José C. Ramalho **Pedro R. Henriques**
Giovani R. Librelotto **Gustavo V. Arnold (Eds.)**

XATA 2003

Primeira Workshop sobre XML, Aplicações e Tecnologias Associadas
Braga, Portugal, 13 e 14 de Fevereiro de 2003.
Actas...

Universidade do Minho

Nota Editorial

A ideia de organizar uma Workshop em torno do XML já vem de longa data. Por isso, quando surgiu a ideia de organizar um encontro para o fecho do projecto de investigação METAMEDIA (projecto conjunto entre o Departamento de Informática da Universidade do Minho e a Faculdade de Engenharia da Universidade do Porto), decidiu-se logo lançar um encontro aberto a nível nacional, onde, os investigadores interessados pudessem participar enviando trabalhos para posterior apresentação e discussão de ideias.

As duas grandes áreas propostas para este encontro foram a dualidade entre XML e Bases de Dados e a Publicação de Conteúdos. Foram as áreas em que o projecto METAMEDIA se centrou. No entanto, há muitas outras apresentações em áreas como os serviços móveis, os sistemas de informação geográfica, a metainformação, o e-learning, ...

Destinado inicialmente a suportar documentos no seu sentido estrito, convencional, o XML é uma linguagem universal para representação de informação independente de ferramentas e plataformas.

Nunca é demais relembrar a motivação que nos poderá levar a utilizar o XML.

Motivação

Se se observar de perto uma empresa, um estabelecimento de ensino ou outra instituição pública ou privada, depressa se conclui que o produto de cada dia de trabalho resulta, directa ou indirectamente, na criação de documentos; cada membro de uma organização, na sua actividade diária, cria, de alguma forma, documentação. Pode ser uma carta, um memo, a planta de um edifício ou o manual de um produto. Quaisquer que sejam os documentos produzidos, eles formam parte da história da instituição e são um importante componente da sua memória corporativa, contendo um manancial valiosíssimo de informação.

Algumas formas de informação são altamente estruturadas. De facto, algumas são tão estruturadas, que podem ser representadas numa forma tabular. Inventários, tabelas de preços, dados sobre empregados, são exemplos deste tipo de informação. Hoje em dia, a maioria dos sistemas de gestão de empresas são desenhados para lidar com informação relacional, muito estruturada (Folhas de Cálculo – FC, Bases de Dados – BDs). Mas, surpreendentemente, estima-se que esta informação representa apenas 10 % do total de informação disponível numa empresa. Assim surgem as seguintes questões: que tipo de informação representam os outros 90 %? Como se poderá tirar partido dela?

Estes 90 % da informação correspondem a textos que são produzidos e circulam dentro da instituição ou entre instituições. É difícil ou mesmo impossível aplicar as metodologias relacionais a texto. Pode-se então colocar a questão: haverá alguma maneira de aceder ao potencial da informação mantida num formato textual? Se ele se mantiver numa forma simples, puramente sequencial, a solução parece difícil; a resposta recai, mais uma vez, na *estruturação* desses textos [Ken96].

Documentos estruturados são documentos que têm uma estrutura explícita, as suas componentes estão identificadas. Se esta estrutura for definida formalmente, identificando as componentes e a maneira de, a partir delas, se construir o documento, torna-se possível estipular um conjunto de regras para a criação desses documentos. Por exemplo, as regras para um determinado manual podem estipular que o documento terá uma página de rosto, um índice e um prefácio; esta parte inicial deverá ser continuada por, no mínimo, quatro capítulos; cada capítulo deverá ter um título seguido de texto, possivelmente dividido em secções. Essas secções poderão ter também uma estrutura definida e assim recursivamente para todos os subelementos de cada secção.

Produzidos de acordo com as regras da empresa ou não, a verdade é que os documentos, na sua maioria, têm uma estrutura (visível ou escondida), que pode ser formalizada.

Este problema, de tentar manter viva a informação correspondente aos 90 % do património de uma instituição, fornece a primeira motivação para a utilização da tecnologia de anotação (XML, XSL).

O segundo mote vem de uma área relacionada: a publicação electrónica.

Nas últimas décadas, a publicação electrónica sofreu uma enorme evolução. O avançar da informática e das tecnologias a ela associadas tem levado a uma substituição gradual e efectiva do papel pelo suporte digital, magnético ou óptico (disquete, disco de computador ou CD-ROM).

