a sua definição encontrava-se numa fase embrionária. Contudo, é encarado como uma tarefa a ser executada a inclusão de um interpretador de *XProc* no *eXPLoit*.

3.1 Definição do XPL

Um ***Infoset*** é um fragmento de *XML*. Cada um desses fragmentos tem uma identificação a qual permite que seja possível o acesso ao seu valor. Exemplos de *Infoset* são os parâmetros de *input* e *output*.

Um **programa *XPL*** é um documento *XML* que indica uma sequência de operações efectuadas sobre um conjunto de dados: *Infosets*. Cada operação é definida como um comando da linguagem. As operações sobre um *Infoset* incluem a produção, o consumo ou a transformação de *Infosets*. Um exemplo de transformações é o *XSL* [9]. O *XPL* inclui ainda comandos de controlo de fluxo de execução, como testes e iterações e, controlo de I/O, à semelhança das linguagens de programação. Os programas *XPL* são validados pelo interpretador antes de serem processados. O interpretador assegura a validade sintáctica dos programas consultando um ficheiro *XSD* (*XML Schema Definition*), incluído no *eXPLoit*.

A cada **comando** está associado um elemento específico no espaço de nomes *http://www.orbeon.com/oxf/xpl*. A especificação do XPL indica a criação de um prefixo *p* associado a este espaço de nomes. Um programa *XPL* suporta os seguintes elementos:

- `p:input, p:output`
  identifica os parâmetros de *input* e *output*, respectivamente. Estes elementos poderão existir dentro dos elementos seguintes, dependendo do seu tipo;

- `p:pipeline`
  engloba todo o programa *XPL*. Este comando tem como argumentos *p:input* e *p:output* que indicam os parâmetros de entrada e saída do programa. Estes argumentos têm uma identificação associada para permitir o seu uso dentro do programa;
- `p:choose`
  teste de condições para controle de fluxos de execução. Este comando contém um conjunto de ramos, cada um deles com um teste. É a verificação destes testes que indica se o ramo deverá ser processado pelo interpretador;
- `p:for-each`
  itera sobre um *Infoset*, executando os comandos embebidos no elemento, por cada iteração. O conjunto de fragmentos *XML* a iterar é dado por uma expressão na linguagem *XPath*[10] aplicado a um *Infoset*. Existe uma referência que é associada a cada fragmento que é iterado. Esta referência permite aceder ao valor do fragmento quando os comandos embebidos no elemento são executados;
- `p:procesor`
  este elemento define um processamento sobre fragmentos *XML*. Os processadores são o ponto de extensão do *motor pipeline*, podendo ser adicionados processamentos para novas funcionalidades. O funcionamento do comando p:processor é análogo a uma chamada a uma função numa linguagem declarativa, onde são passados os parâmetros de *input* e têm como retorno determinados *output*s.

O acesso a um *Infoset* faz-se recorrendo a **referências**. As referências são indicadas em atributos de um comando. Estas podem:

- ser referências a documentos externos: ficheiros alojados no sistema ou páginas *web*;
- identificar dados existentes na tabela de variáveis no decorrer da execução do interpretador. Como por exemplo, *#data*, que corresponde à variável denominada *data*;
- agregar documentos usando a função *aggregate()*;
- seleccionar parte de fragmentos *XML*, usando expressões *XPointer*[11];
- corresponder a um fragmento *XML* que é iterado num *p:for-each*.

3.2 Hello World

O seguinte exemplo de um programa em *XPL* ilustra a estrutura base da linguagem assim como o uso do processador *xpl:hello*, um dos processadores da biblioteca de processadores do *eXPLoit*:

```
<p:pipeline version="1.0"
          xmlns:p="http://www.orbeon.com/oxf/xpl"
          xmlns:xpl="http://www.orbeon.com/oxf/xpl/standard">
    <p:input name="data" infoset="http://www.helloworld.com" />
    <p:output name="data" infosetref="#data" />
```

```
    <p:processor name="xpl:hello" />
</p:pipeline>
```

Este programa vai receber a página alojada em *http://www.helloworld.com* e essa página será o seu *output*. O *p:input* e o *p:output* estão interligados, pois usam o mesmo identificador (*data* e *#data*) como consta nos atributos destas *tags*. O processador *xpl:hello* escreve no *stdout* do sistema o texto "Hello world"quando é executado.

3.3 Processadores da biblioteca básica de processadores

A especificação do *XPL* apresenta um conjunto de processadores que considera serem *standard*. Seguem-se esses processadores e uma breve descrição da funcionalidade de cada um:

- **identity** Usando este processador torna-se possível atribuir a um dado identificador um determinado *Infoset*. Este processador é muito útil para definir *Infoset* associado a um fragmento *XML* no contentor do processador ou até num documento externo. Esta utilização do *identity* pode ser equiparada às variáveis de uma linguagem de programação.
- **pipeline** Com este processador é possível executar sub-programas em *XPL*. Como argumentos, são indicados um conjunto de *input*s e *output*s. Como *input* indicam-se novos *Infoset*s ou *Infoset*s que existam no ambiente de variáveis do programa que usa este processador. Estes *Infoset*s serão guardados no ambiente de variáveis do sub-programa. Como um sub-programa pode gerar vários *output*s, deve-se indicar nos argumentos p:output quais os *output*s em que se está interessado. Esses são passados para o ambiente do programa principal. Este processador permite modular a programação por permitir que se possa chamar o mesmo sub-programa em programas distintos.

4 eXPLoit

O *eXPLoit* é composto por vários componentes conforme esquematizado na figura 1. O interpretador é responsável pela interpretação dos programas *XPL* e os restantes acedem a este interpretador e denominam-se adaptadores. Os adaptadores comunicam com o interpretador usando a *API* que este disponibiliza, e estão associados a diferentes ambientes onde a execução de *XPL* ocorre: numa *shell*, na *web* ou num *webservice*.

4.1 Estrutura base

Cada adaptador é responsável por fornecer ao interpretador um *Infoset* que possua dados relacionados com o ambiente que o adaptador suporta. O *Infoset* tem o identificador *#request*. Segue uma breve descrição dos adaptadores que são incluídos no *eXPLoit* e qual o valor de *#request*:

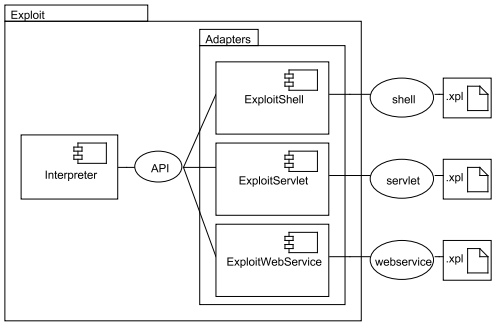

Figura 1. Estrutura base do eXPLoit

- *ExploitWebService* - implementa um *webservice* que comunica com o interpretador. É responsável por enviar os dados que servirão de *input* ao programa *XPL* recebidos no pedido *SOAP* e produz uma mensagem de resposta *SOAP* com o *output* do mesmo. Este módulo usa o *AXIS* para implementar o *webservice*. O *#request* é a mensagem *SOAP* que é enviada para o *webservice* convertida para um fragmento *XML*;
- *ExploitServlet* - uma classe que faz a ligação entre um pedido *HTTP* (*POST* ou *GET*) e a interpretação de um programa *XPL*. Esta classe poderá ser usada com qualquer contentor de *Servlets*, como o *Tomcat*, que recorra a *Servlets* para disponibilizar serviços. O fragmento XML que é identificado por #request contém informações presentes no *HTTP Request*. São indicados valores que existem no *header* como: o *browser* usado, o tipo de conteúdo e o *charset encoding* usados pelo *browser*, entre outros. São também indicados dados referentes ao pedido *HTTP* efectuado, ou seja, o nome das variáveis e respectivos valores;
- *ExploitShell* - é o que indica ao interpretador quais os argumentos colocados na linha de comandos quando a sua execução é feita a partir de uma *shell*. Além dos argumentos também é fornecido ao interpretador o conjunto das variáveis ambiente do sistema operativo e o que é enviado para o *stdin*. Esta informação é construída num fragmento *XML* e associada à variável *#request*.

4.2 Estratégias de avaliação

O interpretador de *XPL* do *eXPLoit* suporta duas estratégias de avaliação - *lazy* e *greedy* - sendo esta última usada por omissão. No caso de uma avaliação *greedy*, o interpretador percorre todos os comandos do programa e executa-os na ordem em que estão definidos. Na avaliação *lazy*, são executados os comandos necessários para o *output* do programa *XPL*. A avaliação *lazy* permite que sejam avaliados apenas os comandos necessários para obter um determinado identificador. Desta

forma, comandos que demorem imenso tempo a serem processados, só o são processados se for necessário. Outra vantagem é a possibilidade de se colocar num programa *XPL* vários blocos de comandos, cada um deles responsável por um determinado *output*. Usando a avaliação *lazy*, a execução desses blocos seria controlado pela indicação do *output* desejado desse programa.

4.3 API

A classe *Interpreter*, responsável pela interpretação de um programa *XPL* comunica pela *API* por forma a inicializar e controlar o processo da interpretação. Os seguintes métodos são usados pelos adaptadores:

- *void setRequest(Document requestDoc)* - inicializa a variável *#request* no interpretador, sendo necessário fornecer um objecto da classe *Document*. Esta variável pode ser usada no resto do programa podendo aceder, por isso, a toda a informação proveniente do contexto de execução;
- *void setLazyEvaluation(boolean lazyEvaluation)* - define qual o método de avaliação que é usado na interpretação. Por defeito, é usada a avaliação *greedy*;
- *void eval(File xplProgramFile)* - indica ao interpretador um apontador para o ficheiro que contém o programa *XPL* a ser interpretado;
- *void eval(Document xplDocument)* - passa ao interpretador um documento cujo conteúdo é um programa *XPL*;
- *String getOutputAsString(String outputId)* - solicita ao interpretador o resultado final de um determinado output nomeado *outputId*. Este resultado é serializado pelo interpretador e retornado para o adaptador;
- *Document getOutputAsDocument(String outputId)* - efectua o mesmo pedido que o método anterior só que o resultado retornado será da forma de objecto *Document*;
- *Source getOutputAsSource(String outputId)* - retorna o mesmo que os métodos anteriores só que o resultado retornado será um objecto *Source*.

4.4 Processadores adicionais

Foram acrescentados à biblioteca básica do *XPL* novos processadores pelo *eXPLoit*. Outros podiam ser criados como, por exemplo, um processador que envie e receba *emails*, um processador que aceda a *RDBMS* ou um processador que guarde resultados de um programa num ficheiro.

A extensão de novos processadores ao *eXPLoit* efectua-se configurando o ficheiro *exploit.xml*. Este ficheiro contém a relação entre o nome do processador (que é indicado no elemento *p:processor*) e o caminho completo para a classe que o implementa. De seguida, são dados os processadores incluídos no *eXPLoit* para linguagem *XPL*:

- **xslt** O processador *xslt* necessita como *input* os atributos *config* e *data*. O *config* refere-se a um ficheiro *.xsl*. O atributo *data* corresponde aos dados

que se pretendem a que seja aplicada a transformação. O resultado será um *Infoset* que é obtido após o uso da transformação indicada em *config* sobre os dados *data*.

- **csv2xml** Este processador, mediante um conjunto de dados em formato *CSV*, cria um documento *XML* que os representa. Esses dados deverão ter como primeiro registo o nome dos campos dos restantes registos. Como parâmetros poderão ser passados separadores de registos e de campos. Por omissão, é usado '\n' (*newline*) e ';'. O resultado final é um documento que, além dos registos, contém a indicação do nome dos campos que cada registo possui.
- **webserviceRPC** Com este processador acede-se *webservices* usando *RPC*. Para se usar este processador, é necessário conhecer-se o ficheiro *WSDL* associado ao serviço que se pretende aceder.
- **webserviceMessage** A funcionalidade deste processador é idêntico ao anterior, só que ao invés de usar *RPC*, usa um outro método que consiste na troca de *Messages*.
- **googlewebservice** Este processador usa a *API*[12] da *Google* que permite aceder a um conjunto de serviços disponibilizados pela Google. São eles: a pesquisa de páginas, o acesso a páginas em cache e o uso de um *spell checker*. O retorno é dependente do tipo de serviço que for pedido.
- **xsdvalidator** Possibilita a validação de documentos *XML* mediante o uso de *XSD* (*XML Schema Definition*). É necessário indicar como input dois parâmetros: *config* e *data*. O primeiro é um ficheiro *.xsd* e o último os dados a serem validados. O retorno será *<validator result="valid"/>* se a validação suceder. Caso contrário o retorno é *<validator result="invalid"/>*.

5 Avaliação

Para ilustrar a flexibilidade do *eXPLoit* e o modo como se obtém dados de várias fontes e de vários ambientes, é dado como exemplo um programa que transforma dados do ficheiro */etc/passwd* existente no sistema *UNIX* num documento *HTML*.

O ficheiro */etc/passwd* é um ficheiro no formato *CSV* cujo delimitador de registos é uma *newline* e o separador de campos é o caracter **':'**. Cada registo deve conter os campos: nome do utilizador, palavra passe, *UID*, *GID*, nome completo do utilizador, directório da área pessoal e a shell a ser usada. De notar que o primeiro registo do ficheiro deve conter o nome dos campos dos registos seguintes. Estes campos são os elementos que figuram na conversão deste ficheiro para *XML*. Este ficheiro *XML* passa por um processo de validação. Para ser considerado válido deve possuir no mínimo três campos não vazios: nome de utilizador, *UID* e *GID*. Caso a validação suceda este é transformado num documento HTML.

A figura 2 mostra um diagrama do modo de funcionamento dos programas. Esta figura ilustra a forma como se pode reutilizar o mesmo programa (*passwd2html.xpl*) através do uso de *sub-pipelines*. De notar que o *sub-pipeline* é executado em dois ambientes de execução distintos: *shell* e *web*. Como vimos neste

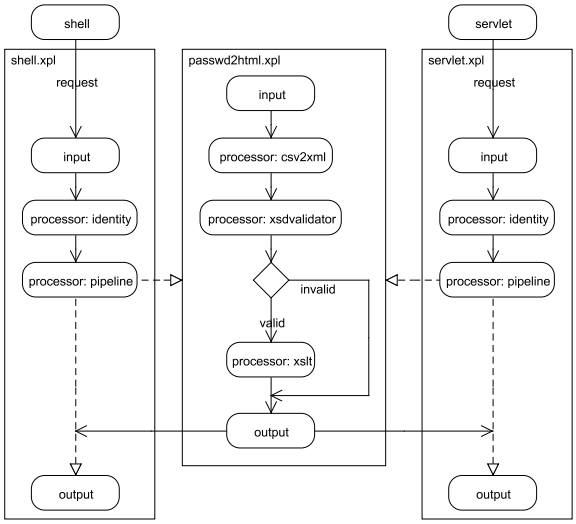

Figura 2. Execução do passwd2html em vários ambientes

artigo, os adaptadores do *eXPLoit* são responsáveis por enviar para o interpretador um fragmento *XML* que possua informação relativa ao pedido (*request*). No caso da *shell* o ficheiro é passado pelo *stdin* enquanto que no ambiente *web* este é enviado por *upload* através de um pedido *POST*. Cada um dos programas (*shell.xpl* e *servlet.xpl*) faz a ponte para a execução do sub-pipeline que efectua a transformação desejada. Esta ponte verifica-se inicializado variáveis através do processador *identity* e com expressões *XPointer*. Estas variáveis têm como conteúdo fragmentos *XML* que são passados como *input* para o sub-pipeline.

O sub-pipeline define uma sucessão de transformações. Em primeiro lugar, é executado o processador *csv2xml* que cria um fragmento *XML* com o conteúdo do ficheiro que faz parte do *request* e com a indicação dos delimitadores de registo e de campo. De seguida, este fragmento é validado conforme a as regras definidas no *XSD*. O *XSD* indica a estrutura que o fragmento *XML* deve possuir assim como a obrigatoriedade dos três campos falados anteriormente. Caso seja um fragmento válido este passa por uma transformação *XSL* que origina um fragmento *HTML* e que é o retorno final. Caso seja um *XML* inválido é indicado como retorno um documento *HTML* com uma mensagem de erro. Neste exemplo, o *output* do *sub-pipeline* é o resultado final.

O programa executado na *shell*, por omissão, imprime o conteúdo de todas as variáveis definidas como output do pipeline. A *servlet* retorna um resultado idêntico assim como o *header HTTP* respectivo. Quer o *header*, quer a variável de *output* desejada são configuráveis nos ficheiros de configuração do contentor de *Servlets*, como por exemplo, o *Tomcat*.

6 Conclusão

O exemplo da secção anterior mostra como o *eXPLoit* é modular na forma como permite interligar vários ambientes que executem o mesmo programa *XPL*. A acrescentar a isso junta-se a capacidade de se acrescentar, de uma forma simples, novos processadores para serem usados nos programas *XPL*. O sistema pode facilmente ser extendido com novos adaptadores para permitir a sua utilização noutros contextos. Um exemplo de um adaptador seria o uso do *eXPLoit* num sistema *GUI*.

O *eXPLoit* é uma aplicação que mostra as vantagens do uso de *pipelines* de *XML* em diferentes contextos. O facto de os fluxos de transformação serem baseados num conjunto de componentes, simplifica a adaptação destes componentes com o programa que se pretende obter. O uso desta abordagem permite que haja menos erros de programação por estes componentes funcionarem como funções pré-definidas de uma linguagem de alto nível.

Utilizando o *eXPLoit*, pretende-se que haja ganhos na implementação de aplicações *XML* pois a ocorrência de erros será menor e o tempo de implementação mais rápido. Dado que o uso de *pipelines* indicia uma abordagem usando-se componentes, o código torna-se modular e, por isso, reutilizável.

Como nota final e como projecto alternativo, sugere-se a criação de uma aplicação *GUI* para criar código de programas em *XPL*, usando-se gráficos que representem os vários componentes: parâmetros *input*/*output*, os comandos *standard* e chamadas a processadores. Traria vantagens porque em aplicações mais complexas, usar um editor *XML* torna-se impraticável.

Referências

1. Apache http://cocoon.apache.org/: (The Apache Cocoon Project)
2. Orbeon, Inc. http://www.orbeon.com/ops/: (Orbeon Forms)
3. World Wide Web http://en.wikipedia.org/wiki/XML_pipeline: (XML pipeline)
4. Bruchez, E., Vernet, A.: XML Pipeline Language (XPL) Version 1.0 (Draft). World Wide Web, http://www.w3.org/Submission/xpl/. (2005)
5. World Wide Web https://smallx.dev.java.net/: (smallx: Project Home Page)
6. World Wide Web http://servingxml.sourceforge.net/: (ServingXML)
7. Walsh, N., Milowski, A.: XProc: An XML Pipeline Language. World Wide Web, http://www.w3.org/TR/xproc/. (W3C Working Draft 17 November 2006)
8. Oracle http://www.oracle.com/technology/tech/xml/xdkhome.html: (Oracle XML Developer's Kit 10g)
9. Clark, J.: (XSL Transformations (XSLT) Version 1.0)
10. Clark, J., DeRose, S.: (XML Path Language (XPath) Version 1.0)
11. Steve DeRose, Ron Daniel Jr., P.G.E.M.J.M.N.W.: (XML Pointer Language (XPointer) Version 1.0)
12. Google http://code.google.com/apis/soapsearch/: (Google SOAP Search API)

DOM-like XML Parsing Providing Import and Export with Bounded Resources

Pedro Côrte-Real[1] pedro@pedrocr.net
Gabriel David[1,2] gtd@fe.up.pt
Silvestre Lacerda[3] lacerda@iantt.pt

[1] Faculdade de Engenharia da Universidade do Porto
[2] INESC – Porto
[3] Instituto dos Arquivos Nacionais/Torre do Tombo

Abstract. XML parsing is usually done using DOM or SAX. DOM is memory-bound while SAX provides a fairly low level of abstraction. We propose an approach to create an object-oriented mapping of an XML format. Based on the resulting representation, API's for reading and writing complete documents in a DOM-like way are provided. To handle very large documents we propose an event-based API that returns full elements as events and a second API to create unlimited sized documents with memory usage proportional to the maximum tree depth.

1 Introduction

The most common models for XML interpretation and generation are DOM[3] and SAX[2]. DOM gives you access to the whole document at once by keeping it all in memory at the same time. This makes the memory usage grow linearly with the document size. SAX solves this limitation by allowing the instrumentation of parser events but is a bit cumbersome to program since the events are fine grained requiring applications to handle them at a low level of abstraction. StAX[6] is analogous to SAX but provides a pull-style API although it still does a very fine grained parsing of XML. Because of this, programmers tend to end up using DOM as long as it is feasible and switching to SAX or StAX when the datasets become too large. This results in a complete change in the way to access the same XML format.

The solution proposed to the problem of XML parsing relies on first creating a mapping of a XML format to an object-oriented structure. Similar approaches have been adopted before[14,5,7]. The proposed solution is more lightweight without any code generation or automatic conversion from DTD's or XML Schemas[4].

API's using the object mapping are provided to import and export documents as a whole, as would be done with DOM. An event parser where events are complete elements mapped to the same objects is provided to allow the importing of unlimited sized documents. A way to export unlimited-sized documents is also provided by filling it in memory much the same way as with the DOM-like

functionality but periodically pushing completed parts to disk and removing them from memory.

With these techniques the importing and exporting API's use the same objects regardless of the size of the dataset, making it easier to evolve a program to handle larger XML documents. These techniques have been implemented in a Ruby framework released as GPL-licensed free software[1,8,10].

2 Mapping XML to Objects

To map a XML format to an object structure a simple approach has been chosen. Each element in the format corresponds to a class and each attribute or sub-element is reached by an accessor in the class. Attributes can be modelled as simple get/set style methods. The sub-elements are stored in a list in the parent element, so as to support the preservation of order. To be able to handle mixed-content elements this list may also contain strings.

No attempt at type conversion was made, so everything is either text or an object representing an element. This is not however a limitation of the technique but of the implementation. We expect that existing approaches in this area[12] would integrate easily with our work.

The implementation is fully functional. Here is an example of mapping a simple XML format into Ruby Objects:

```
require 'rubygems'
require 'xmlcodec'

class SimpleFormat < XMLCodec::XMLElement
  xmlformat 'Race Scores'
end

class Race < SimpleFormat
  elname 'race'
  xmlattr :name
  xmlsubelements
end

class Car < SimpleFormat
  elname 'car'
  xmlattr :name
  xmlattr :number
  xmlsubel :result
end

class Result < SimpleFormat
  elname 'result'
  xmlattr :time
  xmlattr :place
end
```

Although this might look like pseudo-code it is actually fully functional Ruby code. The `elname`, `xmlformat`, `xmlattr`, `xmlsubel` and `xmlsubelements` lines are calls to class level functions inherited from `XMLElement` that are executed when the classes are interpreted and setup all of the XML import/export functionality. If we wanted to create some race results with two cars we'd write:

```
race = Race.new

car1 = Car.new
car1.name = 'Speeding Bullet'
car1.number = 17
result1 = car1.result = Result.new
result1.time = '1 hour 12 min'
result1.place = 2
race << car1

car2 = Car.new
car2.name = 'Overspeeding Bullet'
car2.number = 12
result2 = car2.result = Result.new
result2.time = '1 hour 11 min'
result2.place = 1
race << car2
```

From now on the `race` object will be completely filled with the race information. To generate XML all that's needed is to run `race.xml_text`, which will generate the following document:

```
<?xml version="1.0"?>
<race>
  <car number="17" name="Speeding Bullet">
    <result time="1 hour 12 min" place="2"/>
  </car>
  <car number="12" name="Overspeeding Bullet">
    <result time="1 hour 11 min" place="1"/>
  </car>
</race>
```

To import a XML document something simple like this can be done:

```
xmltext = '<race></race>'
race = SimpleFormat.import_xml_text(xmltext)
```

The `race` object will link to the full representation of the XML tree. The framework includes a little more functionality than was explained here including importing/exporting from a DOM object model instead of XML text. See the framework website and the API docs for more information[10,11].

3 Event Parsing Using Objects

In the previous section we explained how the mapping between XML and objects has been solved. Now we take those same objects and use them together with a standard XML stream parser to produce an event-based XML parser where each event returns a complete object.

The parser is implemented by keeping the current XML element as well as all its ancestors in a stack. The top of the stack will always contain the element currently being parsed. Whenever a new XML element starts a new object is created in the stack and it is connected to its parent so that the XML tree is recreated in memory. Any text content is also added.

Every time a XML element closes we have a complete representation of it in memory and can thus generate an event ourselves to any listener, informing that a full element has been parsed and passing it along to be used.

If nothing else is done the event that closes the root element of the XML document returns an object that indirectly links the whole tree. Although this might be useful for some purposes it would impose a memory limit on the size of the document that could be parsed. To solve this a listener can choose to consume the element during the processing of the event. In this case what happens is that the element is removed from the tree and will not appear as one of the children of its parent when the event for that element comes.

This technique will allow the parsing of an unlimited sized document with memory usage proportional to the maximum depth of the XML tree. If the tree has unbounded depth the memory usage will not be bounded and we won't be able to parse unlimited sized documents. In the real-world big XML documents tend to be very wide instead of deep.

If you do find such a difficult case of an XML document that is huge by being deeply nested even a simple stream parser will consume unbounded amounts of memory and a totally different parsing technique needs to be employed or perhaps discovered.

4 Generating Unlimited-Sized Documents with Objects

Now that we've seen how it is possible to parse a document of unlimited size with low memory usage we need a way to generate such a document. There is no usual equivalent to the stream parser for document generation. Software tends to use DOM when feasible and its own exporter when necessary.

The XML to object mapping we've described can be used the same way as DOM is usually used with the benefit of a better API. To do this we first create the whole XML tree in memory, represented by objects, and then write it to disk in one go.

To be able to use the same API and not keep the whole document in memory while we create it we'd have to implement a fairly complicated virtual memory system so that we could transfer elements back and forth from disk. To make this much simpler all we have to do is impose the restriction that the tree is

filled depth-first and left-to-right. This is the order used in the XML document itself.

If the in memory tree is filled in the same order as the elements appear in the XML text we can continuously write elements to disk and remove them from memory. To do this we use the API as usual but call for the partial export of an element when we are done with it.

What the partial export does is append the text of the element to the XML file and then remove the element from its parent. To be able to do this though we must first at least write the opening tags of the parent elements. This is handled by separating the partial export process into two steps. The first step writes the opening tag for the element with any attributes followed by any sub-elements that have been added to the parent. The second step writes the close tag for the element and removes it from the parent. Between these two steps more elements can be added to the element and exported to disk.

We don't have to fill the tree in the exact same order as the XML text is written. The actual restriction is that after we call to export a certain element we must not change or add any element that would be before it in the XML text since that point in the file has already passed.

Once again this allows the creation of unlimited sized documents with memory usage proportional to the depth of the tree

5 Practical Application

This framework and techniques have been created to solve the problem of creating and interpreting large EAD[13] XML files. Both scalable import and export techniques were used to produce and consume EAD files with sizes of over 100 megabytes.

Because the framework has been extracted from the concrete solution the separation between the two proved very clean. Import/export techniques were first implemented for the EAD case and later made generic. The result is that in the end the EAD library became just a set of class definitions for the format's elements, like the simple example given to illustrate the XML to object mapping. This library has also been released as free software[9].

This application shows that an import/export library can be defined for a specific XML format without hand-coding a lot of infrastructure and still get a scalable library, capable of handling size-unlimited XML files with no size limits.

The library has been developed to handle the EAD creation and parsing needs at *Instituto dos Arquivos Nacionais/Torre do Tombo* (IANTT). The production archive system uses a proprietary import/export format. The two main goals for EAD use where to export and convert the full set of existing descriptions into the format and then make them available in a Web application with full search capabilities.

We'll now see how the import and export API's can be used in a real use scenario to import and export large quantities of EAD. No deep knowledge of the EAD format is required to understand these examples. It suffices for this purpose

to understand that EAD is an XML format to describe a tree of records. Each record is an `archdesc` XML element if it's the root or `c` element otherwise. The tree is described in XML by nesting the `c` elements.

5.1 Generating Large Quantities of EAD XML Files

The archive system at IANTT has an underlying data model based entirely on text fields. Each record is just a set of strings for the fields. Connections between fields are done by brute-force or indexed search over these fields. The record tree is inferred from the reference codes rather than being kept by linking records to each other.

Exporting the contents of the archive system results in a single file containing all the records in sequence. This file has been indexed so that it could be traversed in the lexicographic order of the reference codes for each record. This is the same as visiting the tree of records in pre-order because reference codes are defined much like Unix paths. For example `PT-TT-TSO/IC` is the immediate child of `PT-TT-TSO`.

Using the index of the records the method described in Section 4 was used. For each of the roots of the description tree a single EAD file containing the corresponding tree has been created as follows:

```
1  @curdepth = -1
2  @index.each_with_depth do |obj, depth|
3    if depth == 0
4      change_file(obj)
5    else
6      newel = EADCodec::Level.new
7      if depth > @curdepth
8        @curelements.push(newel)
9      else # depth <= curdepth
10       (@curdepth - depth + 1).times do
11         @curelements[-1].end_partial_export(@file)
12         @curelements.pop
13       end
14       @curelements[-1] << newel
15       @curelements.push(newel)
16     end
17   end
18   @curdepth = depth
19
20   fill_element(@curelements[-1], obj)
21 end
```

Line 2 shows an iteration over every single record. This iteration receives for each element an object encapsulating the text fields of the record and the tree depth we are currently in. The depth is calculated using the reference code. For each record a new object is created representing the EAD element (line 6).

All the ancestors of the element currently being processed are kept in a stack. To maintain this stack we first find out if the depth of the tree has increased (line 7). If so the new element is pushed into the stack. If not the stack is popped to the depth immediately above the one we're currently at and each of the elements is finished and written to disk (lines 10 through 13). The current element is then added to the EAD tree (line 14) and pushed into the stack (line 15).

At the end of each iteration the depth is saved to be used on the next (line 18). Finally the EAD XML element is filled with the contents from the record (line 20). This implements the mapping between the archive system's set of fields and the EAD format. `fill_element` is a rather large method so it will not be listed here. It's operation is however quite simple. The partial export API (as all the API's described) uses the same objects as the more traditional API's so the method fills the object in much the same way as was done in Section 2.

The tree depth is zero when we're in one of the roots so we change the file that's currently being written (lines 3 and 4). `change_file` is defined as:

```
1 def change_file(obj)
2   @ead.end_partial_export(@file) if @ead
3   @file = File.new(obj.RefNo+'.xml', 'w')
4   @ead = EADCodec::Document.new(obj.RefNo, '...')
5   @curelements = [@ead.archdesc]
6 end
```

`change_file` starts by finishing the export of the previous file by calling for the end of the export on the root element (line 2). The framework takes care of recursing down the XML tree and finishing all of its still open elements. After that a new file is created in the filesystem (line 3) and a new EAD document object is created (line 4). The element stack is initialised with a single element, the root of the element tree which is the `archdesc` element (line 5). At this point we could do a call to the partial export methods to export the beginning of the EAD file. This is actually unnecessary since it will be taken care of automatically by the framework the first time an element is exported (line 11 of the previous code listing).

The export from the archive system is about 800 megabytes. After conversion to EAD through the process just described a few hundred files are created with a total size of around 300 megabytes. The process takes approximately 2 hours on a modest computer (3.0Ghz Celeron processor with 1 GiB of memory), most of which is spent indexing the exported file and not doing the conversion itself.

The conversion process takes time proportional to the number of records in time and constant space. The indexing process makes the full conversion process have memory usage linearly proportional to the number of records because it needs to create an index of all of them, sorted by reference code. Time is of NlogN order because of the sorting step. This turned out not to be a problem. The index is small; all it stores is the records' codes along with their offset within the file so that their contents can be retrieved.

Most of the complexity of this process lies in recreating the tree from the flat set of records the archive system provides us with. Besides the `fill_element` method there's just three lines of code that deal with the EAD exporting (line 11 in the export loop and lines 2 and 4 in `change_file`). `fill_element` itself doesn't do any calls to the partial export process and could be used without modification with the more classic export API's. The partial export API is thus very simple to use although it does require care in setting up the correct looping procedure to serially generate the XML tree.

5.2 Parsing and Importing Large Quantities of EAD XML Files

After generating these EAD files a Web application has been created to index and search them. We'll now see how large EAD files can be parsed and processed using the event parser described in Section 3. The code to implement this is extremely simple:

```
1  files.each do |file|
2    l = MyEADStreamListener.new
3    el = EADElement
4    parser = XMLStreamObjectParser.new(el, l)
5    parser.parse(File.new(file))
6  end
7
8  class MyEADStreamListener
9    def el_c(el)
10     handle_object(el)
11   end
12
13   def el_archdesc(el)
14     handle_object(el)
15   end
16
17   def handle_object(o)
18     # ... Do something with this description ... #
19     o.consume
20   end
21 end
```

The main loop (lines 1 through 5) is very simple. For each of the files it creates a listener object (line 2), then a stream parser with the listener and the root element for the EAD format (line 4). Finally the parser is told to parse the file by passing it a file object (line 5).

The stream parser will parse each of the XML files and for each element within them call the listener. The API works almost the same way as a SAX parser. The difference is that the events are full elements instead of the opening and closing of tags. The listener will contain one or several methods named

`el_tagname`. For each of the elements found in the XML file the parser will call the corresponding method in the listener if it exists.

Lines 8 through 21 show an example of a listener for the EAD format. There are methods to listen to `c` and `archdesc` elements that both call `handle_object` to do the actual processing since these elements are similar. `handle_object` will do something with the object and then consume it (line 19) removing it from the XML tree and allowing the memory for it to be freed.

In the Web application `handle_object` adds a record to a database containing the contents of the description as well as adding it to a textual index for searching. Full imports of large EAD files have been successfully completed. Memory usage is constant and running time is proportional to the number of records in the EAD file.

The final version of the application ended up not using this parser. Because of unrelated architectural decisions it sufficed to use a simpler parser of the same type that instead of returning elements as Ruby objects containing the parsed XML content just returned the XML text itself. This could be achieved by doing `o.xml_text` inside `handle_object`. It is however much more efficient to not create the objects to represent the XML structure.

6 Comparison with Other Approaches

We will now compare the existing technologies to the proposed solution. Table 1 shows six different technologies and a set of comparison points. The first two columns indicate whether the API's are capable of both parsing and generating XML. All but SAX do both. The Generating equivalent of SAX is usually to output XML manually by writing the file directly. StAX has a much friendlier way to do essentially the same thing while DOM does it by using the same in-memory structure it creates when parsing to generate XML.

Name	*Parse*	*Generate*	*Mapping*	*Validating*	*API Type*	*Space Used*
XMLCodec	**Yes**	**Yes**	**Yes**	No	Push	**O(Depth)**
JAXB	**Yes**	**Yes**	**Yes**	**Yes**	Push	O(Size)
XML Beans	**Yes**	**Yes**	**Yes**	**Yes**	Push	O(Size)
StAX	**Yes**	**Yes**	No	No	Push	**O(1)**
DOM	**Yes**	**Yes**	No	No	Push	O(Size)
SAX	**Yes**	No	No	No	**Pull**	**O(1)**

Table 1. Feature Matrix of the Various Approaches

JAXB, XML Beans and XMLCodec are all capable of performing the mapping between XML and objects. In this respect our proposal has the most modest feature set. JAXB and XML Beans support full type systems and richer API's. Somewhat as a result of this they also support validating the XML while our solution does not.

StAX, unlike all others, has a push-style API. This means that the control of the advance of the XML parsing process is done by the caller and not the API.

As for the space usage or complexity, JAXB, XML Beans and DOM all use space proportional to the XML document size, since they load it all into memory at once. As we've shown our approach has space complexity proportional to the document's depth. StAX and SAX use constant space since they don't need to keep the state of the XML tree around. This is not usually an actual advantage because most of the applications that use these API's will almost surely have to add a layer above them that will at least keep track of all the elements in the current depth expansion of the XML tree and will thus use space proportional to it's depth.

Our solution equals or improves the common ones over most of the criteria with which XML processing API's are usually compared. It is behind in some areas not because of inherent problems with the approach but because the current implementation is still not full-featured. The only area where the approach has an actual inherent limitation is in space efficiency. As we've seen this is only important in a very small number of cases where SAX or StAX low-level parsers will have to be used. When compared to its most natural competitors like JAXB or XML Beans its space efficiency is a clear improvement.

7 Further Work

The XML mapping to objects and the import/export API's were built out of necessity to solve a particular problem set. The implementation turned out stable and functional enough to suggest several avenues of further work.

7.1 Performance Work

All of the work on performance improvement centred around optimising for space and not time. The technique itself is not algorithmically complex but the current implementation is somewhat naive and unoptimised. There has been some work done in caching some frequent operations and some simple code optimisations. Profiling and optimisation work could probably speed it up a fair amount.

7.2 Validation

The XML mapping has no support for the implementation of validation of XML elements. It would be simple to add support for checks performed when importing and exporting elements.

7.3 Pull-Style API

StAX is different from all other common XML API's in that it gives control over the advance of the parsing process to the caller. This is orthogonal to the parsing

technique described. Our implementation uses a SAX parser but the technique can be just as easily implemented using StAX.

The API for the pull parser would reverse the calling process. Instead of the caller providing a listener to respond to parser events it would call the parser repeatedly to process the XML. Each call would return an object representing a XML element instead of a XML instruction, tag or text. This new parser would be to StAX what our parser is to SAX.

7.4 Auto-Generation from Schemas or DTDs

Writing a XML mapping requires writing a class for each of the elements in the format. It would be feasible to automatically create these classes from a XML schema or DTD. It would then be possible to create the validation rules automatically. This would be similar to what XMLBeans and JAXB do [5,7].

7.5 Generic XML Import/Export

Although the framework has been generalized to work with any XML format it still requires classes to be defined for each element type, instantiating the framework as an import/export library for a specific format. This isn't strictly required for the mapping techniques to work. It would be possible, and relatively easy, to create a generic element type that is instantiated with the element name and to which attributes and elements are added. The import and export techniques could still work with it.

Supporting generic XML import/export would turn the framework into a DOM-style API with the possibility of using the techniques described for handling large documents.

7.6 Virtual-Memory Style Import and Export

As briefly pointed out when describing the framework, the partial export method avoids having to implement a full VM-style abstraction by limiting the filling of the in memory structure to the XML text order and by removing elements from memory after they've been exported to disk. Importing a large document is handled by a stream parser whose events are complete elements. This is useful in a large number of situations but might require several passes through a document for some workloads.

A unified solution to the limitations imposed by these methods would be to treat the XML document as the on-disk representation of the in memory structure and implement the VM-style abstraction that would make that distinction transparent to the API.

This is feasible if not trivial but much care would be needed with the space-time trade-off of such an implementation. The techniques presented were designed explicitly to be both space and time efficient by limiting the XML processing to use the text order. Random access requires navigating back and forth in the structure.

If we keep a full skeleton structure in memory, random access will be fast and memory-hungry since it will only hit the disk to fetch content. Another choice is to only keep in memory whatever element pointers the user has. This will be memory-lean but slow because it must hit disk for every navigation within the tree. A compromise between the two can probably be made by going with the second option but introducing a dynamically sized cache that can be configured to a desired size.

8 Conclusion

The techniques explained aren't just proposals, a fully-functional implementation exists and they've been tested in production environments with large documents. It has been shown that mapping XML to objects can be simple yet API-rich. The techniques for importing and exporting large XML documents have proved useful and shown adequate performance. Further work along these lines could completely break the current separation between DOM and SAX techniques and how they are used.

Although the techniques described are independent of the programming language the current implementation and the way that the XML mapping works would not be possible without the expressiveness of Ruby. All of what's described here has been implemented in less than a thousand lines of code and has been fully unit tested in less than 800 lines. It has also been a joy to write.

References

1. Ruby Programming Language. `http://www.ruby-lang.org/en/`.
2. Simple API for XML. `http://www.saxproject.org/`.
3. W3C Document Object Model. `http://www.w3.org/DOM/`.
4. XML Schema. `http://www.w3.org/XML/Schema`.
5. XMLBeans. `http://xmlbeans.apache.org/overview.html`.
6. BEA Systems Inc. JSR 173: Streaming API for XML. `http://jcp.org/en/jsr/detail?id=173`, October 2003.
7. Ed Ort and Bhakti Mehta. Java Architecture for XML Binding (JAXB). `http://java.sun.com/developer/technicalArticles/WebServices/jaxb/`, March 2003.
8. Free Software Foundation. GNU General Public License. `http://www.gnu.org/copyleft/gpl.html`, June 1991.
9. Pedro Côrte-Real. eadcodec - an EAD importer/exporter library for Ruby. `http://eadcodec.rubyforge.org/`.
10. Pedro Côrte-Real. xmlcodec - a XML importer/exporter framework for Ruby. `http://xmlcodec.rubyforge.org/`.
11. Pedro Côrte-Real. xmlcodec API. `http://xmlcodec.rubyforge.org/doc/`.
12. R. Connor, D. Lievens, P. Manghi, and F. Simeoni. Extracting typed values from XML data, September 2001.
13. SAA Encoded Archival Description Working Group. *Encoded Archival Description Tag Library*. The Society of American Archivists, August 2002.
14. F. Simeoni, D. Lievens, R. Connor, and P. Manghi. Language Bindings to XML, 2003.

Agrupamento e Consulta a Versões de Documentos XML num Ambiente *Peer-to-Peer*

Deise de Brum Saccol[1], Felipe dos Santos Giacomel, Renata de Matos Galante, Nina Edelweiss

Instituto de Informática - Universidade Federal do Rio Grande do Sul
UFRGS - Porto Alegre – RS – Brasil

{deise, fsgiacomel, galante, nina}@inf.ufrgs.br

Abstract. O conceito de versão é bastante explorado no gerenciamento de configuração de *software*. Técnicas de versionamento também têm sido utilizadas no gerenciamento de documentos XML. No entanto, a característica da distribuição em um ambiente *peer-to-peer* (P2P) coloca alguns novos desafios à área de versionamento. Documentos versionados podem estar armazenados em um único *peer* ou espalhados por vários *peers*. Neste contexto, gerenciar e acessar versões de documentos é de fundamental importância, principalmente para possibilitar recuperação histórica de documentos em um ambiente distribuído. Este trabalho propõe um mecanismo de agrupamento e consulta à versões de documentos XML, inserido na arquitetura do *DetVX*, um ambiente para detecção de réplicas e versões XML em um contexto P2P.

Palavras-Chave: XML, Peer-to-Peer, Versionamento.

1. Introdução

O paradigma P2P baseia-se nos conceitos de descentralização e compartilhamento de recursos, objetivando evitar gargalos em servidores centrais e distribuir a carga de trabalho [1]. Cada *peer* do sistema atua tanto como cliente quanto como servidor, e fornece parte dos recursos e informações disponíveis no sistema. Embora a idéia de compartilhamento seja comum a todos os sistemas P2P, a diferença entre eles reside, principalmente, no tipo de arquitetura adotada [19].

A arquitetura *parcialmente centralizada* (ou *híbrida*) contém um servidor central responsável pelo mecanismo de busca e pela manutenção da infra-estrutura; o compartilhamento de recursos fica sob a responsabilidade dos *peers*. *Napster*[2] é um exemplo de sistema que pertence à este tipo de arquitetura. O segundo tipo de arquitetura, chamada *pura* ou *distribuída*, apresenta um funcionamento totalmente descentralizado, onde cada *peer* é responsável pelo mecanismo de busca e pela manutenção da infra-estrutura. *Gnutella*[3] é um exemplo que utiliza esta arquitetura. O último tipo de arquitetura

[1] Este trabalho é parcialmente financiado pelo CNPQ (processo 142396/2004-4), CAPES (processo 1451/06-5) e projeto PERMXML (CNPq 475743/04-0, Edital CNPq 19/2004 – Universal).

[2] http://www.napster.com

[3] http://www.gnutella.com

agrega um subconjunto de *peers* em *super peers*. Neste tipo de arquitetura, a comunicação é estabelecida somente entre os *super peers*, e destes com os seus *peers* agregados, o que pode contribuir para tornar a pesquisa mais rápida. *Kazza*[4] implementa esta arquitetura.

Independente da arquitetura utilizada, um problema que pode ser observado nestes ambientes advém do comportamento evolutivo e dinâmico de alguns recursos disponíveis na rede P2P. Documentos XML compartilhados podem sofrer atualizações de conteúdo e estrutura, criando vários estados possíveis do mesmo recurso ao longo do tempo. A característica de passar por atualizações é um aspecto fundamental de qualquer sistema de informações persistentes. No contexto da pesquisa em banco de dados, o gerenciamento de tempo tem sido bastante estudado nas últimas décadas [2], [10], [11]. Muitos trabalhos foram desenvolvidos para adicionar tempo à modelos de Banco de Dados e fornecer capacidades de armazenamento, consulta e atualização de dados históricos [8], [9], [13].