Nos últimos anos, a explosão da Web (cada vez mais fácil e amplamente acessível) veio acelerar e aumentar ainda mais a produção documental em suporte digital, consolidando novos tipos e arquitecturas de informação: o hipertexto e a hipermédia [DD94].

Esta evolução trouxe com ela vários problemas e veio agravar outros já existentes. O mais grave foi, e é, o da proliferação de formatos para manipulação de documentos, muitos deles privados e incompatíveis entre si. Com a Web e a banalização da produção de CD-ROM, para além dos vários sistemas tradicionais de arquivo e processamento de texto, os utilizadores passam a precisar de manter os seus documentos em mais formatos e suportes diferentes. Um documento leva tempo a produzir, consome espaço de arquivo pelo que, se a sua reutilização não for possível, o custo da sua produção e a distribuição em formas variadas aumentará ainda mais. Por estas razões, a produção documental deve ser inteligente, de modo a que os documentos produzidos se mantenham vivos (ao fim de vários anos, em diferentes sistemas e depois de várias versões do *software* que os produziu); é necessário que a sua reutilização, para os mais variados fins, seja possível!

Outro problema prende-se com a globalização da informação. Hoje a Web é praticamente acessível a todos, tendo-se tornado um veículo apetecível para quem disponibiliza e para quem consome informação. Como resultado, temos uma enormíssima proliferação de documentos acessíveis. Mas para que tal quantidade seja realmente útil, é necessário que existam maneiras eficazes de os agrupar e de pesquisar quais deles fornecem informação sobre determinado assunto. Por isso, torna-se praticamente inviável colocar documentos na Web sem qualquer descrição que facilite a sua catalogação. Os motores de pesquisa e de indexação teriam um trabalho infinito para procurar e encontrar fosse o que fosse. Neste contexto, surgiram projectos como o *Dublin Core* [Hee96] ou o *Text Encoding Initiative – TEI* [SMB94] que visam resolver estes problemas acrescentando metainformação aos documentos. Metainformação é informação sobre informação [Cap95], neste caso sobre documentos. Mais especificamente, trata-se de uma espécie de registo bibliográfico que se agrega a um documento de modo a disponibilizar facilmente informação como a data da sua criação, quem são os seus autores e até mesmo, uma memória descritiva do seu conteúdo.

Recentemente, uma das áreas que tem registado significativos contributos e que tem evoluído muito é a da representação abstracta de documentos ou de objectos com a forma de documento. Aqui os problemas começam logo pela definição de documento. Para algumas pessoas um documento é apenas um registo textual, enquanto que, para outras, pode ser muita coisa desde texto até um ser vivo.

O esforço para normalizar este conceito tem sido enorme e normas como o SGML [Her94], o Hytime [New97], o XML [Wil01] e propostas como o DOM [DOM] e o RDF [Bra97] estão a tornar-se familiares para muita gente e estão a ser seguidos e implementados pela indústria.

Em suma, a Web foi a grande responsável pelas recentes evoluções registadas na publicação electrónica e nos conceitos de documento puramente textual *versus* multimédia e monolítico *versus* estruturado.

Estrutura da Workshop

A Workshop encontra-se dividida em oito sessões temáticas:

s1 – Esta é a sessão de abertura, onde além duma palestra convidada, será apresentado o projecto METAMEDIA e uma perspectiva do ensino do XML nas licenciaturas em informática.

s2 e s6 – São as sessões dedicadas à dualidade do XML e Bases de Dados.

s3 e s7 – São as sessões dedicadas à publicação electrónica de conteúdos e aplicações do XML a casos reais.

s4 – Debaixo do tema ”Tecnologias e WebServices”, colocamos um conjunto de apresentações que retratam pormenores técnicos não directamente relacionados com nenhum dos outros tópicos. Irá ser a sessão com mais variedade nos assuntos a discutir.

s5 – Nesta sessão, apresentam-se várias linguagens XML desenvolvidas pelos autores com objectivos aplicacionais específicos.

s8 – Por último, temos a sessão sobre metainformação. Um dos temas mais em foco actualmente quando se tenta levar a Web para a dimensão seguinte: ”Semantic Web”.

É nossa convicção de que esta workshop será um ponto de encontro entre as várias comunidades que trabalham com XML em Portugal, e de que algo de novo nascerá desta iniciatica.