A característica de temporalidade em documentos XML compartilhados em ambientes P2P pode ser tratada com o uso de versões. Através do uso de versões, pode-se preservar o conteúdo antigo e o conteúdo atual do documento, possibilitando representação da evolução temporal e acesso aos estados passados do documento. No entanto, em um ambiente P2P, versões de documentos são disponibizadas na rede em arquivos físicos separados, os quais podem estar armazenados em um único *peer* ou em *peers* diferentes. Se o usuário submete uma consulta a um *peer* específico solicitando o endereço atual de uma pessoa, o sistema deve retornar somente a última versão desta informação. Supondo que os dados existentes no *peer* onde a consulta foi submetida estejam desatualizados, então a consulta deve ser roteada para outros *peers*. Um outro caso é o usuário submeter consultas que retornam o histórico de documentos. Uma vez que as versões podem estar espalhadas em vários *peers*, recuperar o histórico de documentos implica em executar consultas distribuídas pela rede.

Para evitar o roteamento de consultas desta natureza, este artigo propõe *XVersion*, uma ferramenta para agrupamento e consulta à versões de documentos XML. O mecanismo proposto consiste em agrupar versões de documentos XML em um único arquivo que contém todo o histórico de modificações do documento. Desta forma, um único arquivo pode ser acessado para recuperar toda a história de um determinado recurso. A implementação desta ferramenta está inserida dentro do *DetVX*, um ambiente para detecção de réplicas e versões de documentos XML em contextos P2P.

A seção 2 do artigo apresenta os trabalhos relacionados. A seção 3 apresenta o ambiente *DetVX*, contexto onde o artigo está inserido. A seção 4 apresenta o mecanismo proposto para agrupamento de documentos XML, visando recuperação histórica de recursos espalhados em uma rede P2P. O protótipo da ferramenta é apresentado na seção 5. Conclusões e perspectivas futuras são apresentadas na seção 6.

2. Trabalhos Relacionados

Gerenciamento de tempo tem sido bastante estudado nos últimos anos [13]. Uma possível abordagem para tratar o aspecto evolutivo é através do uso de versões [4]. Algumas técnicas de versionamento foram propostas para gerenciamento de tempo de transação em bancos de dados temporais [11], versões de artefatos em bancos de dados orientados a objetos [10] e gerenciamento de mudanças em dados semi-estruturados [23]. O conceito de versão também já é bem conhecido no gerenciamento de configuração de *software* [7],[15].

[4] http://www.kazaa.com

No entanto, sistemas tradicionais de controle de versões modelam documentos como uma sequência de linhas de texto e utilizam *scripts* para representar diferenças entre versões. Estes sistemas mostram-se inadequados para representar versões de documentos XML [4],[14], uma vez que não preservam a estrutura lógica do documento original [3] e não suportam consultas complexas.

Desta forma, alguns modelos foram propostos para representar o caráter evolutivo de dados XML [20]. A proposta de [32] apresenta algumas técnicas para gerenciamento e consultas temporais de documentos multiversionados. A proposta consiste em representar as sucessivas versões de um documento XML como um outro documento XML que implementa um modelo de dados temporal. A abordagem utiliza algoritmos *diff* para processar os tempos de validade dos elementos no documento [33]. Por fim, a proposta utiliza *XQuery* para formular consultas temporais nos documentos, a partir do uso de funções temporais que podem ser escritas pelo usuário.

Outros trabalhos incluem: em [24] os autores propõem o uso de uma *tag* de marcação *valid* para documentos XML/HTML, a fim de possibilitar visualização temporal em *browsers* utilizando-se XSLT. Um método baseado em dimensões é proposto em [25] para gerenciar mudanças em documentos XML, mas não é mostrado como as consultas temporais poderiam ser tratadas. Em [26] e [27] também é proposto um modelo de dados temporal para representar documentos XML; o suporte à consultas é feito via extensões à API DOM e à linguagem *XPath*. O trabalho de [28] propõe uma outra extensão da linguagem *XPath* para possibilitar consultas temporais. Em [21] é proposta a linguagem τXQuery, uma extensão à linguagem *XQuery* com suporte à temporalidade. O trabalho de [29] propõe uma técnica de versionamento baseada em rótulos temporais e números de nós duráveis para preservar a estrutura e o histórico dos documentos durante a evolução. [14] e [30] propõem um mecanismo para possibilitar consultas baseadas em expressões de caminho para documentos XML versionados; é proposta uma representação de documentos que mantém relacionamentos entre nós de documentos. Por fim, [31] apresenta um mecanismo para gerenciamento temporal de textos em XML através do uso de dimensões temporais e metadados para possibilitar consultas com base em restrições temporais.

A maioria das propostas propõem extensões ao modelo XML para a representação do aspecto evolutivo e algumas estensões à linguagens de consulta existentes [21] ou funções definidas pelo usuário [22], [33] para possibilitar recuperação temporal. Além disso, as propostas apresentadas focam na representação e gerenciamento de versões já conhecidas; o processo de detecção de versões não é tratado. A característica distribuída inerente aos ambientes P2P torna o problema de gerenciamento e consulta à dados versionados mais complexo. Sistemas P2P geralmente utilizam técnicas tradicionais de busca em largura e profundidade, mas não tratam o problema da existência de recursos versionados.

O ambiente em desenvolvimento, *DetVX*, foca nesta lacuna e propõe um ambiente para detecção, gerenciamento e consulta à versões de documentos XML em um contexto distribuído. O ambiente proposto não faz qualquer extensão ao modelo de dados XML para a representação do aspecto temporal dos documentos. Consultas temporais podem ser expressas atráves do linguagem *XQuery* padrão, sem o uso de extensões e sem o uso de funções temporais. Desta forma, operadores temporais básicos, como *after*, *before e now*, podem ser utilizados no acesso aos documentos versionados através de filtros nos rótulos temporais, o que torna o uso do sistema e da linguagem *XQuery* bastante intuitivos na submissão de consultas.

3. Ambiente DetVX

No ambiente *DetVX* (**Det**ecção de Réplicas e **V**ersões de Documentos **X**ML), cada *peer* atua como cliente e servidor. Documentos compartilhados estão armazenados em *peers*, segundo uma arquitetura *super peer*. Neste tipo de arquitetura, o usuário submete uma consulta a um *peer* específico. Se a consulta não puder ser respondida localmente, ela é roteada para o *super peer* do *peer* solicitante, o qual verifica seus *peers* disponíveis que estão aptos a responder a consulta e reenvia a consulta a estes *peers*. Os *peers* processam a consulta e retornam os resultados ao *super peer*, o qual os envia ao usuário final.

O critério de agrupamento de *peers* em *super peers* baseia-se no domínio de conhecimento ao qual os documentos do *peer* pertencem. Para a representação dos conceitos e relações, utiliza-se uma ontologia. Assume-se que cada *super peer* possui pelo menos uma ontologia associada a ele. Desta forma, um *super peer* pode agregar *peers* que contenham documentos pertencentes a vários domínios. Assume-se que existe uma entidade central responsável pelo gerenciamento do ambiente (comum em sistemas P2P que adotam a arquitetura parcialmente centralizada), conforme mostrado na Figura 1.

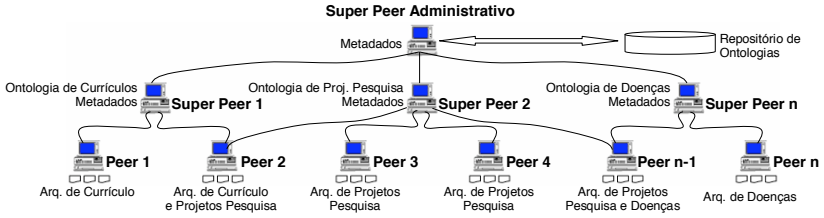

Fig. 1. Ambiente *DetVX*

Cada *super peer* possui metadados com informações sobre seus *peers* agregados e seus respectivos documentos registrados, tais como *data de registro* e *última data de modificação* de cada arquivo no *peer*. O *super peer* administrativo funciona como responsável pelo gerenciamento da rede. Os metadados do *super peer* administrativo descrevem os *super peers* existentes e seus respectivos domínios de aplicação, além de outras informações necessárias ao gerenciamento do ambiente.

Versões e réplicas de documentos podem estar localizadas em um único *peer* ou em *peers* diferentes. Se o usuário submete uma consulta a um *peer* específico solicitando o endereço atual de uma pessoa, o sistema deve retornar somente a última versão desta informação. Supondo que os dados existentes no *peer* onde a consulta foi submetida estejam desatualizados, então a consulta deve ser roteada para outros *peers*. Para descobrir qual(is) documento(s) é(são) necessário(s) acessar para responder uma determinada solicitação, o sistema consulta os metadados disponíveis no *super peer*.

A fim de fornecer as funcionalidades para o ambiente de detecção e gerenciamento de versões, este trabalho propõe a arquitetura mostrada na Figura 2. De maneira geral, o usuário interage com o sistema via uma interface, a qual possibilita registrar *peers* e documentos no ambiente (através do gerenciador de *peers*) e submeter consultas posteriores aos recursos compartilhados (através do processador de consultas).

Os módulos propostos na arquitetura possuem as seguintes funcionalidades:

- gerenciador de *peers*: responsável por conectar e reconectar *peers* ao sistema, além de verificar periodicamente a ocorrência de modificações nos arquivos compartilhados;
- gerenciador de ontologias: responsável pela manutenção do repositório de ontologias e pela associação de ontologias a *super peers*;

- gerenciador de réplicas e versões: responsável por identificar e representar réplicas e versões de documentos;
- processador de consultas: responsável por verificar à qual domínio uma consulta pertence e roteá-la ao *peer* adequado, com base nos metadados disponíveis.

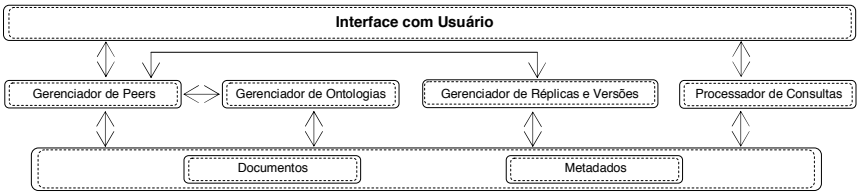

Fig. 2. Arquitetura do ambiente *DetVX*

Os módulos fazem uso intensivo de metadados para representação da informação. Metadados são representados como documentos XML e são classificados em dois níveis: metadados do *super peer* administrativo e metadados do *super peer*. Os metadados basicamente descrevem: *super peers* existentes e respectivos *peers* agregados; identificadores locais, tempos de registro e última data de atualização de cada arquivo no *peer; peers* agregados e seus respectivos documentos registrados; versões e réplicas disponíveis em um determinado *super peer;* e rótulos temporais correspondentes a cada elemento (*TS*,*TE*) encontrado em um dado documento de um *peer*. Rótulos temporais são derivados a partir das datas de modificação dos arquivos, e visam representar a evolução temporal de cada elemento, juntamente com seus respectivos tempos de validade. Este artigo foca nos módulos de gerenciamento de versões e processamento de consultas, descritos em mais detalhes nas seções a seguir.

4. Gerenciamento de Versões de Documentos

O trabalho proposto assume que a evolução dos documentos cria uma seqüência linear de versões temporalmente ordenadas: V_1, V_2, ..., V_n, onde a última versão V_n é a atual. Uma versão pode possuir apenas uma versão antecessora e outra sucessora. A fim de ordenar as versões de arquivos em uma linha de tempo, leva-se em consideração os tempos de modificação destes arquivos. Assume-se que o arquivo com tempo de modificação mais recente é uma derivação de uma versão anterior. Supondo duas versões de arquivos, *F1* e *F2*, com as respectivas datas de modificação *10/10/2005* e *03/04/2006*, conclui-se que o arquivo *F2* é uma versão modificada do arquivo *F1*.

A detecção de versões baseia-se na similaridade entre documentos. Em uma abordagem simples, se dois arquivos possuem um conjunto pequeno de diferenças significativas, então eles são considerados duas versões do mesmo documento. Se eles possuem um conjunto grande de diferenças significativas, então são considerados dois documentos diferentes. Para definir o ponto de corte do conjunto de diferenças, propõe-se utilizar um limiar de similaridade. Arquivos que apresentam similaridade acima deste limiar serão considerados versões de um mesmo documento. Arquivos que apresentem similaridade abaixo deste limiar serão considerados dois documentos distintos. No entanto, a detecção de versões não é o foco deste artigo e não será apresentada.

A maioria dos sistemas de gerenciamento utilizados no suporte à configuração de *software* consideram arquivos como objetos básicos de controle de versão. Estes sistemas

armazenam diferenças entre duas versões de um arquivo, utilizando deltas (processados através de algoritmos *diff*), e utilizam os deltas para reconstruir versões, anteriores ou posteriores, dos arquivos [12]. A proposta deste trabalho não armazena versões prévias (ou posteriores) como mudanças delta sobre o estado corrente de um certo documento. Ao invés disso, versões são armazenadas na rede como arquivos físicos separados.

Para identificar as mudanças ocorridas entre as versões, processa-se as diferenças com um algoritmo *diff* que constrói o delta representativo destas mudanças. A entrada de um algoritmo *diff* consiste de dois documentos e sua saída é um documento delta que representa as diferenças entre os dois documentos de entrada. Vários algoritmos podem ser encontrados na literatura [5],[18]. Alguns são mais eficientes em termos de tempo de execução, mas nem sempre garantem a solução ótima nas diferenças detectadas [6]. Até o presente momento, o algoritmo *XyDiff* [5] tem sido utilizado nos testes realizados. Entretanto, a arquitetura proposta permite a troca por outro algoritmo *diff*.

Considere dois arquivos XML contendo informações relacionadas a currículos, mostrados na Figura 3 (a) e Figura 3 (b). Nota-se que algumas modificações foram realizadas no segundo arquivo: o elemento *phone* foi atualizado, o elemento *fax* foi removido e o elemento *maritalSt* foi inserido. Utilizando-se o algoritmo *XyDiff*, obtém-se as seguintes operações, mostradas na Figura 3 (c). As operações 2 e 3 correspondem à atualização do elemento *phone*. A operação 1 corresponde à remoção do elemento *fax*. A operação 4 corresponde à inserção do elemento *maritalSt*. Estas operações são estruturadas em um documento XML, mostrado na Figura 3 (d).

a) Arquivo Original

```
<curriculum>
   <personalData>
       <name>Ana Silveira</name>
       <phone>5665458</phone>
       <address>Rua A</address>
       <fax>2888888</fax>
   </personalData>
</curriculum>
```

b) Arquivo Modificado

```
<curriculum>
    <personalData>
        <name>Ana Silveira</name>
        <phone>333000</phone>
        <address>Rua A</address>
        <maritalSt>single</maritalSt>
   </personalData>
<curriculum>
```

c) Diferenças detectadas com XyDiff

```
1.DeleteNode:
 <fax>2888888</fax> path
 0:0:0:3
2.DeleteNode: 5665458 path
 0:0:0:1:0 (update)
3.InsertNode: 333000 path
 0:0:0:1:0 (update)
4.InsertNode: <mari-
 talSt>single</maritalSt> path
 0:0:0:3
```

d) Arquivo XML com as diferenças detectadas

```
<delta>
   <Deleted pos="0:0:0:3"> <fax>2888888</fax>
   </Deleted>
   <Deleted update="yes"
      pos="0:0:0:1:0">5665458</Deleted>
   <Inserted update="yes"
      pos="0:0:0:1:0">333000</Inserted>
   <Inserted
   pos="0:0:0:3"><maritalSt>single</maritalSt>
   </Inserted>
</delta>
```

Fig. 3. Documento XML original (a) e modificado (b), com alterações detectadas (c) (d)

Por questões de desempenho, este artigo propõe gerar uma nova representação contendo todas as modificações do documento, e armazenar esta versão consolidada no *super peer*. Desta forma, consultas que solicitam o histórico de um documento podem ser respondidas, acessando-se um único arquivo localizado no *super peer*, eliminando a necessidade de um processamento distribuído por toda a rede. Nem todos os documentos possuem uma versão consolidada armazenada no *super peer*, mas principalmente aqueles documentos que são mais frequentemente acessados pelo sistema. O arquivo físico que contém todo o histórico de um documento XML é referenciado neste artigo como arquivo *H-Doc*. O mecanismo de agrupamento de versões de documentos em um único arquivo físico é descrito na seção a seguir.

4.1 Agrupamento de Documentos nos *Super Peers*

A partir de *n* arquivos físicos contendo a evolução do documento, gera-se uma representação única, contendo todas as modificações. Esta nova representação pode ser armazenada como um novo arquivo (*H-Doc*). Desta forma, para cada documento com *n* versões, onde *n>1,* uma nova representação é criada e armazenada no *super peer*. Rótulos temporais são responsáveis por validar e invalidar dados em versões específicas, baseados nos tempos iniciais (*TS*) e tempos finais (*TE*). *TS* e *TE* representam o período de tempo no qual uma informação é válida no mundo real.

Assume-se que cada elemento possui dois atributos temporais, *TS* e *TE*. *TS* é inicializado com o tempo de modificação do arquivo no qual o elemento foi adicionado/modificado. *TE* é inicializado com *now* e permanece com este valor enquanto for válido; um elemento é considerado válido até sua próxima modificação/remoção. *TE* é atualizado quando um elemento é modificado ou removido; neste caso, *TE* é encerrado quando uma nova instância de elemento é criada. A Figura 4 mostra a representação em árvore gerada para armazenar o histórico dos documentos apresentados na Figura 3.

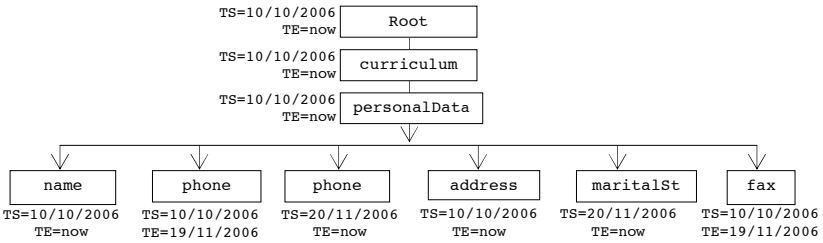

Fig. 4. Representação em árvore

A tarefa de gerar uma versão consolidada contendo todo o histórico do documento é similar ao processo de *merging* de arquivos, bastante utilizado em sistemas de versões concorrentes (CVS). No entanto, na proposta deste trabalho, não existem conflitos a serem resolvidos, uma vez que todo o histórico do documento é armazenado, com os respectivos rótulos temporais.

A representação consolidada do documento mostrado na Figura 4 pode ser armazenada fisicamente em um novo arquivo XML, mostrado na Figura 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT TS="10/10/2006" TE="now">
 <curriculum TS="10/10/2006" TE="now">
  <personalData TS="10/10/2006" TE="now"><name TS="10/10/2006" TE="now">…</name>
  <phone TS="10/10/2006" TE="19/11/2006">…</phone><phone TS="20/11/2006" TE="now">…</phone>
  <address TS="10/10/2006" TE="now">…</address><marital TS="20/11/2006" TE="now">…</marital>
  <fax TS="10/10/2006" TE="19/11/2006">…</fax>
  </personalData>
 </curriculum>
</ROOT>
```

Fig. 5. Versão consolidada – arquivo *H-Doc*

Representações consolidadas são armazenadas no *super peer* e podem ser utilizadas para um processamento mais rápido de certas consultas que solicitam histórico de documentos. Dessa maneira, não há a necessidade de acessar todas as versões de um documento espalhadas pelos *peers*. Obviamente, os metadados do ambiente *DetVX* devem prever a representação destas versões consolidadas, de forma que o processador de consultas possa tirar vantagem dos arquivos *H-Doc*, quando estes existirem. O detalhamento dos metadados não é descrito neste artigo.

4.2 Consulta à Versões de Documentos

Sistemas que fornecem suporte a documentos versionados devem disponibilizar mecanismos para recuperação destas informações. No ambiente *DetVX*, antes de rotear uma consulta a um determinado *peer*, o sistema verifica se a consulta pode ser respondida localmente. Em caso afirmativo, a consulta é processada e os resultados são retornados ao usuário; caso contrário, a consulta é roteada para o *peer* adequado. Para verificar se uma consulta pode ser respondida localmente (pelo *peer* solicitante), o sistema verifica os metadados de seu *super peer*. Por exemplo, considere uma consulta que solicita a primeira versão de um arquivo *F*; através dos metadados, verifica-se em qual *peer* esta informação está armazenada (analisando-se o número da versão solicitada).

Com o agrupamento de versões em um único arquivo físico (*H-Doc*), determinadas consultas podem ser diretamente respondidas pelo *super peer*, como: obter o endereço do autor antes de *10/10/2006*; obter a última versão do endereço dos autores; obter o histórico de endereço dos autores, etc. Caso o *peer* solicitante não tenha condições de responder a consulta, a solicitação é enviada ao *super peer*, o qual se responsabiliza por respondê-la (caso possua o histórico daquele documento armazenado localmente) ou roteá-la para o *peer* que possui a informação solicitada. Desta forma, evita-se que a rede fique sobrecarregada com requisições que somente determinado *peer* (ou *super peer*) está apto a responder. Nesta seção será considerado apenas o processo de consulta em arquivos *H-Doc*. Questões prévias de análise de metadados e roteamento de consultas não são abordadas.

Considerando um elemento qualquer *e* de um documento *D*, os filtros temporais aplicados podem ser baseados em uma data *x* ou em um intervalo de tempo *x* e *y*. Algumas cláusulas utilizadas para a execução de consultas temporais são mostradas a seguir:

- *select_Before (e, x)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* antes da data *x*. Em outras palavras, procura-se por elementos cujo tempo inicial de validade (*TS*) seja menor à data *x* especificada;
- *select_After (e, x)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* após a data *x*. Em outras palavras, procura-se por elementos cujo tempo final de validade (*TE*) seja maior à data *x* especificada;
- *select_Between (e, x, y)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* entre o perído *x* e *y*, onde $x<y$. Em outras palavras, procura-se por elementos cujo tempo inicial de validade (*TS*) seja menor ou igual a *y* e cujo tempo final de validade (*TE*) seja maior ou igual a *x*;
- *select_Now (e)*: esta cláusula retorna o conteúdo dos elementos *e* que são válidos no documento *H-Doc* no momento atual. Em outras palavras, procura-se por elementos cujo tempo final de validade (*TE*) seja igual a *now*;
- *select_Before (D, x)*: esta cláusula retorna o conteúdo do documento *D* que é válido antes da data *x*. Em outras palavras, procura-se por documentos cujo tempo inicial de validade (*TS*) do elemento raiz seja menor à data *x* especificada;
- *select_After (D, x)*: esta cláusula retorna o conteúdo do documento *D* que é válido após a data *x*. Em outras palavras, procura-se por documentos cujo tempo final de validade (*TE*) do elemento raiz seja maior à data *x* especificada;
- *select_Between (D, x, y)*: esta cláusula retorna o conteúdo do documento *D* que é válido entre o perído *x* e *y*. Em outras palavras, procura-se por documentos cujo tempo inicial de validade (*TS*) do elemento raiz seja menor ou igual a *y* e cujo tempo final de validade (*TE*) do elemento raiz seja maior ou igual a *x*;

- *select_Now (E)*: esta cláusula retorna o conteúdo do documento *D* que é válido no no momento atual. Em outras palavras, procura-se por documentos cujo tempo final de validade (*TE*) do elemento raiz seja igual a *now*.
  Alguns exemplos de consultas temporais implementadas são mostrados na seção 5.2.

5. Implementação

A implementação da ferramenta *XVersion* disponibiliza um ambiente que agrupa em um único arquivo de saída os *n* arquivos XML de entrada, através do processamento de diferenças. O agrupamento destes arquivos baseia-se no uso de algoritmos *diff* e rótulos temporais que especificam a validade de cada elemento dentro do documento de saída gerado. Com a geração da representação única do histórico dos documentos, o usuário pode submeter consultas temporais, utilizando *XQuery* [17].

5.1 Agrupamento dos Documentos

Dentre as funcionalidades da *XVersion*, a primeira a ser utilizada pelo usuário é o comparador de arquivos XML. Para esta comparação, foi utilizada a implementação em Java do algoritmo *XyDiff*. A partir de *n* documentos XML fornecidos como entrada, executa-se o *XyDiff* para cada par de arquivos a_i e a_{i+1} (onde *i* é o índice do documento XML dentro da lista de documentos fornecidos), obtendo assim *n-1* documentos representando as diferenças entre cada par de arquivos fornecido. A Figura 6 mostra os arquivos delta gerado para três versões de documentos XML fornecidas, utilizando *XyDiff*. Considere que as três versões possuem as seguintes datas de modificação, respectivamente: *09/09/2006*, *05/11/2006* e *11/11/2006*.

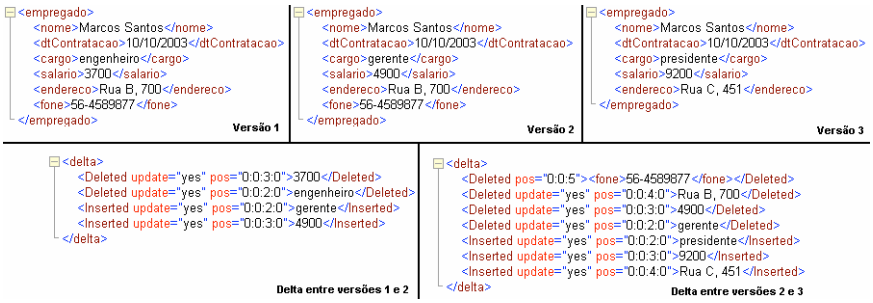

Fig. 6. Arquivos delta gerados para três versões de documentos XML

A seguir, é criada a representação do arquivo consolidado para a representação do histórico das versões. Para isto, é criado um arquivo XML representando a união de cada par de versões fornecidas. Esta união é feita a partir dos dois arquivos fornecidos e do arquivo delta contendo as diferenças entre estes. O último passo é juntar todas as uniões geradas em um único arquivo. A Figura 7 mostra o documento *H-Doc* resultante, contendo o histórico das versões dos arquivos mostrados na Figura 6.

Embora a geração do documento *H-Doc* seja simples de se entender, o desenvolvimento do algoritmo que trata da união de mais de dois arquivos foi complexo. Um nodo pode

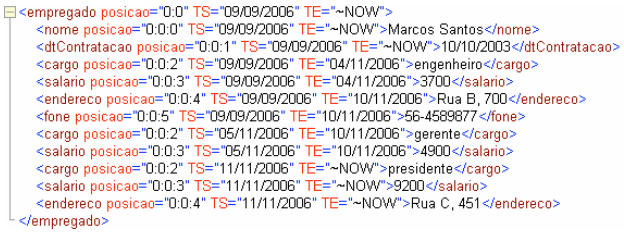

Fig. 7. Históricos das versões - documento *H-Doc* resultante

sofrer várias alterações ao longo das versões, o que implica em aparecer várias vezes no documento consolidado gerado. Neste caso, torna-se necessário identificar qual das versões deste nodo é a válida. Adotou-se neste trabalho uma abordagem baseada na equivalência de nodos por posição, na qual metadados são inseridos nos arquivos XML antes de executar qualquer operação sobre eles. Estes metadados indicam a posição do elemento dentro do arquivo XML (atributo *posicao*) e a partir deles é possível descobrir a equivalência de nodos entre versões. A estrutura de geração dos valores do atributo *posicao* segue a identificação de nodos utilizada pelo algoritmo *XyDiff*.

5.2 Consulta ao Histórico de Documentos

A última funcionalidade implementada na *XVersion* é a possibilidade de se executar consultas sobre o arquivo *H-Doc*. Desta forma, uma única consulta é expressa sobre um único arquivo físico. Através de filtros temporais, informações históricas podem ser retornadas sem a necessidade de processar a consulta em toda a rede. O trabalho apresentado em [16] mostra uma proposta similar, e usa funções definidas pelo usuário para expressar consultas temporais em *XQuery* à versões de documentos XML.

Exemplos de consultas temporais incluem: obter informações válidas *antes* de uma certa data/período de tempo, obter informações válidas *após* uma certa data/período de tempo, obter informações válidas *entre* um certo período de tempo, obter informações válidas no momento atual, obter todo o histórico de uma certa informação, entre outras. Algumas consultas executadas na ferramenta são mostradas na Figura 8.

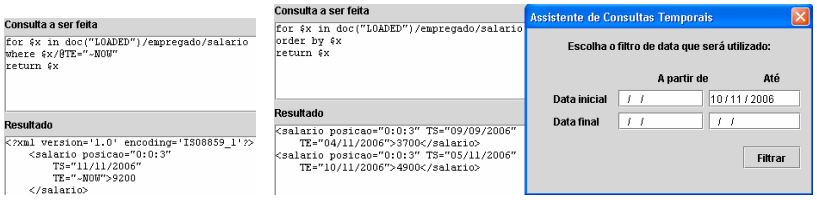

Fig. 8. Exemplos de consultas temporais

A primeira consulta retorna o conteúdo do elemento *salário* no momento atual. Para responder a esta consulta, a ferramenta busca todos os elementos cujo tempo de validade final seja *now*. A segunda consulta retorna os valores de salário válidos antes do dia *10/11/2006*. Para responder a esta consulta, a ferramenta busca todos os elementos *salário* cujo tempo inicial de validade seja inferior à data especificada. Para a execução das consultas foi utilizada a biblioteca *Qizx/Open*[5], distribuída sob a licença GNU.

[5] http://www.xfra.net/qizxopen/

6. Conclusões

Este artigo apresentou um mecanismo de agrupamento e consulta à versões de documentos XML, inserido na arquitetura do *DetVX*, um ambiente para detecção de réplicas e versões XML em um contexto P2P. Gerenciamento de versões tem sido bastante estudado, mas a característica distribuída dos ambientes P2P torna o problema mais complexo. Este artigo apresentou uma abordagem simples de representação de versões em um único arquivo físico, gerado pelo processamento de diferenças entre versões. Consultas podem ser expressas através da restrição de valores nos rótulos temporais. Desta forma, elimina-se a necessidade de processar consultas distribuídas na rede P2P, uma vez que a representação consolidada com o histórico das versões é armazenada no *super peer*. Com a abordagem proposta, espera-se melhorar o desempenho das consultas em documentos frequentemente acessados. No entanto, este artigo não apresentou resultados comparativos de desempenho, os quais serão realizados futuramente, à medida que o ambiente no qual a ferramenta *XVersion* está inserido seja finalizado.

Atualmente o foco do ambiente em desenvolvimento é trabalhar no problema de *ranking* de páginas Web em máquinas de busca. Um dos critérios utilizados em *ranking* de páginas é o número de *links* que apontam para uma determinada página *p*. No entanto, uma nova versão de uma página *p* pode ter um pequeno número de apontadores, principalmente porque as páginas que apontam para *p* ainda não estão cientes da nova versão disponível. Neste contexto, detecção de réplicas e versões pode ser útil para melhorar o *ranking* de novas versões de páginas mesmo que o número de apontadores ainda seja pequeno.

O ambiente proposto neste artigo baseia-se na arquitetura *super peer*, mas faz uso de uma entidade central para o gerenciamento de alguns metadados. No entanto, o ambiente pode ser utilizado em redes distribuídas (utilizando DHT – *distributed hash tables*), com algumas adaptações. Como trabalho futuro, pretende-se ampliar o conjunto de funções temporais fornecido pela *XVersion* e incorporar a ferramenta ao ambiente *DetVX* em desenvolvimento.

Referências

1. Aberer, K. and Hauswirth, M.. An Overview on Peer-to-Peer Information Systems. Workshop on Distributed Data and Structures (WDAS), Paris, France, 2002.
2. Chawathe, S. e Widom, J.. Representing and Querying Changes in Semistructured Data. Proc. of 14th Intl. Conference on Data Engineering, 4-13, 1998.
3. Chien, S.; Tsotras, V.J. e Zaniolo, C.. Version Management of XML Documents. Lecture Notes in Computer Science, 2001.
4. Chien, S.Y.; Tsotras, V.J. e Zaniolo, C.. XML Document Versioning. SIGMOD Rec., ACM Press, 30, 46-53, 2001.
5. Cobena, G., Abiteboul, S. e Marian, A.. Detecting Changes in XML Documents. , Proc. of 18th Intl. Conference on Data Engineering, 41-52, 2002.
6. Cobena, G.; Abdessalem, T. e Hinnach, Y.. A Comparative Study for XML Change Detection. Verso report number 221, INRIA, 2002.
7. Conradi, R. e Westfechtel, B.. Version Models for Software Configuration Management. ACM Comput. Surv., 30(2):232–282, 1998.
8. Grandi, F., Mandreolo, F., Tiberio, P. E Bergonzini. P.. A Temporal Data Model and Management System for Normative Texts in XML Format. Proc. of the 5th ACM Intl. Workshop on Web Information and Data Management, (WIDM), ACM Press, 29-36, 2003.

9. Jensen, C. S., Dyreson, C. E. et al.. The Consensus Glossary of Temporal Database Concepts. In: Temporal Databases - Research and Practice. Springer-Verlag, 1998.
10. Katz, R. e Chang, E.. Managing Change in a Computer-Aided Design Database. Proc. of VLDB Conference, 1987.
11. Ozsoyoglu, G. e Snodgrass, R.. Temporal and Real-Time Databases: A Survey. IEEE Transactions on Knowledge and Data Engineering, 7, 513-532, 1995.
12. Ronnau, S.; Scheffczyk, J. e Borghoff, U.M.. Towards XML Version Control of Office Documents. DocEng '05: Proc. of the 2005 ACM symposium on Document engineering, ACM Press, 10-19, 2005
13. Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A. e Snodgrass, R. T.. Temporal Databases: Theory, Design and Implementation. Redwood City: Benjamin/Cummings Publishing Company, Inc., 1993
14. Vagena, Z.; Moro, M. e Tsotras, V.. Supporting Branched Versions on XML Documents. Proc. of 14th Intl. Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 137-144, 2004.
15. Westfechtel, B., Munch, B. P., e Conradi, R. A Layered Architecture for Uniform Version Management. IEEE Trans. Software Eng., 27(12):1111–1133, 2001.
16. Wang, F.; Zaniolo, C.; Zhou, X.; Moon, H, J.. Version Management and Historical Queries in Digital Libraries. Proc. of the 12th Intl. Symposium on Temporal Representation and Reasoning, 207-209, 2005.
17. XQuery. XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 8 June 2006. Disponível em http://www.w3.org/TR/xquery/
18. Wang, Y., DeWitt, D. J., Cai, J.. X-Diff: An Effective Change Detection Algorithm for XML Documents. Intl. Conference on Data Engineering, 519-530, 2003.
19. Schollmeier, R.. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architetures and Applications. Proc. of the 1st Intl. Conf. on Peer-to-Peer Computing, 27-29, Linköping, Sweden. IEEE Computer Society 2001, ISBN 0-7695-1503-7.
20. Su, H., Kramer, D., Chen, L., Claypool, K. T., Rundensteinrer, E. A.. XEM: Managing the Evolution of XML Documents. Proc. of 11th Intl. Workshop on Research Issues in Data Engineering, Heidelberg, 2001.
21. Gao, D. and Snodgrass, R.T.. Temporal Slicing in the Evaluation of XML Queries. Proc. of Very Large Database Systems, 2004.
22. Wang, F. and Zaniolo, C.. Representing and Querying the Evolution of Databases and their Schemas in XML. In Workshop on Web Engineering, SEKE, San Francisco, USA, 2003.
23. Chawathe, S.S.; Abiteboul, S. and Widom, J.. Managing Historical Semistructured Data. Theory and practice of object systems, 2000.
24. Grandi, F. e Mandreoli, F.. The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. Proc. of Advances in Information Systems, (ADVIS 2000), 2000.
25. Gergatsoulis, M. e Stavrakas, Y.. Representing Changes in XML Documents using Dimensions. First XML Database Symposium, (XSym), Berlin, Germany, 208-222, 2003.
26. Amagasa, T.; Yoshikawa, M. e Uemura, S.. A Data Model for Temporal XML Documents. Proceedings of the 11th Intl. Conf. on Database and Expert Systems Applications, (DEXA), London, England , 2000.
27. Amagasa, T.; Yoshikawat, M. e Uemura, S.. Realizing Temporal XML Repositories using Temporal Relational Databases. Proc. of the 3rd Intl. Symposium on Cooperative Database Systems for Advanced Applications, (CODAS), 60-64, 2001.
28. Dyreson, C.. Observing Transaction-Time Semantics with /sub TT/XPath. Proc. of the 2nd Intl. Conf. on Web Information Systems Engineering , 1, 193-202, vol.1, 2001.
29. Chien, S.; Tsotras, V.; Zaniolo, C. and Zhang, D.. Storing and Querying Multiversion XML Documents using Durable Node Numbers. Proc. of the 2nd Intl. Conf. on Web Information Systems Engineering, 1, 232-241, vol.1, 2001.
30. Vagena, Z. e Tsotras, V.. Path-Expression Queries over Multiversion XML Documents. Proc. of Intl. Workshop on the Web and Databases (WebDB), 49-54, 2003.
31. Grandi, F., Mandreoli, F., Tiberio, P.. Temporal Modeling and Management of Normative Documents in XML Format. Data & Knowledge Engineering, v. 54, n. 3, p. 327-354, September, 2005.
32. Wang, F. e Zaniolo, C.. Temporal Queries in XML Document Archives and Web Warehouses. Proc. of the 10th Intl. Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic, 47-55, 2003.
33. Wang, F.; Zaniolo, C.; Zhou, X. e Moon, H.J.. Managing Multiversion Documents and Historical Databases: a Unified Solution Based on XML. Proc. of the 8th Intl. Workshop on the Web, (WebDB), Baltimore, Maryland, USA, 151-153, 2005.

Utilização de XML numa plataforma de Data Mining distribuído

Ruy Ramos, Carlos Adriano Gonçalves and Rui Camacho

LIACC, Rua de Ceuta 118 - 6º 4050-190 Porto, Portugal
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal

{ruyramos,rcamacho}@fe.up.pt, cadriano.goncalves@gmail.com

Resumo O processo de Extracção de Conhecimento em Bases de Dados (*Knowledge Discovery in Databases* - KDD) envolve a análise de extensas bases de dados e recurso a complexos algoritmos de análise de dados (*Data Mining*). Este processo requer, geralmente, recursos computacionais dedicados e de elevado custo o que reduz significativamente o número de utilizadores capazes de efectuar tais análises. Neste artigo apresentamos uma arquitectura baseada em computadores pessoais distribuídos numa rede de computadores de uma organização e que permite a realização de tarefas de KDD sem recursos computacionais dedicados e sem perturbar o funcionamento da organização. A arquitectura é denominada *Harvard* - ***HARV****esting* ***A****rchitecture of idle machines fo****R*** *****D****ata mining*. O Harvard utiliza uma linguagem de especificação e controlo de tarefas baseada em XML. A linguagem XML no caso do Harvard é imprescindível para a interoperabilidade entre os diferentes componentes do ambiente descrevendo claramente todos os aspectos da tarefa de *KDD* a ser executada de forma distribuída. Os resultados alcançados por diferentes nós do sistema são transcritos em XML, de modo a facilitar a apresentação ao utilizador do ambiente *Harvard* e ainda permitir integração com outros sistemas de extracção de conhecimento.

1 Introdução

O crescente uso das tecnologias da informação associado ao crescimento e desenvolvimento económico de vários sectores (empresarial, governo, comunidade científica e académica, entre outros) têm permitido que as organizações e a sociedade em geral reúnam uma enorme quantidade de dados e informações sistematicamente em Bases de Dados (BD) [4].

Do ponto de vista humano, torna-se praticamente impossível interpretar, analisar e obter resultados a partir de grandes quantidades de dados sem o auxílio de computadores. Torna-se muito difícil diferenciar informações verdadeiramente importantes e úteis em tão grande quantidade de dados [4,8]. Neste caso, a aplicação de técnicas automáticas de análise capazes de "extraír" informação útil torna-se inevitável dada a complexidade e a extensão das BD.

A aplicação de técnicas de *Data Mining* para auxiliar a análise de grandes quantidades de dados torne-se uma alternativa viável e aplicável, usando para isso os algoritmos de *Machine Learning* como *Inductive Logic Programming* (ILP) [12] e *Association Rules Discovery* (ARD) [1]. Uma desvantagem destes algoritmos é que são computacionalmente muito exigentes e este aspecto tem limitado o seu uso em aplicações de extracção de conhecimentos no mundo real.

Além disso, os algoritmos de *Machine Learning* foram originalmente construídos para abordagens sequenciais monolíticas pressupondo sempre um ambiente centralizado. É necessário que os dados estejam todos num único local para que as técnicas de *Data Mining* possam ser aplicadas. A quantidade de informação actualmente disponível é tão grande que centralizá-la se torna oneroso e extremamente complexo. Há casos mesmo em que os dados estão dispersos e não podem ser centralizados devido a limitações de rede, falta de espaço de armazenamento, questões de segurança, confidencialidade ou privacidade. Assim, há necessidade de adoptar novas abordagens para análise de dados nos seus respectivos locais de armazenamento.

Desenvolver técnicas de *Data Mining* para arquitecturas distribuídas tem sido um grande desafio da comunidade científica de *Knowledge Discovery in Databases - KDD*. O desenvolvimento de um novo estágio do KDD denominado de DPKDD (*Distributed and Parallel Knowledge Discovery in Databases*) está ligado principalmente pelo avanço em áreas como *storage, access e analysis* [10] e ao uso de tecnologias como *Grid Computing* [6].

A nossa proposta contempla uma arquitectura computacional dedicada ao processo de *Data Mining Distribuído* denominada de *Harvard* - ***HARV****esting* ***A****rchitecture of idle machines fo****R*** ***D****ata mining,* com os seguintes objectivos: fornecer ao analista de dados (utilizador) uma linguagem simples mas poderosa para especificar as tarefas de análise de dados (*KDD*); viabilizar a utilização optimizada de computadores pessoais quando ociosos em prol de um processamento cooperativo e útil; construir uma plataforma que possa ser usada independentemente dos sistemas operativos instalados (*Linux* ou *Windows*) na organização; permitir análise de dados de forma distribuída (computação distribuída e acesso a dados fisicamente distribuídos); permitir o uso de diferentes algoritmos de *Machine Learning* e suas implementações (ferramentas) sem qualquer alteração da plataforma, sendo portanto independente da ferramenta usada pelo analista; e permitir recuperação de fallhas do próprio sistema (tolerância a falhas).

Além dos objectivos apresentados, o *Harvard* apresenta as seguintes vantagens: o utilizador pode facilmente definir o processo de análise de dados mais adequado para suas necessidades; com este ambiente pode-se expandir o uso da análise de dados a um vasto leque de organizações já que os custos envolvidos são bastante baixos, as máquinas usadas são as mesmas que servem para as tarefas rotineiras. E além disso, o processamento computacional *Harvard* não perturba o trabalho normal da organização uma vez que só usa recursos computacionais desocupados ou ociosos.

O *Harvard* permite ao utilizador descrever cada tarefa do processo de KDD através da linguagem de anotação XML (*Extensible Markup Language*)[3] e especificar o fluxo de sua execução (*workflow*) numa linguagem descritiva de sintaxe simples mas no entanto poderosa. O sistema corre num *Servidor* com uma colecção ilimitada de estações *Cliente*. Tanto o *Servidor* como os *Clientes* são programados em *Java*. Os *Clientes* podem aceder aos dados directamente de uma base de dados (usando JDBC) e todo *software* necessário de análise de dados, via protocolo HTTP.

Em termos de trabalhos correlatos destacamos o Condor [11] da Universidade de Wisconsin-Madison e o BOINC (*Berkeley Open Infrastructure for Networking Computing*) [2]. O ambiente Condor caracteriza-se por gerir um ambiente composto de vários servidores e/ou *cluster*, como o "Beowulf". É basicamente um ambiente *Grid Computing* para gerir a capacidade de processamento de recursos computacionais de um "campus" ou de uma organização. Tem como características principais a gestão de *jobs*, definição de políticas de uso do ambiente distribuído e gestão e monitoramento dos recursos computacionais. Há uma variante denominada Condor-G [7] que facilita a integração com ambientes *Grid Computing* baseados no *Globus Toolkit* [5]. Por outro lado, o BOINC é uma arquitectura computacional distribuída que utiliza recursos computacionais registados voluntariamente a partir da Internet. É uma plataforma que permite aos cientistas gerir recursos computacionais públicos disponíveis na Internet em benefício de projectos científicos de grande dimensão. Alguns projectos como o *SETI@home*, *Folding@home* utilizam a plataforma BOINC. A arquitectura é composta por servidores de projectos que operam suas próprias aplicações e bases de dados, e partilham processamento voluntário disponível a partir de estações ligadas à Internet que "doam"ciclos de processamento por intermédio de aplicações instaladas para esta finalidade.

O artigo está organizado da seguinte forma: na Secção 2 descrevemos como as tarefas do processo de KDD podem ser especificadas por meio de uma linguagem simples mas poderosa usando como base XML; na Secção 3 apresentamos a arquitectura do *Harvard* e seu funcionamento; e por fim apresentamos as conclusões e trabalhos futuros na última Secção.

2 O processo de Extracção de conhecimento

Segundo Fayyad [4], *Data Mining* é o núcleo principal de um processo mais amplo denominado de *Knowledge Discovey in DataBases* (KDD), ou Descoberta de Conhecimento em Bases de Dados, conforme apresentado na Figura 1.

Knowledge Discovey in DataBase (KDD) é um processo não trivial de identificar, validar e reconhecer padrões de dados que possam prover informação válida gerando conhecimento sobre uma determinada Base de Dados. Os dados referem-se a representação de factos e os padrões são definidos por uma expressão que descreve um subconjunto desses mesmos dados [4].

De acordo com *Fayyad* [4] o processo de KDD é composto pelas fases:

- preparação dos dados;

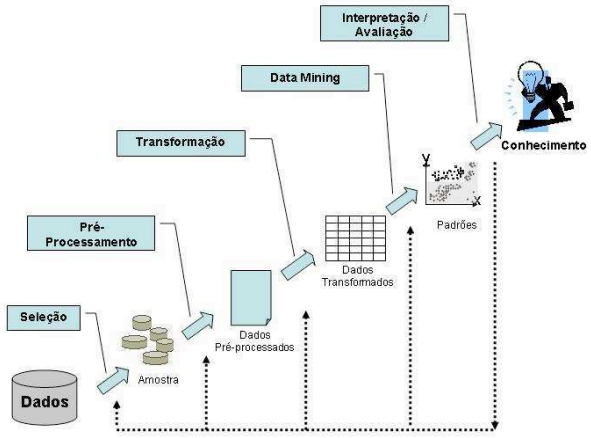

Figura 1. Processo de Extracção do Conhecimento

- selecção de dados;
- pré-processamento de dados;
- transformação de dados;
- *data mining*;
- interpretação e avaliação do conhecimento.

2.1 A fluxo do processo de análise de dados

O *Harvard* inicia o processo de análise de dados lendo, de um ficheiro, a especificação do fluxo do processo (*workflow*) juntamente com a descrição de cada uma das tarefas do processo KDD. O fluxo do processo é representado por um grafo com dois tipos de nós: sequenciais e paralelos. Cada nó regista o conjunto de tarefas a serem executadas, representadas e designadas como UT (Unidades de Trabalho). Na Secção 3, que descreve a arquitectura do *Harvard*, as UT são descritas com maior detalhe.

No caso do nó sequencial, as tarefas deverão necessariamente ser executadas em ordem de precedência devido às suas interdependências no processo de análise de dados. Por outro lado, os nós paralelos especificam tarefas que podem ser executadas em simultâneo.

Na Figura 2, apresentamos um exemplo de descrição de um processo KDD. A figura mostra ainda em detalhe um conjunto de tarefas definidas $T1$ ... $T15$ que abrangem todo o processo, desde o pré-processamento, seleccionando os atributos mais relevantes, até à consolidação dos resultados ($T15$). A descrição de cada tarefa Tn está codificada em XML em ficheiros distintos que serão processados pelo módulo Gestor de Tarefas (GT) do *Harvard*, conforme detalhado na Secção 3.