José Carlos Ramalho

Referências

[Bra97] Tim Bray. Rdf and metadata. Technical report, Seybold and O'Reilly Publications, 1997.

[Cap95] Priscilla Caplan. You call it corn, we call it syntax-independent metadata for document-like objects. *The Public-Access Computer Systems Review 6*, 4, 1995.

[DD94] Steven DeRose and David Durand. *Making Hypermedia Work*. Kluwer Academic Publishers, 1994.

[DOM] Document object model (dom). http://www.w3c.org/DOM/.

[Hee96] Rachel Heery. *Review of Metadata Formats*, volume 30, pages 345–373. 1996.

[Her94] Eric Herwijnen. *Practical SGML*. Kluwer Academic Publishers, 1994.

[Ken96] Dianne Kennedy. Tales from the front understanding structured documents. *¡TAG¿ The SGML Newsletter*, 1996.

[New97] Steven Newcomb. Document architectures: the new hytime. International SGML Users Group (ISUG) Newsletter, 1997.

[SMB94] C. M. Sperberg-McQueen and Lour Burnard. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Association for Computers and the Humanities, Association for Computational Linguistics, Association for Literary and Linguistic Computing, 1994.

[Wil01] Heather Williamson. *XML: the complete reference*. Osborne/McGraw-Hill, 2001.

Organização da Conferência

Presidente da Comissão Organizadora

José Carlos Ramalho (Universidade do Minho, Braga)

Comitê de Programa

Gabriel David (Metamedia - INESC/Porto)
Pedro Henriques (Metamedia - DI/UM)
Cristina Ribeiro (Metamedia - INESC/Porto)
José Carlos Ramalho (Metamedia - DI/UM)
Marta Jacinto (ITIJ - DI/UM)

Comissão Organizadora

Gabriel David (Metamedia - INESC/Porto)
Pedro Henriques (Metamedia - DI/UM)
Cristina Ribeiro (Metamedia - INESC/Porto)
José Carlos Ramalho (Metamedia - DI/UM)
Ademar Aguiar (Metamedia - INESC/Porto)
Gustavo Arnold (Metamedia - DI/UM)
Catalin Calistru (Metamedia - INESC/Porto)
Giovani Librelotto (Metamedia - DI/UM)
Ricardo Martins (Metamedia - DI/UM)
Carla Oliveira (CI/UM)

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Índice

Actualização de Dados de GIS usando Assistentes Digitais Pessoais - Henrique Silva, ISMAI; Alexandre Valente de Sousa, ISMAI; João Correia Lopes, FEUP-UP;

Engenharia reversa de HTML usando tecnologia XML - José João Almeida, DI-UM; Alberto Simões, DI-UM;

Especificação XML de Aplicações para WWW - Alexandre Martins, DI-UM, FEUP-UP; Pedro Rangel Henriques, DI-UM; Gabriel David, FEUP - UP;

Xexam - uma linguagem de suporte para exames online (e-learning) - Daniel Edgar Pinto Soares , DI-UM; Maria da Conceição Vieira Mota, DI-UM; José Carlos Ramalho, DI-UM;

XML Templates for Constraints (XTC), Um Nível de Abstracção para Linguagens de Especificação de Restrições - Marta Henriques Jacinto, DI-UM, ITIJ-MJ; Pedro Rangel Henriques, DI-UM; José Carlos Ramalho, DI-UM;

IXDIRQL: Uma linguagem interactiva de perguntas para o XML - Alda Lopes Gançarski, LIP6 - Universidade Pierre et Marie Curie Paris 6, Paris, França;

XML Topic Map Builder: Specification and Generation — Giovani Rubert Librelotto, DI-UM; José Carlos Ramalho, DI-UM; Pedro Rangel Henriques, DI-UM;

HaQuery: A Haskell Combinator Library for Querying XML\\ Extended Abstract - João Saraiva, DI-UM; David Lopes, DI-UM; António Faria, DI-UM;

FrameDoCMS _ um sistema de gestão de conteúdos para documentação de frameworks baseado em XML e WikiWikiWeb - Ademar Aguiar, FEUP - UP; Gabriel David, FEUP - UP;

XML na Modelação de Sistemas Hipermédia - Luís Carriço, DI-FC-UL; Rui Lopes, DI-FC-UL; Miguel Rodrigues, DI-FC-UL; Amadeu Dias, DI-FC-UL;