```
# This is the Tasks control description
# using the Task Control Language (.tcl)

seq
  T1  # make a 70%/30% train/test set

  par # execute the tasks in parallel
    seq
     T2  # get dataset without Att1
     par
       T3  # eval dataset without Att1 using m = 10
       T4  # eval dataset without Att1 using m = 50
     endpar
    endseq

    seq
     T5  # get dataset without Att5
     par
       T6  # eval dataset without Att2 using m = 10
       T7  # eval dataset without Att2 using m = 50
     endpar
    endseq

  barrier T[3-4], T[6-7] # wait for all tasks to finish

  T8  # choose the best set of attributes
  T9  # make the 5-fold CV blocks

  par
    T[10-14] # do each CV i

  barrier T[10-14]  # wait for all CV folds

 T15 # run with all data to produce the final theory

endseq
```

Figura 2. Descrição do processo KDD em tarefas sequencias e paralelas.

2.2 A especificação das tarefas

A linguagem de especificação das tarefas, conforme ilustrado na Figura 3, parte inicialmente da sintaxe básica da XML. A partir deste contexto são definidas variáveis com seus respectivos parâmetros que posteriormente serão interpretadas pelo *Havard* no seu módulo que trata da especificação da tarefa - o Gestor de Tarefas (GT).

Na especificação de cada tarefa o utilizador (analista de dados) deverá fornecer em detalhe toda a informação relevante. Isso inclui descrever entre outros ítens:

- local e nome do ficheiro que contém os dados para análise;
- algoritmo(s) a ser(em) usado(s) com respectivos parâmetros de análise;
- nome da aplicação que implementa o algoritmo escolhido (*e.g.*: C4.5)
- ferramentas de pré-processamento a ser usada (*e.g.*: aplicação de *script perl* para eliminar ou consolidar atributos);
- local e formato de representação dos resultados.

2.3 O uso da linguagem XML no *Harvard*

A linguagem XML usada no *Harvard* permite que o utilizador possa especificar claramente o processo de KDD pois "provê um conceito para descrever, armazenar, permutar e manipular dados estruturados" [9,3].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tasks SYSTEM ''tasks.dtd'' >
#  Run C4.5 on a data set stored in a Data Base and return the
#  in a "results table" of the same DB
#####      Task Description Language (.tdl) file         #####
<workunit>
 <id> T1 </id>
                    ### data ###
   <fetch-data>
      <method> jdbc     </method>
      <server> dbserver </server>
      <user>   dmuser   </user>
      <db>     kdd99    </db>
      <password> ______</password>
      <db-access>
       <source-data>
          <query> select * FROM data LIMIT 5000 OFFSET 100 </query>
          <file>  kdd99.data </file>
      </source-data>
      ...
     </db-access>
   </fetch-data>
                  ### source code ###
  <fetch-code>
   <source-code>  # C4.5 code to construct the Decision Tree
     <getMethod> http </getMethod>
     <url> http://www.fe.up.pt/~rcamacho/c4.5 </url>
   </source-code>
   ...
  </fetch-code>
                ### sub-tasks execution ###
 <execution>
    <results-storage>
      <method> jdbc </method>
      <server> dbserver </server>
      <user>   dmuser   </user>
      <db>     kdd99    </db>
      <password> ______ </password>
    </results-storage>
### sub-task 1 ###
 <subtask>
   <exec-mode>   noninteractive           </exec-mode>
   <exec-command> c4.5 -f kdd99 -m 100 -u </exec-command>
   <results-file> c45.output              </results-file>
   <exec-time>  30 </exec-time>
 </subtask>
### sub-task 2 ###
   ...
 </execution>
                ### required resources ###
   <resources>
     <hd> 10 </hd>
     <ram> 0.5 </ram>
     <opsystem> linux </opsystem>
   </resources>
</workunit>
```

Figura 3. Descrição em detalhe de uma tarefa.

Decorre que a linguagem XML se caracteriza por permitir estruturar a informação de forma hierárquica, facilitar a edição devido à sintaxe simples (qualquer editor de texto pode ser usado), e permitir a fácil compreensão dos dados estruturados, já que não requer nenhuma ferramenta sofisticada para visualização. No caso do *Harvard* os dados estruturados em XML permitem a fácil interpretação pelos diferentes módulos da plataforma, e também a geração dos resultados para posterior integração, seja com outras ferramentas de análise de dados, ou para o armazenamento em bases de dados.

Considerando que o *Harvard* é desenvolvido em *Java*, o processamento de ficheiros em XML é facilitado devido as bibliotecas (*classes*) disponíves. E considerando as características da linguagem XML, alterações em ficheiros XML não significam necessariamente alterações em código de programação Java. Pode-se, por exemplo, acrescentar novas *tags* como requisito de um novo módulo do *Harvard*, ou simplesmente para melhor apresentar os resultados.

O utilizador pode através de um simples editor ou de qualquer outra ferramenta de edição/visualização para XML especificar o processo de análise de dados bastando para isso carregar um *template* de tarefa do *Harvard* que define todos os atributos passíveis de especificação com seus respectivos parâmetros. Uma vez escrito o ficheiro de especificação de tarefas, este é inserido como parâmetro de activação do *Harvard.*

Cabe destacar que os resultados do *Harvard*, por serem representados em XML, não necessitam de tratamento especial para apresentação ao utilizador, sendo visualizados em qualquer navegador compatível.

3 O ambiente Harvard

A infra-estrutura básica do *Harvard* é composta por dois tipos de *nós:* um nó *Servidor* e vários nós *Clientes*. Para além destes "componentes principais", a arquitectura pode aceder a um servidor *Web* para obter programas de análise de dados e um sistema gestor de BD para obter dados, armazenar resultados e efectuar um *back-up* actualizado de informação dos nós *Clientes*. Uma visão global da arquitectura da infra-estrutura básica pode ser vista na Figura 4.

A arquitectura inclui as seguintes características:

- A plataforma é baseada numa arquitectura *Servidor/Cliente* (mestre/escravo)
- Os nós *Clientes* estão sujeitos a uma política de utilização que permite a sua activação/desactivação sempre que essa política o determinar
- A linguagem de programação da infra-estrutura básica é o *Java*
- A arquitectura é modular
- A infra-estrutura básica é independente do(s) algortimo(s) de análise de dados
- O *cliente* pode ser executado tanto em sistemas operativos *Linux* como *Windows*
- Em caso de falha de um qualquer nó a tarefa em execução nesse nó é reinicializada num outro nó (tolerância à falhas dos clientes)

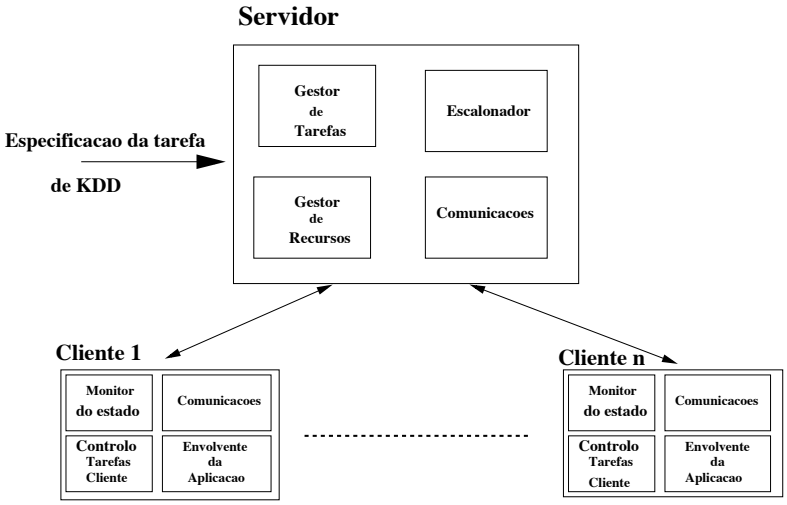

Figura 4. Arquitectura *Harvard*

- Em caso de falha do Servidor um dos clientes (previamente designado) assume o papel de servidor (tolerância a falhas do Servidor)
- Permite acesso directo a Bases de Dados e a repositórios Web para transferência de dados ou de programas

3.1 Servidor

O nó *Servidor* quando inicia "lê" informações estáticas sobre todos os recursos computacionais disponíveis. A parte dinâmica da informação dos recursos computacionais é actualizada regularmente durante a execução. O nó servidor "lê", ainda, o conjunto de tarefas a realizar, descritas como um conjunto de Unidades de Trabalho (UT). Para cada UT é seleccionada "a melhor" máquina onde vai ser executada. O servidor atribui essa UT à máquina indicada. Quando uma UT termina associa o resultado recebido à UT e devolve-o ao utilizador.

O *Servidor* é constituído por quatro módulos:

- o gestor de tarefas (GT),
- o escalonador (ESC),
- o gestor de recursos (GR) e
- o módulo de comunicação (COM).

Módulo Gestor de Tarefas (GT): Este módulo GT basicamente controla todo o processo KDD representado através de um grafo de *workflow* com todas as tarefas pertinentes ao processo de análise de dados. As tarefas por sua vez são convertidas em uma ou mais UT (unidades de trabalho) que serão processadas

pelo ambiente de forma independente. À medida que cada uma das tarefas do processo termina, os resultados são associados e armazenados no respectivo nó do grafo e um *back-up* feito num BD residente numa máquina diferente do *Servidor*. Isso permite o acompanhamento do processo de análise de dados passo-a-passo pelo utilizador e facilita a consolidação final dos resultados para apresentação. O módulo GT interage com o módulo ESC provendo todas as especificações necessárias para a distribuição e execução das tarefas.

Módulo Escalonador (ESC): O módulo ESC recebe do módulo GT as tarefas para serem executadas pelos clientes. As tarefas são especificadas em uma ou mais unidades de trabalho (UT) de acordo com o formato mostrado na Figura 5. Para cada UT, o escalonador solicita ao módulo de gestão de recursos (GR) uma máquina adequada à execução da tarefa. O módulo GR devolve a especificação de uma máquina disponível e adequada para a tarefa ou a indicação de que não há máquinas disponíveis para essa tarefa. O escalonador compõe a mensagem de requisição da tarefa e entrega-a ao módulo de comunicação para ser enviada. O escalonador recebe ainda mensagens que indicam a conclusão de tarefas. Deve recolher os resultados e devolvê-los ao utilizador.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<workunit>
  <identification> T1 </identification>
  <apllication>
    <urlapl>www.fe.up.pt/ilp/IndLog/indlog.tgz</urlapl>
    <script>www.fe.up.pt/ilp/IndLog/script-ilp.scp</script>
    <parameters>www.fe.up.pt/ilp/IndLog/parameters.txt
    </parameters>
  </apllication>
  <data>
    <dataset>kdd99</dataset>
    <DBserver>www.fe.up.pt/mysql</DBserver>
    <DB>kdd99</DB>
    <translationscript>toilp.scp</translationscript>
  </data>
  <requirements>
    <memory unit="Mbyte">1000</memory>
    <processor>Pentium</processor>
    <harddisc unit="Mbyte">1000</harddisc>
  </requirements>
  <estimatedtime unit="s"> 30 </estimatedtime>
  <results>
   <filename>kdd99.out.gz</filename>
   <DBserver>www.fe.up.pt/mysql</DBserver>
   <DB>kdd99</DB>
  </results>
</workunit>
```

Figura 5. Exemplo de especificação de uma tarefa em UT usando a linguagem XML.

Módulo Gestor de Recursos (GR): Este módulo mantém informação sobre as máquinas de acordo com a especificação indicada na Figura 6, e que inclui: informação estática e informação dinâmica que é regularmente actualizada. Entre a informação de cada máquina está o estado dela: indisponível, disponível ou em utilização. O conjunto de máquinas é mantido em 3 filas (uma para cada estado possível). Este módulo recebe ainda pedidos do módulo escalonador no formato

de um subconjunto da especificação de uma máquina e devolve uma máquina adequada ao pedido ou a indicação de que não há máquinas disponíveis para esse pedido. O módulo GR recebe periodicamente mensagens de cada cliente indicando a carga de trabalho no cliente e quantos utilizadores estão a usar a máquina.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<machine>
 <identifier>
   <ip>0.0.0.0</ip>
   <hostname>tau4</hostname>
 </identifier>
 <hardware>
   <cpu>Intel</cpu>
   <clockspeed unit="Gh">1.6</clockspeed>
   <ram unit="Mbyte">50</ram>
   <hd unit="Mbyte">250</hd>
 </hardware>
 <opsystem>Linux</opsystem>
<availability>
  <static> </static>
  <workload> </workload>
  <login> </login>
</availability>
</machine>
```

Figura 6. Descrição dos recursos disponíveis.

Módulo de Comunicações (COM): O módulo de comunicação é o único ponto de interacção directa entre *Servidor* e *Clientes*. Implementa processos de comunicação por RMI, *socket* e processa utilizando o protocolo HTTP. Realiza ainda os acessos a BD com JDBC. Este módulo é igual para o *Servidor* e *Clientes* e está detalhado na secção a seguir onde são descritos os módulos do *Cliente*.

3.2 Cliente

Um *Cliente* recebe uma UT, realiza as operações especificadas nessa UT e devolve o resultado ao servidor. O *Cliente* realiza em simultâneo a monitorização do estado da máquina em que executa. É da responsabilidade do *Cliente* activar e monitorizar o programa aplicação. Toda a comunicação entre o *Servidor* e *Cliente* é feita pelo módulo de comunicação. Um *Cliente* quando inicia regista-se no *Servidor*.

Um *Cliente* é constituído por quatro módulos:

- um módulo de monitorização do estado da máquina (MON),
- um módulo de execução de tarefas (TRAB),
- um módulo envolvente do programa aplicação (WRAP) e
- um módulo de comunicação (COM).

Módulo Controlo de Tarefas ou Trabalhador (TRAB): Este módulo interpreta as mensagens vindas do servidor e que são de três tipos: *execução de uma Unidade de Trabalho, terminação da tarefa actual e terminação de toda a actividade do Cliente*. No caso de uma mensagem para execução de uma tarefa este módulo interpreta os campos da especificação da UT. Vai buscar o programa aplicação, lança a aplicação, vai buscar os dados e guarda-os localmente no directório onde colocou a aplicação. A aplicação é carregada através de HTTP do servidor *Web* de aplicações. Os dados são carregados de uma BD usando JDBC. Na especificação da UT estão as *queries* SQL para acesso à BD e o *script* de execução da aplicação. O ficheiro resultado da execução da tarefa é colocado numa tabela da BD e comunicado ao servidor o fim da tarefa.

Módulo Envolvente da Aplicação (WRAP): Este módulo lê, interpreta linha a linha, o *script* de controlo da execução da tarefa, e coloca cada linha no *stdin* da aplicação e recolhe o seu *stdout* e *stderr*. Coloca o *stdout* num ficheiro como resultados da execução da tarefa.

Módulo de Monitorização de Estado dos Recursos (MON): Este módulo verifica o número de utilizadores e carga da máquina e comunica regularmente essa informação ao Servidor.

Módulo de Comunicações (COM): A comunicação é feita de acordo com o destinatário e a natureza da mensagem. O tipo de protocolo está indicado num campo XML da mensagem. As mensagens trocadas entre clientes e servidor utilizam RMI ou *sockets*. Existe mais um tipo de mensagem que requer o uso do HTTP e é utilizada para trazer a aplicação do servidor *Web* de aplicações para o local do cliente. Um último tipo de mensagens indica a utilização de JDBC para trazer os dados a serem processados pela aplicação. O módulo de comunicações é constituído por quatro sub-módulos e duas filas de mensagens: uma de mensagens a enviar e outra de mensagens recebidas. Cada sub-módulo é implementado por *threads* independentes que consultam filas de mensagens, identificam, e processam as mensagens que lhes competem. Cada um destes sub-módulos implementa um protocolo: sub-módulos RMI; sub-módulos *sockets*; sub-módulo HTTP e; sub-módulo JDBC. Os sub-módulos HTTP e JDBC respondem a mensagens dirigidas especificamente ao módulo de comunicações e são utilizados para buscar aplicações e dados, respectivamente.

4 Conclusões e trabalhos futuros

Neste artigo descrevemos a arquitectura e funcionamento do *Harvard* como plataforma para análise de grandes quantidades de dados. O uso da linguagem XML neste caso particular é fundamental não só pela facilidade em se codificar de forma estruturada um processo de análise de dados, mas também por proporcionar a interoperabilidade e integração com ambientes similares de análise

de dados. Também possibilida a criação de ferramentas de fácil interacção com o utilizador (analista de dados) e apresentação dos resultados. Destacamos que não utilizamos todo o potencial da linguagem nesta primeira versão do *Harvard*, e sim o que de essencial ela proporciona - a versatilidade. A versatilidade da linguagem XML foi um dos factores que permitiram seu uso no *Harvard*, seja ao nível de troca de mensagens, entrada de dados ou ainda na consolidação e apresentação de resultados. Dada a evolução da linguagem XML e das tecnologias agregadas, pretendemos expandir o *Harvard* para uma versão baseada em tecnologias *Web service*, inclusive com integração ao um ambiente *Grid Computing*, notadamente o *Globus Toolkit* (GT4). Pretende-se também efectuar a definição adequada de um *subset* da linguagem XML dedicado ao processo de KDD.

Referências

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
2. D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 8 Nov. 2004.
3. Tim Bray. Extensible markup language (xml) 1.0 (fourth edition). Technical report, W3C, 2006.
4. Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
5. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2)(2):115–128, 1997. Provides an overview of the Globus project and toolkit.
6. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 11, pages 259–278. MORGAN-KAUFMANN, 1999. The whole contents of the book is also useful for a complete understanding of Globus.
7. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 55–63, 7-9 Aug. 2001.
8. J. Han and M. Kamber, editors. *Data Mining: Concepts and Techniques*. Morgan-Kaufmann Publishers, 2001.
9. Paulo Heitlinger. *O guia pratico da XML*. Centro Atlantico Ltda, Lisboa, 10 2001.
10. H. Kargupta and P. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 2000.
11. M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, 13-17 June 1988.
12. Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

Adaptabilidade Web no SOM

Pedro Silva e José Paulo Leal

DCC-FC & LIACC, Universidade do Porto, Portugal
psilva@dcc.fc.up.pt
zp@ncc.up.pt
WWW: http://www.niaad.liacc.up.pt/Site-O-Matic

Resumo Neste artigo é descrita uma metodologia desenvolvida no âmbito do projecto Site-O-Matic (SOM) para introdução de adaptabilidade em sítios da internet. Esta baseia-se numa *framework*, com uma arquitectura orientada a serviços, constituída por um conjunto de módulos heterogéneos e que trocam entre si mensagens XML através de um *broker*.
A troca de mensagens recorre às tecnologias *Web Services* e XMLHttpRequest. Esta última é usada na comunicação com navegadores, sendo os *Web Services* usados para os restantes módulos. As mensagens trocadas representam uma linguagem de adaptabilidade onde se encontram os dados necessários aos diferentes passos da adaptação.
Neste artigo são focados diferentes aspectos da utilização das tecnologias XML na implementação da *framework*, incluindo: o desenho da linguagem de adaptação; a utilização de Ajax para construção e envio de mensagens e a sua conversão em HTML; a implementação de serviços *web* de manipulação de mensagens; a parametrização de módulos de adaptação. É também apresentado um exemplo de aplicação da metodologia proposta e é avaliado o impacto da sua utilização.

1 Introdução

O Site-O-Matic[7] é um projecto de investigação do grupo de Inteligência Artificial e Análise de Dados[8] (NIAAD) que tem como objectivo, o desenvolvimento de uma plataforma e uma metodologia para automatizar actividades relacionadas com a gestão de sítios. Exemplos dessas actividades, são a gestão do conteúdo, a sua estruturação, recomendação e personalização. O Site-O-Matic propõe-se ainda a ter em conta o comportamento dos utilizadores e os objectivos do sítio na aquisição dos seus próprios objectivos.
O Site-O-Matic debruça-se sobre três aspectos:

- Definir uma plataforma flexível para sitíos *web* que permita a aquisição de dados de qualidade, assim como transformações e adaptações *online* da estrutura e interface dos sítios;
- Desenvolver técnicas para realizar adaptações *web* usando *data mining*;
- Obter conteúdos focados em tópicos, sendo essa obtenção guiada por objectivos bem definidos e monitorização de valores de desempenho sobre os dados da utilização.

Para atingir estes objectivos este projecto dividiu-se em várias tarefas, uma das quais deu origem a este trabalho com o objectivo de definir uma metodologia e uma infra-estrutura que facilite a introdução de adaptabilidade em sítios *web*. A infra-estrutura deve permitir a modificação automática de páginas *web* usando mecanismos de adaptação que funcionem sobre dados de utilização. Deve também recolher esses dados necessários à adaptação.

1.1 Adaptação de sítios *web*

Para realizar adaptações sobre páginas *web*, qualquer solução tem de reunir a capacidade de fazer um conjunto de tarefas[1] necessárias à adaptação, das quais a nossa abordagem destaca:

- Recolha de dados de adaptação;
- Integração de mecanismos de adaptação;
- Aplicação de transformações.

A adaptação em função do utilizador requer algum conhecimento sobre as suas preferências, de modo a criar páginas com conteúdo relevante para o utilizador. Para isso, é necessário recolher dados para adaptação, que dêem a conhecer algo sobre o utilizador e que possam ser levados em conta na adaptação. Os dados sobre o utilizador têm um papel preponderante para os processos de adaptação. Os mecanismos que produzem adaptações usam-nos para moldar os seus resultados, conseguindo assim um maior refinamento e uma maior orientação ao utilizador. Depois de obtidos os resultados é necessário transformá-los em visualizações, por forma a reflectirem-se de alguma maneira na página que o utilizador está a consultar. Outras tarefas propostas[1] estão relacionadas com a utilização directa de um gestor de conteúdos.
Independentemente destas tarefas foi possível identificar com este trabalho duas visões de adaptação[2]:

Recomendação Adaptações em que são inseridos novos conteúdos.
Reformatação Adaptações em que o conteúdo não é alterado mas reorganizado, seja do ponto de vista da sua ordem relativa ou da formatação.

Em ambos os casos o objectivo é melhorar a experiência do utilizador.

1.2 Recolha de dados de adaptação

Para adaptar páginas *web* é necessário obter informações sobre a pessoa a quem se destina a adaptação.
Tradicionalmente os dados usados para atingir estes objectivos são os que se encontram nos ficheiros de registos dos servidores *web* e nos sistemas de informação[1][4]. Usando estes dados relacionam-se por exemplo página vistas, artigos comprados e dados pessoais. Contudo, estes mecanismos apresentam algumas limitações[1][4]. Por exemplo, não é possível saber se um utilizador que acedeu a uma determinada página a leu ou não. O utilizador pode de facto ter

acedido à página mas nunca sabemos se se ausentou, se leu todo o seu conteúdo ou apenas uma parte. Naturalmente este vazio de informação pode levar a interpretações erradas aquando da geração de conteúdos adaptados. Se conseguirmos perceber determinados comportamentos do utilizador podemos aumentar a qualidade da informação coleccionada sobre este. Aquilo que propomos é acrescentar aos dados tradicionais, que estão do lado do servidor *web* e dos servidores de informação, dados que estão disponíveis junto do utilizador através do navegador, nomeadamente eventos.
Ao processarmos eventos de rato como *click*, *focus*, *scroll* ou *select*, podemos inferir algum contexto da navegação do utilizador. Se um utilizador, ao ler um texto, seleccionar uma parte desse texto, podemos traduzir esse evento como interesse nesse texto. Do mesmo modo, se o utilizador deslocou a página (*scroll*) para conseguir ver uma parte que não estava visível, podemos interpretar esse movimento como interesse sobre a zona oculta da página. Podemos também criar **meta-eventos** para agregar um conjunto de eventos com outras variáveis como o tempo, para interpretar interesse ou desinteresse. Por exemplo, se durante um determinado espaço de tempo um utilizador não gerou eventos ou mudou várias vezes de página podemos considerar que houve um desinteresse sobre o que estava a ser apresentado. O mesmo se podendo dizer no caso inverso. Se um utilizador gerar uma série de eventos numa determinada página durante um período alargado de tempo podemos interpretá-los como interesse no conteúdo apresentado.

2 Arquitectura da *Framework*

As diferentes soluções existentes[1] apresentam abordagens monolíticas, e mesmo algumas que se dizem ser uma *framework* estão orientadas a um determinado gestor de conteúdos. A proposta que apresentamos é uma *framework* que apresenta uma arquitectura com características de uma arquitectura orientada a serviços[5] (SOA). Os dois componentes que compõem a *framework* propriamente dita são: o **Cliente** e ***Broker***.
O Cliente é constituído por uma biblioteca JavaScript[6] responsável pelas operações a realizar no navegador. O *Broker* é um componente central único, responsável pelos processos de comunicação e registo de dados. Esta solução não define como componente os geradores de adaptações ou **Adaptadores**, pois entende-se que estes devem ser deixados ao critério dos criadores de adaptações. A única imposição é que cada adaptador implemente um serviço *web* segundo uma dada especificação WSDL[5], podendo assim receber, processar e enviar mensagens XML.
Como se trata de uma arquitectura de inspiração SOA uma das características da *framework* é a utilização de mensagens na comunicação entre os diversos componentes. São geradas mensagens nos Clientes com a informação de utilização que são enviadas aos Adaptadores e são geradas mensagens nos Adaptadores com os resultados das adaptações que são enviadas para os Clientes.
Independentemente dos meios usados para a circulação destas mensagens, elas

cumprem um conjunto de regras estabelecidas *a priori* e que definem uma linguagem de adaptação. Esta definição estabelece uma linguagem que representa a informação necessária à adaptação, abstraindo essa definição da implementação.

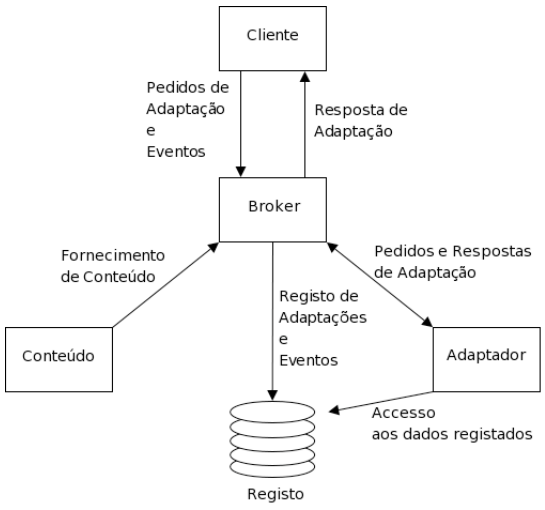

Figura 1. Proposta para arquitectura geral.

2.1 Cliente

O módulo Cliente tem duas funções principais: realizar operações sobre mensagens e realizar a reformatação e transformação de conteúdos. As operações sobre as mensagens incluem: a criação de mensagens que representam os dados a serem adaptados no Adaptador e enviadas através do *Broker*; a interpretação de mensagens que chegam do *Broker* provenientes dos Adaptadores, com os resultados da adaptação; e a criação de mensagens de notificação com os eventos gerados pelo utilizador. As operações de reformatação ou transformação são as operações sobre o conteúdo que visam reflectir as adaptações realizadas. Nas operações sobre as mensagens o Cliente deve ter a capacidade de construir e processar as mensagens mas também de as enviar e receber, se possível sem prejudicar a navegação do utilizador. No que diz respeito à manipulação do conteúdo o Cliente deve ter a capacidade de o manipular, se possível de forma simples e sem introduzir efeitos secundários. O Cliente é definido como **consumidor de adaptação**. Define uma API que permite invocar o processo de adaptação através do método `adapt(adaptor_id,adaption_parameters,element_id)`. Ao incluir este método no código HTML é iniciado o processo de adaptação. Os parâmetros deste

método indicam: a identificação do adaptador; eventuais parâmetros a passar ao adaptador; e o elemento da página sobre o qual será feita a adaptação.

2.2 *Broker*

O *Broker* é o ponto central da *framework*. Por ele passam todas as comunicações, tendo uma função assumidamente de *proxy* e *logger*. Mensagens de adaptação e notificação partem do Cliente, são analisadas e seguem os seus respectivos caminhos. As competências do *Broker* foram definidas da forma mais simples possível para não comprometerem a sua escalabilidade. Sendo o ponto central de toda a *framework* e processando todas as mensagens, é importante que o tempo gasto com cada uma seja o mais reduzido possível.
Num ambiente de produção, numa arquitectura distribuída como esta, irão existir n clientes e n adaptadores que irão contribuir para o escalar de mensagens no sistema. O *Broker* terá de ser capaz de lidar com diversas ligações, registos em base de dados e algum processamento de mensagens de forma eficiente para não prejudicar o sistema.

2.3 Adaptador

O Adaptador não é um componente da *framework*, mas é um componente fundamental no processo de adaptação. É ele que possui a capacidade de gerar adaptações com os seus algoritmos específicos. Por outro lado tem de ter a capacidade de comunicar usando a mesma linguagem que os restantes componentes da *framework* esperam e acordaram inicialmente. Cada adaptador tem definido através de uma especificação WSDL o seu interface público, o seu endereço e o tipo de comunicação a usar.
Os Adaptadores são definidos como **fornecedores de adaptação**.

3 Linguagem de adaptação

A linguagem de adaptação do SOM permite abstrair os dados de adaptação, passando estes a estar contidos exclusivamente no conteúdo das mensagens. Na descrição da *framework* definiu-se como objectivo de concepção que toda a informação trocada entre os vários componentes fosse explicitada nas mensagens. Por exemplo, evitou-se que alguma informação fosse codificada nas mensagens HTTP, nomeadamente em *cookies*, normalmente usadas pela maioria das soluções existentes[1][4].
Devido ao uso do XML as mensagens são facilmente legíveis por humanos permitindo durante o processo de desenvolvimento inspeccionar as informações duma determinada comunicação, com vantagens evidentes durante o *debugging*.

3.1 Estrutura das Mensagens

Como já referimos, as mensagens que circulam na *framework* têm um tipo predefinido. Essa definição em XMLSchema é também usada pelo *Broker* para validar

as mensagens em circulação. Deste modo é garantida a validade das mensagens em circulação na *framework*. Isto serve para evitar, por exemplo, mensagens destinadas a corromper ou utilizar o sistema com outros fins que não o da adaptação. As figuras seguintes são exemplos de alguns elementos que compõem as mensagens. Existe um elemento de topo `message`, que contém como informação adici-

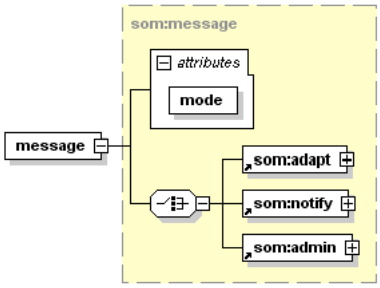

Figura 2. Elemento `message`.

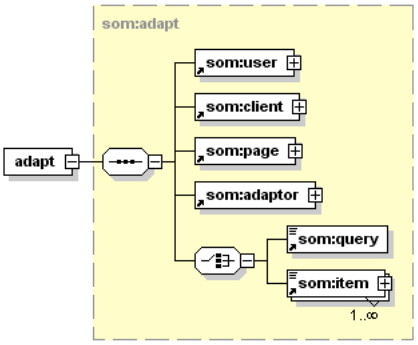

Figura 3. Elemento `adapt`.

onal de contexto um atributo `mode` indicando o tipo de mensagem (`request` ou `response`). Este elemento pode albergar três tipos de elementos e que correspondem no fundo aos três tipos de mensagens possíveis:

`adapt` Mensagens de adaptação que circulam entre os componentes com os respectivos pedidos e respostas de adptação.

`notify` Mensagens de notificação que são enviadas pelo Cliente para efeitos de recolha de dados.

`admin` Mensagens de administração que são usadas para diversas tarefas de manutenção e configuração do sistema.

Em todos os casos há elementos comuns contidos nas três definições, como por exemplo informação sobre o utilizador ou a página. Apenas diferem em alguns elementos que são significativos para o seu tipo. É o caso das mensagens de adaptação que contêm informação sobre o adaptador que deve ser usado, ou o caso das de notificação que contêm conteúdos específicos, resultado de eventos processados junto do utilizador.

4 Comunicação e Processamento

Para utilizar mensagens XML na adaptação foi necessário encontrar meios para a sua construção e comunicação. Para o Cliente a solução passou pela utilização de **Ajax**[9] e para a comunicação entre o *Broker* e os Adaptadores a escolha recaiu sobre os ***Web Services***[5]. Em particular esta escolha deveu-se às necessidades de cada módulo consoante a sua localização.

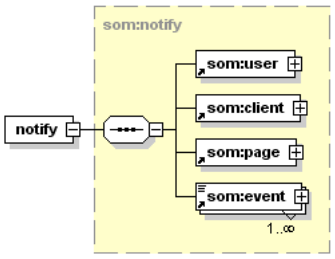

Figura 4. Elemento `notify`.

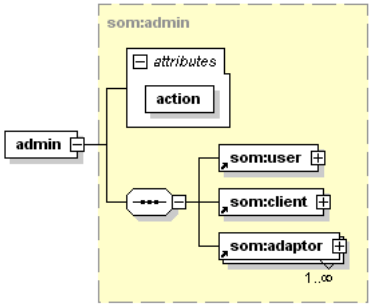

Figura 5. Elemento `admin`.

Como um dos objectivos era introduzir adaptabilidade em sítios já existentes, no caso do cliente a solução encontrada passou pela introdução de uma biblioteca JavaScript no código HTML da página. A biblioteca define uma função que é accionada em cada carregamento de página e despoleta todo o processo de adaptação. Desta forma é possível ligar e desligar facilmente a adaptação do lado do cliente e de modo selectivo.
No caso dos Adaptadores a solução dos *web services* foi escolhida por permitir independência de tecnologia. Usando *web services* não se impõem nenhuma obrigatoriedade sobre a a construção dos Adaptadores. Estes podem ser construídos em qualquer linguagem com capacidade de utilização de um motor SOAP, algo que muitas das linguagens hoje existentes já suportam. Ambas as soluções permitem manter nas mensagens a definição e especificação da adaptação, tornando a camada de implementação completamente independente.

4.1 Ajax

O Ajax - Assynchronous JavaScript and XML, é uma designação comum para um conjunto de tecnologias que permitem a realização de pedidos HTTP e alterações na estrutura e conteúdos de páginas HTML sem necessidade de as redesenhar na totalidade. Para realizar os pedidos HTTP o Ajax usa um objecto JavaScript, o XMLHttpRequest, que permite realizar pedidos de forma síncrona ou assíncrona. Os pedidos assíncronos tem a particularidade de permitir que o processamento não seja bloqueado pelo aguardar da resposta. Isto faz com que uma página possa ser utilizada e simultaneamente realize pedidos HTTP, sem que o utilizador se aperceba. Com este método, fica facilitada a troca de informações que sirvam de complemento a um sítio. A utilização do Ajax traz ainda algumas vantagens adicionais, em particular a diminuição da largura de faixa usada. Como os pedidos feitos com recurso ao Ajax transportam apenas a informação necessária contribuem para um descongestionamento das vias de comunicação e dos servidores.
O Ajax é usado na *framework* em três tarefas:

– Construção de mensagens de adaptação;

– Comunicação assíncrona de dados;
– Reformatação da página a adaptar.

A construção de mensagens de adaptação é feita de duas maneiras: compilando dados existentes na página corrente e através do processamento de eventos. Os eventos permitem recolher informações que apenas se encontram do lado do navegador, mais concretamente permitem obter dados sobre a utilização real das páginas por parte de um determinado utilizador.
A construção de mensagens de adaptação assim como a modelação da página, depois de obtida a resposta de adaptação, recorrem à DOM para aceder e navegar no seu conteúdo. Usando os objectos que a DOM disponibiliza, a biblioteca de JavaScript consegue ler e escrever directamente sobre a página visualizada. Isto permite acelerar o processo de actualização da página visto que se processa directamente sobre o objecto em memória, tornando as alterações instantâneas. Toda a comunicação de envio e recepção é feita assincronamente permitindo que o utilizador não seja afectado enquanto decorrem este processos. O utilizador tem sempre acesso à página pedida independentemente da demora do processamento de adaptação.
De salientar que a utilização de Ajax ficou a dever-se ao facto de o JavaScript ser uma linguagem largamente aceite pela maioria dos *browsers*. No entanto outras linguagem podem ser usadas desde que tenham capacidade de processar XML e comunicar assincronamente, como por exemplo o Flash.

4.2 *Web Services*

Os *web services* permitem o estabelecimento de ligações ponto a ponto de forma transparente e autónoma. Recorrendo ao protocolo SOAP os *web services* criam canais de comunicação por onde circulam mensagens XML. O facto de os *web services* apresentarem características que se encaixam no modelo SOA influenciou a nossa escolha. A existência de um interface público que permite dar a conhecer o serviço sem que seja conhecida a sua implementação permite que os diferentes componentes possam trocar mensagens facilmente. No caso da solução proposta, a utilização do motor SOAP Axis[10], permitiu usar um método de comunicação que facilita a integração de documentos XML nos envelopes SOAP. O Axis suporta quatro métodos de comunicação : *RPC*, *Document*, *Wrapped* e *Message*. Os serviços `RPC` destinam-se à invocação de métodos mapeando os elementos do XML com os argumentos do método. Os serviços `Document` e `Wrapped` fazem mapeamento de objectos entre o Java e o XML e são também orientados à invocação de métodos. Por fim, os serviços do tipo `Message` têm como característica deixarem para a implementação do próprio serviço o processamento do corpo das mensagens SOAP, permitindo que estes manipulem directamente o XML das mensagens geradas.

4.3 Interface Message

Apesar de implementados em linguagens diferentes e usarem protocolos diferentes de comunicação Cliente, *Broker* e Adaptador comunicam segundo a mesma

linguagem, a linguagem de adaptação. Esta linguagem é comum a todos os componentes e como tal as necessidades para o seu processamento e manipulação são também comuns. Isto permitiu definir um interface para as mensagens. O

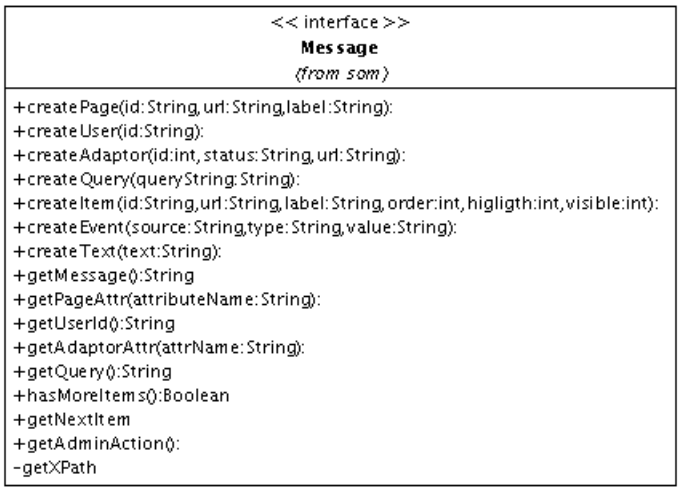

Figura 6. Diagrama de classes do interface `Message`.

interface `Message` apresentado na figura 6 é o esqueleto para qualquer implementação que necessite de lidar com as mensagens que circulam nos componentes. Esse interface foi criado com o intuito de homogeneizar processos e acelerar a implementação de funções que são comuns aos diferentes componentes. Há no entanto que salientar que este interface é implementado em diferentes linguagens, podendo, em algumas, não ser possível codificá-lo usando os recursos da linguagem. Este é o caso do módulo Cliente implementado em JavaScript. Apesar disso, a implementação em JavaScript e em Java seguiu os mesmos padrões, mantendo nomes de métodos e o modo como são usados, o que nos permite dizer no fim que ambos os módulos partilham um mesmo interface.
A definição deste interface, independente das linguagens de implementação, simplifica a construção e leitura de mensagens de adaptação nos fornecedores e consumidores de adaptação.

4.4 Adaptadores

Os adaptadores são um ponto de extensão da *framework* que através do WSDL ficam disponíveis para realizar adaptações. Este é caso dos adaptadores simples já incluídos na *framework* para efeitos de testes. No entanto, a *framework* inclui um **Meta-Adaptador** para permitir a inclusão rápida de programas externos

em linguagens de *scripting*. Esta capacidade tem como objectivo permitir aos investigadores, integrar rapidamente o seu código com a *framework*, escrito na sua linguagem de *scripting* favorita. Para conseguir isto definiu-se uma configuração em XML que indica ao meta-adaptador como interagir com o programa externo. A linguagem define um conjunto de regras que permitem através dos canais de `output` e `input` interagir com o programa externo.
As instruções mapeiam elementos do XML, como por exemplo `commandLine`, `queryCommand` ou `getItemProperty` respectivamente com a linha de comandos a usar para lançar o programa, o método que permite executar uma *query* ou o método que permite obter uma determinada propriedade. Foi este método que foi usado para construir os adaptadores de teste usados na aplicação desenvolvida para realizar a validação e avaliação da solução.
A utilização deste meta-adaptador é opcional podendo, se assim se desejar, desenvolver um adaptador que implemente um *web service* ficando directamente disponível para o *Broker*. Aliás, em ambientes de produção é de esperar que os Adaptadores tenham implementações mais robustas e de maior desempenho.