Geração de Conteúdos para Plataformas Móveis - Filipe Figueiredo Correia, ParadigmaXis; Joel Varanda, ParadigmaXis; João Correia Lopes, FEUP-UP; Alexandre Valente de Sousa, ISMAI;

Catalogação e distribuição de Meta-Informação Geográfica em XML]- Jorge Gustavo Rocha, DI-UM; André Nuno Faria, DI-UM; Paulo Brito, DI-UM;

Partilha de Conteúdos e Semântica Multimédia - José Manuel Torres, INESC-Porto;

Desafios e contribuições resultantes da utilização da XML na norma ISO/MPEG-7 e suas aplicações - Rui J. Lopes, Comp.Dep.-Lancaster University; Adam T. Lindsay, Comp.Dep.-Lancaster University; David Hutchison, Comp.Dep.-Lancaster University;

Omnipaper- Descrição de recursos de noticias digitais em RDF - Teresa Susana Mendes Pereira, DSI-UM; Ana Alice Baptista, DSI-UM;

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

13 Fev.	S1(9.00h) Abertura da Workshop	[*Metamedia*] - **Cristina Ribeiro**, FEUP; [*Estudio Comparativo entre las Bases de Datos Relacionales y XML, y una Nueva Aportación a la Problemática Existente*] - **Ana Fermoso García**, Escuela Universitaria de Informática - Universidad Pontificia de Salamanca ; [*O XML no curricula das licenciaturas em informática*] - **José Carlos Ramalho**, DI-UM;

MetaMedia 2 - Meta-informação na Preservação e Pesquisa de Componentes Multimédia

Gabriel David e Cristina Ribeiro
FEUP / INESC Porto
Universidade do Porto
Porto, Portugal
{gtd@fe.up.pt, mcr@fe.up.pt}

Palavras-chave: Meta-informação, arquivos multimédia, recuperação de informação, linguagens de anotação.

Resumo

Os arquivos multimédia, tanto no sentido de repositórios locais como, numa perspectiva mais geral, no de fracções da Internet, têm aumentado em importância e disponibilidade. Contudo, são geralmente reconhecidas as dificuldades em produzir pesquisas de resultados precisos e de confiança. A adição de meta-informação [1] aos objectos de arquivo viabiliza pesquisas informadas e eficientes, desde que aquela inclua descrições de conteúdo e de contexto. Tem-se realizado uma pesquisa intensa quer sobre a normalização da descrição de conteúdos multimédia [2,3] quer na extensão da descrição de contexto tradicional ao mundo dos documentos electrónicos [4].

O objectivo principal deste projecto é aprofundar a investigação da combinação de ambas as abordagens, produzindo um modelo para a descrição arquivística multimédia e ferramentas que suportem um ambiente correspondente.

O conceito de objecto multimédia é aqui estendido para abranger componentes de software. Os objectos multimédia já não são apenas itens passivos mas, na senda da programação orientada por objectos, incluem frequentemente o código que explicita o seu comportamento. A preservação destes objectos constitui um grande desafio para os arquivos. Considerando a sua taxa de produção, o grosso da descrição dos objectos multimédia deve ser gerado em tempo de criação. Isto exige que as ferramentas de produção sigam as normas mais do que estas serem aplicadas apenas a posteriori, na catalogação. Detecta-se aqui um paralelo com as boas práticas de desenvolvimento de software, que requerem que a documentação seja produzida ao longo do processo. As técnicas usadas na documentação automática de componentes de software podem portanto ser valiosas na geração de descrições de objectos multimédia. Ao contrário, também os requisitos de arquivos podem ser explicitamente tratados pelas ferramentas de desenvolvimento de software.

Os resultados do projecto incluem um modelo (Esquema de Descrição na terminologia MPEG-7) para descrição contextual e de conteúdo de arquivos multimédia estendidos, contemplando aspectos relevantes para os componentes de software. As implicações desta extensão na linguagem de descrição serão estudadas e derivar-se-ão ferramentas como analisadores, editores dirigidos por sintaxe e geradores de saídas, as quais constituem a base de um ambiente para arquivos multimédia. Para armazenar as complexas estruturas de dados por elas produzidas, utilizar-se-á um modelo de base de dados que generaliza o já desenvolvido no projecto Metamedia. As ferramentas, a base de dados e uma interface de utilizador adequada serão empacotadas num "software framework" [5] a instanciar para aplicações concretas.

Como domínio de aplicação seleccionou-se a produção audiovisual. O arquivo de produção-AV será conforme ao MPEG-7 e servirá de banco de ensaios para estratégias de pesquisa inteligente que tirem partido da natureza multifacetada da meta-informação.

Referências

[1] Metadata - Pathways to Digital Information, Murtha Baca, ed., Getty Information Institute, 1998.
[2] MPEG Requirements Group, "MPEG-7 Context and Objectives", Doc. ISO/MPEG N2207, MPEG Tokyo Meeting, March 1998
[3] MPEG Requirements Group, "MPEG-7 Requirements Document V.5", Doc. ISO/MPEG N2208, MPEG Tokyo Meeting, March 1998
[4] Sperberg-McQueen, C.M. and Burnard, Lou. "Guidelines for Electronic Text Encoding and Interchange (TEI P3)". Published by [Association for Computers and the Humanities, Association for Computational Linguistics, Association for Literary and Linguistic Computing] ,1994.
[5] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns-Elements of reusable object-oriented software, Addison-Wesley 1995.

Estudio Comparativo entre las Bases de Datos Relacionales y XML. Nueva Aportación a la Problemática Existente

Ana Fermoso García
Universidad Pontificia de Salamanca, Facultad de Informática
Salamanca (España)
afermoso@upsa.es

María José Gil Larrea, Jesús Díaz Labrador
Universidad de Deusto, Facultad de Ingeniería (E.S.I.D.E.)
Bilbao (España)
marijose@eside.deusto.es, josuka@eside.deusto.es

Resumen

Actualmente gran cantidad de información está almacenada en Bases de Datos Relacionales, pero otra gran cantidad lo esta en otros formatos, entre los que destaca XML, que se está convirtiendo en el estándar más utilizado para el intercambio de datos en el mundo de la web y los negocios a través de Internet.

Se hace imprescindible trabajar conjuntamente con diversos formatos de información, lo que conlleva una serie de problemas y limitaciones. Para paliarlos una posible solución podría ser un sistema que sirviese para consultar de forma transparente varios formatos, en particular entre SQL y XML, un Modelo de Consulta Integrada. Además, por el número creciente de aplicaciones que exige que la información tenga que ser tratada en un entorno Web, el sistema podría utilizar un lenguaje de consulta basado en XML, y que las respuestas a estas consultas también se devolviesen en el mismo formato XML.

En este trabajo, a partir de un detallado estudio de distintos sistemas existentes en los que se relacionan las Bases de Datos y XML, se estudian las limitaciones de dichos sistemas y se plantean las características que debería tener uno que trate de solventar dichas limitaciones.

Palabras clave: XML, Bases de Datos Relacionales, Consulta e Integración de datos en el entorno Web, Lenguajes de Consulta en XML.

1. Introducción

La historia de XML está íntimamente ligada a la evolución del World Wide Web [Bray 1998], de hecho XML se ha convertido en el estándar de facto para el intercambio y representación de datos en el mundo los negocios a través de Internet: comercio electrónico, aplicaciones Bussines to Bussines (B2B), Bussines to Client (B2C) y el resto de variantes.

Por otra parte, a la hora de manejar grandes volúmenes de información, siguen siendo los Sistemas de Gestión de Base de Datos la herramienta más utilizada, y entro ellos el modelo más extendido sigue siendo el Relacional, aunque el modelo Orientado a Objetos vaya tomando fuerza día a día.

En consecuencia, resultaría muy útil poder manejar las Bases de Datos Relacionales, a través de XML, es decir, que podamos consultar las BD a través de un lenguaje basado XML y también utilizar este lenguaje como interfaz para pasar la información entre diferentes bases de datos o sistemas en general, que interaccionan entre sí.

En algunos casos, se puede pensar incluso en la sustitución de las bases de datos por XML como almacén de datos. Esto sólo resultaría útil en ciertas circunstancias [Morrison et al. 2000], por ejemplo cuando la representación de la información sea compleja, e incluso variante, o cuando los datos no sufren muchos procesos de actualización.

En todos los casos anteriores puede resultar ser más eficiente usar XML que una base de datos relacional para almacenar la información, pero en el resto de situaciones sigue siendo mejor continuar con el almacenamiento en bases de datos.

Por otra parte, en ciertos entornos se está convirtiendo en habitual tener que manejar dentro del mismo negocio, tanto datos almacenados en las bases de datos tradicionales, como en documentos XML. Esto sucede por ejemplo en aplicaciones B2B en las que, por un lado, hay que implementar procesos de negocio con la información almacenada en una base de datos relacional , y por otro, se tienen que integrar aplicaciones y negocios con los de otras empresas. Para asegurar la interoperatibilidad entre ellas, por ejemplo por que tengan que intercambiar información entre sí siguiendo unos determinados estándares, se suele utilizar XML.

En consecuencia, se trata de establecer un método que sirva como sistema de consulta tanto para información almacenada en cualquier base de datos relacional, independientemente de su complejidad, tamaño y área o tema al que pertenezca la información, como para información almacenada en XML; es decir, que permita consultar información de la misma forma, independientemente del formato de almacenamiento real de esos datos. Para el usuario final este proceso debe ser transparente, como si sólo estuviera accediendo a información en XML.

Una de las premisas fundamentales del método que se propone, y que lo diferencia de otros sistemas de este tipo, es que los datos seguirán almacenados en su formato original, es decir, en las bases de datos o en XML, según donde estuvieran, no siendo necesario migrar a XML, en el caso de las bases de datos, para poder ser consultados, independientemente de que los resultados de las consultas, e incluso las consultas, si que se hagan en XML. En el caso de las bases de datos relacionales, el método tiene otra ventaja, como al final las consultas se hacen directamente sobre la base de datos y en el lenguaje de consulta tradicional SQL, aunque todo esto resulte transparente al usuario, no se pierde eficiencia ni velocidad a la hora del acceso, ya que se seguirán utilizando para ello los motores de búsqueda de la propia base de datos.

Existen muchos trabajos que tratan de relacionar bases de datos relacionales y XML, y más concretamente, la consulta de información en uno u otro formato. El sistema planteado se puede utilizar también para consultar documentos XML. Esto último también es una novedad, pues la mayor parte de estos sistemas de consulta a través de XML, o servirán para consultar base de datos, previo volcado al formato XML, o para consultar datos en XML directamente, pero no para ambas al mismo tiempo y con la misma filosofía.

Lo importante será por tanto establecer las bases teóricas y conceptos fundamentales que permitan crear el modelo de integración en el que se apoye el sistema de consulta, este será el primer objetivo. Una vez logrado, se sentarán las bases para especificar el proceso de traducción de las consultas de usuario al lenguaje de la base de datos.

2. Estado del Arte: XML y las Bases de Datos Relacionales

En casi todos los trabajos e investigaciones que relacionan las bases de datos Relacionales con XML se pretende hacer accesibles los datos de la base de datos, así como transferirlos a otros sistemas, a través del formato XML, pero en casi ninguno se permite que además el mismo método haga también accesibles los datos que ya están en XML. Algunos como Shanmugasundaram [Shanmugasundaram et al. 1999] hacen justo lo contrario, transformar documentos XML a bases de datos para luego hacer la consulta sobre esa información en un lenguaje como SQL, propio de las bases de datos relacionales.

En otros como el denominado Monet [Schimidt et al. 2000], se plantea un modelo de datos y ejecución para permitir un almacenamiento y devolución eficiente de documentos XML en una base de datos relacional.

Los documentos XML se representan en forma arborescente y la cuestión clave es cómo almacenar los datos de este árbol como una instancia de base de datos que suministre unas capacidades de acceso eficientes. Para conseguirlo se aplican una serie de reglas que permiten almacenar los datos del árbol (ramas o enlaces, nodos y valores finales del documento XML, que están en las hojas del árbol) en forma de tablas de la base de datos, y luego sobre estas tablas se hacen consultas normales en SQL, que devuelven como resultado la información de los documentos XML consultados.

En X-Ray [Kappel 2000] lo que se plantea es un modelo o metaesquema que permite tanto pasar datos almacenados en una base de datos relacional a documentos XML que siguen un determinado Data Type Definition (DTD), como lo contrario, que datos almacenados en XML siguiendo un determinado formato o DTD, se almacenen en una base de datos relacional, es decir, sirven para migrar datos de XML a base de datos como antes, y al revés. El metaesquema

de X-Ray está formado por tres componentes, el *DBSchema*, que contiene la información del esquema de la BD, el *XMLDTD*, que contiene la información del