5 Avaliação

Como prova de conceito foi criada uma aplicação *web* que faz uso desta *framework*. Com esta aplicação foi possível observar o impacto da solução proposta e avaliar a aplicabilidade em ambientes de produção.
A aplicação criada foi um leitor *web* de manuais[1]. Ao aceder à aplicação um utilizador depara-se com uma página dividida em três zonas: uma zona de topo com um campo de pesquisa para indicar o tópico a consultar; uma zona central onde é visualizada a informação pedida; e uma zona lateral onde são apresentados conteúdos adaptados ao utilizador. Quando um utilizador acede ao sistema e realiza um pedido de uma página, um pedido de adaptação é gerado e é enviado assincronamente para o *Broker*. De notar, que neste ponto o utilizador pode já estar a ver a página pedida. Chegado ao *Broker*, para além de registar numa base de dados o pedido, a mensagem gerada no cliente é enviada ao Adaptador, por forma que este a possa processar. Quando o Adaptador termina as suas tarefas constrói uma mensagem de resposta com o conteúdo adaptado. No caso da aplicação criada trata-se apenas da compilação das últimas páginas vistas, ou seja um histórico. Essa mensagem é então encaminhada ao *Broker* e por sua vez ao Cliente que irá tratar de processar a mensagem e realizar as alterações necessárias à página. Neste caso estas alterações baseiam-se na construção de um menu lateral com a indicação do histórico.
Paralelamente a estas actividades a monitorização de eventos regista um conjunto de dados pré-definidos: *click*, *scroll*, ou *focus* que vão sendo registadas na base de dados. A aplicação é simples mas permite observar o funcionamento de todos os componentes e avaliá-los.

[1] O leitor de manuais foi testado com os manuais do Unix (*man pages*).

5.1 Resultados e observações

A avaliação da solução foi feita recorrendo a ferramentas de teste unitários e medições de tempos de resposta e processamento. Nas várias medições feitas registou-se um comportamento aceitável dos vários componentes mesmo em situações de utilização intensiva, isto é, com vários adaptadores e utilizadores. Observando o gráfico da figura 7 podemos ver que à medida que aumenta o

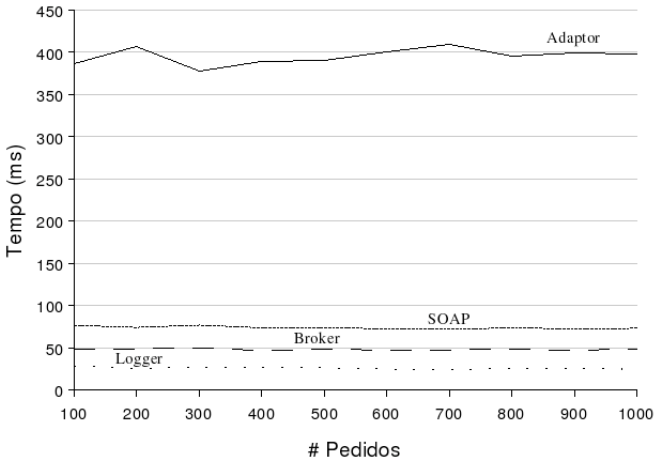

Figura 7. Tempos médios em ms, para 3 clientes e uma configuração de página com 2 adaptadores, com o número de pedidos a variar entre 100 e 1000.

número de pedidos o comportamento do *Broker*, *Logger* e da comunicação se mantêm quase constantes. Apenas o Adaptador apresenta valores elevados em virtude do mecanismo usado para interagir com o programa externo. A implementação na linguagem de *scripting* e o recurso aos canais de *input* e *output* limitam o seu desempenho. Para além deste teste foram também realizados testes de desempenho onde foram medidos os tempos para uma utilização com uma determinada sequência de pedidos. Os valores obtidos para esse teste assemelham-se aos do gráfico da figura 7.

6 Conclusões e trabalho futuro

Neste trabalho foi descrita uma metodologia para a introdução de adaptabilidade em sítios *web*. A solução apresentada cumpre os objectivos propostos através da definição desta *framework* e recorrendo a tecnologias como o Ajax, os *Web Services* e o XML. Esta solução permitiu-nos ter:

– Independência das tecnologias *web*;

– Abstracção das operações de adaptação;
– Expansibilidade dos módulos de adaptação.

Os resultados obtidos nos testes mostram que os benefícios da adaptabilidade não diminuem a qualidade da interacção para o utilizador final. Acresce ainda o facto de se ter conseguido introduzir um novo ponto de recolha de dados que vem trazer novas capacidades aos adaptadores de conteúdo.
Como se trata de um protótipo há ainda algum pontos a tratar no futuro entre os quais estão:

– Utilizar métodos e bibliotecas compatíveis em diversos *browsers*, para permitir uma fácil integração da biblioteca JavaScript fornecida com a *framework*;
– Permitir a utilização de modelos para especificar o modo como os dados de e para adaptação devem ser tratados no cliente, mantendo as características da página original;
– Definir interfaces de administração para tarefas básicas de configuração do *Broker* ou Adaptadores;
– Utilizar métodos para o registo de Adaptadores, como por exemplo o UDDI;
– Definir métodos de teste apropriados para aplicações que façam uso do Ajax como é o caso do protótipo criado, permitindo assim obter dados sobre o cliente duma maneira mais sistemática e menos empírica;
– Realizar testes com um grupo de utilizadores para fazer uma avaliação qualitativa, usando um sítio de produção com utilização regular.

Referências

1. Eirinaki, M., Vazirgiannis, M: Web Minig for Web Personalization. ACM Transactions on Internet Technology 3,1 1-27 Feb. 2003
2. Mikroyannidis, A., Theodoulis, B. A Theoretical Framework and an Implementation Architecture for Self Adaptive Web Sites. Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on 20-24 Sept. 2004 pages: 558 - 561
3. Mobasher, B., Cooley, R, Srivastava J. Automatic personalization based on Web usage mining. Comunications of the ACM August 2000 43,8
4. Jaideep Srivastava,Robert Cooley,Mukund Deshp,Pang-ning Tan Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. SIGKDD Explorations Jan. 2000 1,2 12-23
5. Thomas Erl: Service-Oriented Architecture - Concepts, Technology, and Design. Prentice Hall, 2005
6. David Flanagan: JavaScript The Definitive Guide. O'Reilly, 2002
7. Site-O-Matic: http://www.niaad.liacc.up.pt/Site-O-Matic
8. Núcleo de Inteligência Artificial e Análise de Dados: http://www.niaad.liacc.up.pt/
9. Mozilla Developer Center - Ajax: http://developer.mozilla.org/en/docs/AJAX
10. Apache Axis: http://ws.apache.org/axis

Integrador Automático de Notícias

Daniel Silva, Pedro Abreu, Pedro Mendes, e Vasco Vinhas

Faculdade de Engenharia da Universidade do Porto Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal

Resumo A proliferação dos serviços noticiosos na Internet resulta da crescente importância deste meio de comunicação no que concerne à publicação de conteúdos, tendo as entidades competentes definido um conjunto de normas por via a consolidar o processo de difusão. Contudo, constatam-se elevados níveis de dispersão e não unificação de fontes de dados. Neste contexto identificam-se oportunidades de aplicação de conceitos de informação semi-estruturada, armazenamento e pesquisa orientada ao documento. É proposta uma solução capaz de fornecer serviços de integração e pesquisa no contexto da problemática exposta, recorrendo a XML e tecnologias associadas: XSD, XSL, XPath, XQuery, XUpdate e eXist. Utilizando as tecnologias mencionadas alcançou-se o acesso automatizado às fontes de dados, conversão da informação semi-estruturada para formato normalizado, actualização automática dos conteúdos locais, e interfaces de administração para gestão de fontes e utilizadores, assim como pesquisa de informação, gestão de subscrições e organização em árvore multinível das fontes de dados.

1 Introdução

Os serviços noticiosos disponíveis na *web* consistem numa das formas mais eficientes para qualquer indivíduo com pretensões de se manter informado com elevada regularidade em determinadas áreas de conhecimento. Com o progresso da tecnologia e a evolução das normas de representação, as notícias têm vindo a perder o seu impacto como forma única de manter um dado sítio *web*. Actualmente, os utilizadores procuram formas mais directas de agregar a informação noticiosa, potencialmente proveniente de diversas fontes, livres de eventuais conteúdos publicitários e com menores necessidades a nível de exigências de rede, assegurando que os dados provenientes da Internet contêm na sua globalidade informação noticiosa.

Foi neste contexto que surgiu, por volta do ano 2000, a norma RSS (*Really Simple Syndication* na versão 2.0)[1], que consiste num dialecto XML[2] normalizado tendo em vista o armazenamento simplificado de informação e meta informação noticiosa. As entidades produtoras de conteúdos, se assim o entenderem, têm neste formato um novo meio para os publicar, sendo assegurado que do lado dos utilizadores novas formas são oferecidas para os integrar em aplicações de âmbito mais alargado ou especificamente orientadas a este domínio. Nesta gama de ferramentas, existem já muitas que oferecem ao utilizador formas mais ou menos eficazes de garantir uma actualização constante dos conteúdos.

Dada a relevância da norma no domínio da publicação de conteúdos, as ferramentas existentes para consulta têm na sua maioria uma componente comercial associada e surgem das mais diversas formas. Existem ferramentas *online*, *desktop*, tanto multi-plataforma como orientadas a determinado sistema operativo e, sob a forma de extensões para outras aplicações. Serão por ventura estas últimas as mais utilizadas, na medida em que estão associadas a ferramentas de uso diário dos utilizadores comuns, como clientes de correio electrónico e navegadores *web*, levando a uma coexistência de funcionalidades em tudo útil na eficiente gestão do tempo que o utilizador tem disponível para a consulta de conteúdos[3].

De entre estas a destacar o *Mozilla Thunderbird*[4], que sendo um cliente de correio electrónico, integra nas suas principais funcionalidades um agregador de RSS. Esta ferramenta permite uma gestão personalizada das *feeds* subscritas mesmo que, protegidas por camadas de autenticação e apresenta diálogos de busca avançados permitindo na situação mais complexa especificação de intervalos de tempo. O acesso distribuído é contudo limitado pois implica conhecimentos por parte do utilizador de criação de configurações distribuídas de aplicações. Para o cliente de correio electrónico da *Microsoft*, existem extensões desenvolvidas por *third-parties*, de entre as quais se destacam o *Attensa*[5] e o *intraVnews*[6] contudo face à aplicação da *Mozilla*, apresentam algumas limitações. O *Attensa* apenas funciona com a versão inglesa do *Microsoft Oulook*, e apresenta erroneamente caracteres nas notícias usados em outros idiomas. O *intraVnews* não apresenta estas deficiências, suportando ainda a norma *Atom*[7] todavia, a sua instalação só é possível em *Microsoft Outlook XP* ou *2003*.

Os navegadores *web* em geral também apresentam algumas funcionalidades de forma a facilitar a consulta deste tipo de conteúdos. Maior destaque poderá ser dado ao *Mozilla Firefox*[8] que nas primeiras versões (1.0) tinha possibilidade através de extensões evidenciar as notícias desta forma. Na versão actual (2.0), estas funcionalidades encontram-se presentes no pacote base da aplicação. A nova versão do navegador da *Microsoft* (*Internet Explorer 7.0*[9]), apresenta igualmente funcionalidades neste âmbito. A presença deste tipo de funcionalidades é hoje factor decisivo no que concerne à avaliação de um navegador *web*.

Todavia, a relevância dos serviços noticiosos não se remete apenas aos conteúdos publicados numa perspectiva actual ou local. A forma como surgem e os canais onde são publicados são aspectos extremamente importantes e mesmo críticos em diversas situações e contextos, pois destes factores muitas vezes se inferem questões como relevância sócio-cultural ou mesmo político-económica. É precisamente neste aspecto que a grande maioria das ferramentas disponíveis apresenta algumas falhas, não sendo as funções e metodologias de pesquisa facilitadas nem alargadas aos múltiplos campos característicos de uma notícia. Ainda neste aspecto, as notícias, quando descarregadas da Internet são armazenadas em sistemas de ficheiros, quase sempre sem qualquer tipo de indexação, quer sobre conteúdos textuais, quer sobre metadados associados. Não existe igualmente uma preocupação evidente em manter um repositório de notícias, o que impossibilita a caracterização de um dado fenómeno se à sua ocorrência estiver associado um intervalo de tempo considerável. A título de exemplo deste tipo de utilização,

podemos invocar a recuperação de notícias relacionadas com a problemática do défice orçamental de Portugal, tema recorrente nos últimos anos.

O projecto desenvolvido pretende dar resposta a alguns destes problemas, nomeadamente a construção e manutenção de um arquivo dinâmico de conteúdos noticiosos, oferecendo ainda a flexibilidade necessária para que os dados a manipular sejam apresentados de uma forma personalizada a cada utilizador.

Ao longo do presente documento é realizada, no capítulo 2, uma análise às diversas tecnologias associadas ao conceito de informação semi-estruturada. No capítulo 3 são apresentados e descritos os componentes genéricos do projecto, enquanto o quarto encerra a divulgação dos resultados. O 5º capítulo é dedicado à identificação das conclusões e melhoramentos futuros.

2 Informação Semi-Estruturada

A quantidade de informação hoje disponível para o público em geral teve um incremento sem precedentes. O acesso aos conteúdos, que têm verdadeiramente relevância para quem os pesquisa, é actualmente mais difícil, dado o elevado número de fontes disponíveis e a grande diversidade de meios para estruturar a informação.

Formatos completamente estruturados exigem definição de normas e desenvolvimento de ferramentas sofisticadas que permitam a sua eficiente manipulação, muito embora tais características restrinjam a disponibilidade da informação, levando a formatos fechados e dificilmente descodificáveis.

A alternativa não estruturada remete os métodos de pesquisa para técnicas de processamento que procuram simular a inteligência humana no que concerne aos processos interpretativos, o que pelos padrões actuais é ainda algo que está longe de ser atingido, nomeadamente ao nível da correlações de temas e respectiva localização temporal.

A solução de compromisso passa pela adopção de métodos semi-estruturados, de armazenamento de informação, que tornam possível a associação entre informação e meta-informação relevantes na captação da significância dos dados. Neste âmbito, a meta-linguagem XML, definida pelo W3C é o padrão corrente.

2.1 Processamento

Os documentos XML, devido à sua estrutura de anotações, podem ser representados esquematicamente como árvores em memória. Este facto foi determinante na concepção das duas principais APIs que definem as formas de processar e manipular os documentos.

DOM (*Document Object Model*)[10] especifica o carregamento completo da árvore para memória, fornecendo, posteriormente, um conjunto de métodos por via a permitir acesso e alteração dinâmica dos nós constituintes da mesma. SAX (*Simple* API *for* XML)[11] oferece uma abordagem guiada por eventos, emitidos pelos componentes validadores, permitindo o registo dos *handlers* correspondentes e consequente execução de tarefas específicas. Esta abordagem é mais

eficiente no que respeita aos recursos necessários ao processamento dos documentos, sendo contudo menos flexível do que DOM aquando da existência de necessidade de manipulação do documento a nível da sua estrutura.

Uma outra forma de processar documentos XML passa pela especificação de folhas de estilo XSL[12]. Os documentos XSL, também eles XML, definem um conjunto de primitivas que possibilitam a especificação de acções a despoletar aquando do processamento por um motor adequado, de forma semelhante à abordagem com SAX. Genericamente, as acções focam-se na criação de um outro documento, tipicamente também ele XML, sendo esta uma das formas mais compactas e concisas de especificar conversões entre ficheiros obedecendo a esquemas de dados distintos.

2.2 Interrogação

Na medida em que os ficheiros XML apresentam a versatilidade necessária para possibilitar a representação de informação oriunda de diversos domínios, a necessidade de criar uma linguagem padrão para interrogação sistemática tornou-se emergente.

XQuery[13] surge neste contexto. A linguagem retira alguns conceitos de outras conhecidas de interrogação como SQL, e OQL e poder-se-á mesmo considerar como uma extensão da segunda versão de XPath[14], na medida em que os resultados devolvidos por perguntas semelhantes nas duas são equivalentes. XQuery necessita, contudo, de blocos de informação representados em XPath 1.0 para satisfazer necessidades de localização unívoca de nós pretendidos. Pretende ser sobretudo um padrão de fácil compreensão, um pouco à imagem do SQL, para as bases de dados relacionais, adaptado a documentos semi-estruturados.

Apesar de ser completo a nível de capacidades interrogativas, não inclui ainda mecanismos que permitam alterar, dinamicamente, quer o conteúdo, quer a estrutura de documentos. Para atingir tais fins, existem outras normas que definem formas para actualizar os documentos em causa. Entre elas, é de destacar o XUpdate[15] da XML:DB[16] que define uma estrutura para documentos XML cujo conteúdo exprime operações de interrogação e actualização.

2.3 Armazenamento

O armazenamento de informação formatada segundo XML é atingível de diversas formas, tendo cada uma delas um conjunto de valências e aspectos mais prejudiciais neste âmbito[17].

A forma mais imediatista de atingir este objectivo passa pelo simples armazenamento num sistema de ficheiros. Desta forma os tempos de acesso são completamente determinados pela arquitectura física que o suporta e pelas características do sistema operativo que o mantém.

Outra forma de manter a informação persistente consiste em armazenar como BLOBS (*Binary Large Objects*) ou CLOBS (*Character Large Objects*) em bases de dados puramente relacionais. Seguindo esta abordagem, há obviamente perda

de informação no que concerne aos metadados associados, muito embora alguns fornecedores deste tipo de produtos tenham vindo a incluir tipos de dados e estruturas orientadas a documentos XML como é o caso da *Oracle*[18] e *PostgresSQL*[19].

A terceira alternativa consiste em ter bases de dados orientadas especificamente a XML, usando esta estrutura para assim armazenar nativamente os documentos em causa. Este tipo de método de armazenamento é mais adequado quando a informação a salvaguardar consiste primordialmente em documentos com grandes porções contínuas de texto.

Para o projecto, optou-se pela última abordagem dado que a informação noticiosa integra em si os conceitos que mais partido tiram da mesma.

3 Aplicação

O Integrador Automático de Notícias foi desenhado segundo a arquitectura de três camadas para aplicações baseadas no paradigma cliente/servidor. Este desenho proporciona, quando comparado com arquitecturas de camada dupla, um incremento dos níveis de desempenho, flexibilidade, manutenção, reusabilidade, e escalabilidade[20]. Com o intuito de promover a inteligibilidade do funciona-

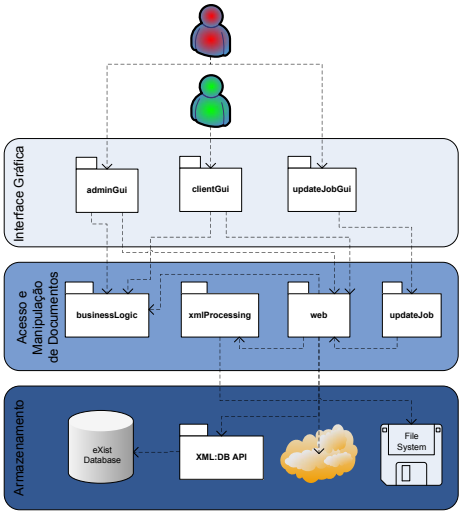

Figura 1. Diagrama da Arquitectura do Integrador Automático de Notícias

mento multinível do sistema, optou-se por apresentar em secções individuais cada um dos seus componentes agregadores, ilustrados através da Figura 1.

3.1 Armazenamento e Esquema de Dados

Todos os documentos relacionados com a aplicação são armazenados em formato XML na base dados nativa XML eXist[21]. Os documentos encontram-se divididos de acordo com o seu âmbito de utilização no sistema geral. Deste modo, é realizada uma primeira divisão dicotómica de conteúdos entre documentos referentes a repositórios de notícias e documentos relacionados com a administração e configuração da aplicação.

Seguindo este último ramo, encontram-se as colecções respeitantes à informação relativa às fontes com o respectivo período de actualização de dados e descrição textual, encontrando-se organizadas em categorias, de forma a facilitar a sua gestão. Ainda no directório de administração é armazenado o documento com a informação relativa aos utilizadores do sistema, de modo a possibilitar os processos de autenticação. Para além da informação pessoal de cada utilizador e das suas credenciais de autenticação, é armazenada a sua organização pessoal das fontes de dados numa estrutura em árvore multinível, em tudo semelhante à tradicional organização de sistemas de ficheiros[22]. Esta opção possibilita que cada utilizador personalize a organização das fontes de dados do modo que mais lhe aprouver, potenciando o grau de usabilidade da aplicação. De referir, por último, que o esquema de dados permite que uma mesma fonte de dados se encontre replicada em diversas (sub)categorias.

O ramo do repositório de notícias encontra-se estruturado de modo aplanado, procedendo-se à constituição de um documento por cada fonte de dados. A actualização das notícias por documento é responsabilidade do módulo descrito na secção 3.3. Tendo em consideração o facto da norma RSS não contemplar um identificador unívoco para cada entrada noticiosa, nem obrigar a que o campo *data de publicação* seja consistente com sua a posição relativa no documento, o modelo de actualização do Integrador tem por pressuposto que as entidades responsáveis pela publicação dos conteúdos noticiosos procedem à modificação dos documentos RSS segundo o modelo FIFO[22], procedendo à identificação de cada entrada pelo seu título. Esta assumpção, através dos diversos testes realizados a múltiplas fontes de dados revelou-se perfeitamente válida e, mesmo que venha a ser quebrada, o único impacto na aplicação traduz-se na repetição ou exclusão de algumas notícias. Esta decisão foi assumida após o estudo experimental de diversas fontes de dados de referência, quer de âmbito nacional, quer internacional, concluindo-se que a única análise consistente verificada foi a da opção descrita, não se verificando outras tidas como válidas *a priori*, tais como a análise posicional por data de publicação e comparação de diversos campos.

3.2 Acesso e Manipulação de Documentos

Este conjunto de pacotes, em conjunto com o componente responsável pela actualização periódica de informação, descrito no ponto seguinte, enquadra-se na camada de lógica de negócio do modelo seguido. Resulta da necessidade de encapsular um vasto conjunto de funcionalidades directamente relacionadas com o âmago da aplicação, para posterior utilização por parte das interfaces com o

utilizador. De entre este conjunto vasto de funcionalidades são de destacar as áreas identificadas na Figura 1.

Analisando em particular cada um dos referidos pacotes, refere-se, de imediato, o *xmlProcessing*, responsável pela manipulação e transformação da totalidade dos conteúdos noticiosos guardados, numa primeira fase, sem tratamento, no sistema de ficheiros local. A estes dados não tratados, resultado do acesso aos documentos publicados na *web*, é-lhes aplicada uma transformação XSL com o intuito de seleccionar apenas as notícias ainda não armazenadas, descartando simultaneamente diversos elementos, promovendo a conversão para um formato universal a todas as fontes tratadas pelo Integrador.

O pacote *businessLogic* encerra a representação das entidades mais significativas para a aplicação sob a forma de classes. De grosso modo, assume a responsabilidade das funções de mapeamento entre as realidades salvaguardadas nos documentos em base de dados e entidades manipuláveis pelas demais classes do Integrador. Para melhor compreensão, e a título de exemplo não exaustivo, é possível manipular entidades representadoras de clientes (entendidos como os utilizadores finais da aplicação responsáveis pela execução de consultas), notícias e fontes de dados.

Por último, e revestindo-se de uma importância superior, encontra-se o pacote *web* que tem um conjunto relativamente vasto de classes que suportam todas as actividades relacionadas com o acesso a dados, encontrando-se estes, quer em base de dados nativa XML, quer nos locais referenciados como pontos de publicação das entidades noticiosas monitorizadas. No caso dos documentos se encontrarem em base de dados, foi desenvolvido um conjunto de operações que, para além do simples acesso, permitem a inserção, edição e remoção de conteúdo, utilizando para tal efeito as tecnologias apresentadas na secção 2.2.

3.3 Actualização Periódica de Informação

De modo a garantir uma actualização constante dos documentos contendo informação noticiosa, para as fontes declaradas segundo o modelo exposto na secção 3.1, foi desenvolvido um serviço tendo em vista a execução periódica de tarefas, segundo intervalos de tempo definidos. O serviço de actualização está estruturado de forma a garantir a execução paralela das várias tarefas de actualização, seguindo uma arquitectura caracterizada por múltiplos fios de execução. Apesar deste aspecto ser, indubitavelmente, aquele que se reveste de maior relevância no contexto do projecto, e igualmente no componente em análise, duas outras funcionalidades foram desenvolvidas, sem as quais, este não poderia ser considerado completo. Estas duas funcionalidades estão relacionadas com os seguintes pontos:

- Construção das estruturas de dados que materializam as tarefas a executar. Envolve naturalmente o recurso às facilidades oferecidas pelo componente previamente descrito na secção 3.2.
- Cancelamento das tarefas activas. Sempre que o utilizador directamente desencadeie este conjunto de actividades e nas situações em que ocorram erros

de execução, o módulo desactiva todas as tarefas activas, esperando que as que estejam eventualmente a correr terminem a sua execução.

3.4 Interface Gráfica

A interface gráfica com o utilizador foi desenvolvida tendo por base a plataforma de desenvolvimento *Java Swing*[23]. Esta opção justifica-se com a necessidade da utilização de um meio de eficácia comprovada de interacção com humanos, sendo simultaneamente importante manter a independência relativamente ao sistema operativo dos sistemas dos utilizadores finais. Tal como é perceptível através da estruturação do presente capítulo, da consulta da Figura 1, e da leitura da secção 3.1, em especial o parágrafo dedicado à explicitação da organização da estrutura de dados, foram desenvolvidas interfaces tendo em vista a sua utilização por utilizadores de dois níveis distintos: o administrador da aplicação, de cariz mais técnico, e o agente pesquisador de notícias, sem necessidade de competências informáticas.

Os pontos seguintes esclarecem a natureza destes dois tipos de utilizador ao mesmo tempo que apresentam, de modo sumário, as principais funcionalidades das interfaces.

Gestão de Actualização Graficamente, o serviço de gestão da actualização periódica de informação, descrito na secção 3.3, materializa-se num *tray icon* localizado na *system tray*. Esta opção é justificada pelo número reduzido de operações permitidas ao utilizador. Por outro lado, trata-se também de uma forma de manter o controlo do serviço, espacialmente próximo da localização habitual dos mecanismos de controlo dos servidores que suportam a camada persistente de dados, nomeadamente o *Apache Tomcat* da *Apache Software Foundation*[24]. O menu associado ao ícone está igualmente disposto de forma semelhante face ao seu congénere do Tomcat. As opções acima descritas fundamentam-se em dois princípios fundamentais de usabilidade, especificamente a normalização de sequências de acções[25] e o princípio da localização espacial[26].

Gestão de Entidades Pretendeu-se, com a definição desta interface, fornecer o suporte a todas as actividades relativas ao domínio de dados da administração do sistema. Este domínio foi já alvo de referência na secção 3.1, consistindo, basilarmente, em duas realidades: utilizadores do sistema e fontes de dados. Desta forma, é fornecido um mecanismo visual para as operações básicas de inserção, eliminação e edição das referidas entidadades, estando cada um dos tipos armazenados em documentos distintos. De referir ainda o facto de as fontes de dados estarem categorizadas segundo uma estrutura em árvore de nível único, conduzindo à permissão da migração de fontes de dados entre as diversas categorias.

Consulta e Pesquisa Personalizada A interface gráfica de apoio à consulta e pesquisa de notícias encerra mais funcionalidades do que as referidas nos pontos

anteriores e, ao invés destas, foi desenhada para utilização por parte do agente pesquisador de notícias. São identificáveis três grandes conjuntos de actividades disponibilizadas:

- Gestão de Subscrições Permite-se ao utilizador que este realize a operação de subscrição de fontes de dados de entre as previamente disponibilizadas através do processo descrito no ponto anterior. Naturalmente, a funcionalidade simétrica encontra-se, igualmente, contemplada.
- Gestão da Organização de Fontes Este conjunto de funcionalidades possibilita que o utilizador proceda à livre organização das diferentes fontes de dados subscritas, segundo uma estrutura em árvore multinível que admite replicação de entidades.
- Pesquisa e Análise de Informação Trata-se do âmago da aplicação, prevendo a apresentação de um formulário onde é possível a especificação das palavras chave a pesquisar, quer no título, quer no corpo da notícia. É igualmente disponibilizado um filtro por data de publicação através da definição de um intervalo (aberto ou fechado) de datas. Ainda que por omissão a pesquisa seja efectuada em todas as fontes de dados subscritas pelo utilizador, é possível a escolha do conjunto de fontes a interrogar.

De modo a potenciar o grau de compreensão das entidades descritas na presente secção, dedicada às interfaces gráficas com o utilizador, aconselha-se a leitura do capítulo 4, onde, a propósito da exposição dos resultados do projecto, se apresentam ilustrações de algumas interfaces do projecto.

4 Resultados

As metas inicialmente definidas foram totalmente alcançadas, sendo que grande parte das funcionalidades são suportadas por tecnologias relacionadas com XML: eXist, XPath, XSL, XQuery e XUpdate. Deste modo, os principais requisitos funcionais implementados, agregadores dos vários conceitos do projecto, encontram-se listados de seguida:

- Acesso automatizado às fontes de dados.
- Conversão da informação semi-estruturada para um formato normalizado.
- Actualização automática dos conteúdos locais.
- Interface de administração de gestão de fontes e utilizadores totalmente funcional (ver Figura 2-A).
- Interface totalmente funcional de pesquisa de informação, gestão de subscrições e organização em árvore multinível das fontes de dados (ver Figura 2-B).

No último passo da Figura 2-B, é gerada uma interrogação em XQuery sempre que é efectuada uma pesquisa de notícias. A título de exemplo, e traduzindo a interrogação requerida pelo utilizador, apresenta-se o código correspondente através da Figura 3.

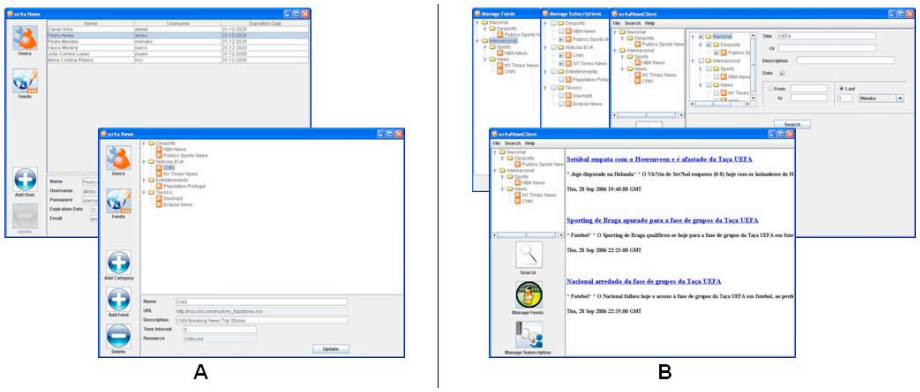

Figura 2. A-GUI de Gestão de Entidades. B-GUI de Consulta e Pesquisa Personalizada

```
xquery version "1.0";
for   $x in doc("publicodesporto.xml")/news/item
let       $listaMeses := ("Jan","01", "Feb","02","Mar","03","Apr","04","May","05","Jun","06","Jul","07","Aug","08","Sep","09","Oct","10","Nov","11","Dec","12"),
          $listaTimezones := ("Z", "GMT", "Z", "EDT", "-04:00", "EST", "-05:00", "CST", "-06:00", "CDT", "-05:00", "PST", "-08:00", "PDT", "-07:00" ),
          $data := $x/pubDate, $diasemana := substring($data, 1, 3), $dia := substring($data, 6, 2), $mesLetra := substring($data, 9, 3),
          $ano := substring($data, 13, 4), $hora := substring($data, 18, 2), $minuto := substring($data, 21, 2), $segundo := substring($data, 24, 2),
          $timezone := substring($data, 27, 3),$mes := subsequence($listaMeses , index-of($listaMeses, $mesLetra) + 1, 1),
          $timezoneValue := subsequence($listaTimezones , index-of($listaTimezones, $timezone ) + 1, 1),
          $totaldata := xs:dateTime(concat(concat(concat($ano, "-"),concat(concat($mes, "-"),concat(concat($dia, "T"),concat(concat($hora, ":"),concat(concat($minuto, ":"),$segundo))))), $timezoneValue))
where contains($x/title, "UEFA") and contains ($x/description, "") and $totaldata ge xs:dateTime("2006-09-26T00:00:01Z") and $totaldata le xs:dateTime("2006-10-17T00:00:01Z")
order by $totaldata descending
return <res><news>{$x}</news><dateZ>{$totaldata}</dateZ></res>
```

Figura 3. Exemplo de uma Interrogação de Pesquisa de Notícias

5 Conclusão e Futuros Desenvolvimentos

Tendo em conta a apresentação dos resultados efectuada no capítulo anterior, encontram-se identificados diversos temas que resultariam em desenvolvimentos futuros do projecto. A análise seguinte apresenta-se estruturada de acordo com o grau crescente de dificuldade dos componentes, e consequentemente, da previsão da data da sua implementação.

Identificada como funcionalidade de dificuldade reduzida, encontra-se classificada a análise de texto mais avançada com a introdução de pesquisa *fuzzy* e *case insensitive*. Esta funcionalidade poderia ser implementada através da utilização das respectivas funções disponibilizadas pela norma XQuery. Ainda nesta classe de desenvolvimentos, é de considerar a introdução de operações lógicas avançadas na pesquisa e múltiplos intervalos de datas, bem como a salvaguarda de esqueletos de interrogações típicas, acessíveis por todo os utilizadores. No capítulo da segurança, prevê-se o refinamento das operações de autenticação, promovendo a utilização de toda a informação respeitante aos utilizadores. Por último, neste domínio, identifica-se como futuro desenvolvimento o suporte a múltiplos idiomas e fusos horários.

Considerados como desenvolvimentos de complexidade superior, encontram-se os relacionados com operações sobre o conteúdo noticioso. De entre estes, há a destacar a análise do contexto em que os termos são empregues[27], a tradução automática de conteúdos para um conjunto pré-definido de idiomas

e a incorporação na análise do conteúdo apontado pelos hiperligação presentes na fonte de dados. Numa vertente mais relacionada com as pesquisas encetadas pelos utilizadores, importa referir a possibilidade de salvaguarda das mesmas por utilizador e a manutenção do histórico de interrogações efectuadas.

6 Agradecimentos

Neste capítulo final, os autores gostariam de agradecer ao professor Eugénio de Oliveira pelas repetidas lições respeitantes à importância do estudo e redacção de artigos científicos. Uma nota especial também para os professores João Correia Lopes e Maria Cristina Ribeiro pela atenção prestada ao trabalho dos autores aquando da sua condição de alunos da disciplina de Linguagem, Anotação e Processamento de Documentos.

Referências

1. RSS Advisory Board - `http://www.rssboard.org/`
2. Bradley, N.: The XML Companion Third Edition Addison Wesley (2001)
3. Most used RSS aggregators - `http://www.newsonfeeds.com/faq/aggregators/`
4. Mozilla Thunderbird - `http://www.mozilla.com/en-US/thunderbird/`
5. Attensa - `http://www.attensa.com/`
6. intraVnews - `http://www.intravnews.com/`
7. Atom - `http://tools.ietf.org/html/rfc4287/`
8. Mozilla Firefox - `http://www.mozilla.com/en-US/firefox/`
9. Internet Explorer 7 - `http://www.microsoft.com/windows/ie/default.mspx`
10. W3C DOM Interest Group - Document Object Model - `http://www.w3.org/DOM/`
11. SAX Project Page - `http://www.saxproject.org/`
12. XSL 1.0 Technical Report - `http://www.w3.org/TR/xsl/`
13. XQuery 1.0 Technical Report - `http://www.w3.org/TR/xquery`
14. XPath 1.0 Technical Report - `http://www.w3.org/TR/xpath`
15. XUpdate Table of Contents `http://xmldb-org.sourceforge.net/xupdate/xupdate-req.html`
16. XML:DB Initiative for XML Databases `http://xmldb-org.sourceforge.net/`
17. Bourret, Ronald: Consulting, writing, and research in XML and databases - `http://www.rpbourret.com/xml/`
18. Oracle. XML to the Power of SQL `http://www.oracle.com/technology/tech/xml/xdkhome.html`
19. PostgreSQL Project Page - `http://www.postgresql.org/about/advantages`
20. Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." Open Information Systems 10, 1 (Janeiro 1995). Capítulo 3, Página 20.
21. eXist Project Page - `http://exist.sourceforge.net/`
22. Tanenbaum, Andrew S. "Modern Operating Systems", 2nd ed, Prentice-Hall, 2001.
23. JAVA Swing Project Page - `http://java.sun.com/products/jfc/`
24. Apache Tomcat Project Page- `http://tomcat.apache.org/`
25. U.S. Department of Health and Human Services. 2006. "Research-Based Web Design & Usability Guidelines". Capítulo 2, página 11.

26. Constantine, L. L., and Lockwood, L. A. D. Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design. Reading, MA: Addison-Wesley, 1999. Capítulo 8.
27. Sarmento, Luís - Agrupamento de contexto de palavras polisémicas, Relatório Técnico 2006.

Lusco-fusco

Trabalhos em Curso

Integração de Bases de Dados Heterogéneas através de Ontologias: Um Estudo de Caso na Área de Comércio Exterior

Valéria de Mariz Sarmento Cotrim[1] and Ana Cristina Bicharra Garcia[1]

Universidade Federal Fluminense

Resumo Uma ampla, diversificada e heterogénea rede de serviços electrónicos é mantida pelo governo, baseada em um conjunto de sistemas e bancos de dados que, em grande parte, não foram projectados visando integração e interoperabilidade, muito embora isso fosse desejável para optimização de todo o processo. Muito se tem discutido sobre os problemas decorrentes da fragmentação e redundância de dados existentes na esfera governamental.
Este trabalho consiste numa investigação da viabilidade de empregar ontologias visando promover a integração entre sistemas e bases de dados heterogéneos na área de governo electrónico, aproveitando a estrutura já instalada e operacional.

Uma Aplicação para Produção de uma Estruturação HTML Automatizada na Geração de um Tutorial – Projecto CALE

João Moutinho[1], Pedro Faria[1], Diamantino Freitas[1] e Carlos Oliveira[1]

Faculdade de Engenharia da Universidade do Porto

Resumo No âmbito de um projecto POS_Conhecimento (INCLUSÃO DIGITAL) foi desenvolvido um tutorial com o objectivo de providenciar uma ferramenta de aprendizagem para uso de um leitor de ecrã destinado a Cegos ou Amblíopes, tendo em vista a utilização de computadores pessoais. Entre os seus conteúdos podemos encontrar assuntos como a própria instalação do leitor de ecrã, a sua configuração, utilização do Windows, ferramentas do Office, e até mesmo opções de scripting avançadas relacionadas com o próprio leitor de ecrã.

Do ponto de vista técnico, e tendo em conta que os conteúdos deste tutorial foram concebidos pelos dos autores da ideia, uma equipa de 4 pessoas cegas ou amblíopes, formadores experimentados na utilização do referido leitor de ecrã (*), foi requisito essencial que esta ferramenta incorporasse entre as suas funcionalidades uma forma de permitir aos autores a introdução dos conteúdos de uma forma simples e sistemática e que não levantasse problemas de acessibilidade. Como tal, desenvolveu-se um parser (baseado em php) que mediante ficheiros de texto criados pelos autores dos conteúdos, cria todo o tutorial de uma forma automática. Este motor de parsing possui os mecanismos básicos de interpretação dos conteúdos, mediante o contexto e a forma de organização dos ficheiros de texto originais. Este interpreta, constrói a hierarquia e liga os conteúdos de forma a criar os conteúdos finais do tutorial. Os conteúdos organizados e completamente criados pelo parser, estão estruturados em "mark-up" HTML para serem utilizados com um "browser", e através de Javascript foi possível criar todas as funcionalidades desejadas. Contudo, está em projecto a possível adaptação do parser para geração directa de XML, para permitir usar os conteúdos em outras interfaces que não necessariamente um browser HTML.

A ferramenta que funciona como motor do tutorial propriamente dito, é um pequeno "browser" alterado para permitir o uso de HTML e Javascript sem problemas de segurança ou "pop-ups" inacessíveis, e proporcionar condições de acessibilidade que os actuais "browsers" não permitem. Implementou-se assim uma aplicação baseada em hipertexto e com características próprias, que não necessita um período de habituação que desmotiva a iniciação do processo de aprendizagem. Quer seja um utilizador avançado ou um pouco experiente, o processo de navegação e consulta da informação pode ser feito unicamente com as teclas direccionais, juntamente com a barra de espaços. Deste modo qualquer utilizador com deficiência visual consegue enveredar pela aprendizagem da informática com a simples introdução do CD na unidade de leitura, e tudo é despoletado de uma forma automática e intuitiva. No entanto, para utilizações mais avançadas, existem combinações de teclas, e funcionalidades que permitem aumentar a eficiência da navegação.

Para além de cumprir as regras da acessibilidade da W3C, o tutorial foi também alvo de testes e constantes adaptações que permitiram optimizar as suas funcionalidades e adaptar constantemente todos os mecanismos em função da vontade e opiniões dos utilizadores cegos e amblíopes, que constituíram o painel de teste. (*) Agradecimentos aos exmos autores do tutorial Srs Daniel Serra, Domingos Ferreira, João Fernandes e Maria da Luz Ribeiro. Este trabalho foi financiado pelo Programa Operacional Sociedade do Conhecimento (POSC).

Recolha e Processamento de Informação para a Elaboração de um Dicionário Português – Chinês

Rui Pais, Francisco Carvoeira e Ferrer Gamito[1]

Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Beja, Portugal

Resumo Este artigo apresenta a parte nuclear de um sistema em desenvolvimento, para elaboração de um dicionário Português/Chinês. Esta parte irá agrupar a informação de vários dicionários disponíveis e efectuar a classificação da sua utilização conjunta, através da utilização de um Web Service e de um conjunto de métricas. Serão suportados dicionários de Português, sinónimos de Português, Português – Inglês, sinónimos de Inglês, Inglês – Chinês, Chinês e sinónimos de Chinês.

O mesmo Web Service que pretende agrupar um conjunto de funcionalidades para tratamento de línguas orientais, nomeadamente as seguintes:

- Conversão entre sistemas de codificação GB, Big5 e Unicode;
- Conversão entre caracteres chineses e sistema fonético Pinyin;
- Cálculo de um conjunto de métricas utilizadas;
- Armazenamento da informação em formato XML.

As métricas utilizadas pretendem classificar a informação armazenada por ordem de importância para, numa fase posterior, simplificar a sua correcção através da utilização de dicionários em papel.

Preservação de Bases de Dados

José Carlos Ramalho[1]

Departamento de Informática da Universidade do Minho

Resumo No contexto da preservação digital a preservação de Bases de Dados é um assunto fulcral devido à enorme quantidade de informação existente nesse suporte. Nesta apresentação será apresentada uma arquitectura de software desenvolvida com o objectivo de dar solução ao problema. O XML representa um papel central uma vez que é proposta uma linguagem de anotação DBML, que servirá de denominador comum na arquitectura e permitirá representar da mesma forma qualquer tipo de base de dados.

Using an XML Based System in Group Psychotherapy

Luís Duarte[1], Luís Carriço[1] e Marco de Sá[1]

LaSIGE & Department of Informatics, Faculty of Sciences University of Lisbon

Resumo On the course of group psychotherapy sessions, both therapists (it is common to find more than one therapist in a group psychotherapy session taking on an observer role) and patients perform various activities. Therapists deliver paper artifacts which patients have to fill-in during the sessions and then deliver them again to the former to discuss the results. Our system aims at delivering computational support to these activities by offering a set of tools and functionalities to perform them with the aid of small mobile devices, laptops and large public displays. Among the exchanged artifacts we can distinguish questionnaires, questionnaire filling-in logs, thought registries and personal notes shared by the therapists. These artifacts are generated with the aid of our XML specification, oriented to handle questionnaire and log construction and interpretation. Although the system uses an XML specification to handle data, a central relational database is also used to store it, as the system may be part of a large clinical facility.

A Communication Server is responsible to ensure the flow of communication in the system, receiving and forwarding messages from / to the correct senders / recipients. The Server is complemented by an important feature which enables session workload distribution: a message subscription system. With this module, therapists are allowed to configure which message types they want to receive or whose messages are forwarded to them. The Communication Server is therefore responsible for keeping an updated table with the current subscription settings for the session. When it receives a message, the subscription table is consulted in order to correctly forward the message to the desired recipients. The subscription table can be modified at run time during a session.

The exchanged XML messages follow a simple structure comprised by 'From', 'To', 'Session', 'Operation', 'Extra' and 'Body' fields. While the first two fields' purposes are self explanatory, the 'Session' tag refers to the identification number of the session the messages belong to. The 'Operation' tag describes the message's type (e.g. a database request). This field can be complemented by the 'Extra' tag which can offer additional information about the message (e.g. the forwarding of a questionnaire or log file). Finally, the 'Body' field contains the actual payload of the message which can be plain text or full XML questionnaires, result files or log files. Some variations of this schema can be used in setting up the subscriptions. This XML message and file based approach promotes flexibility and mobility to handle different group therapy scenarios, despite the presence of a central database.

Índice de Autores: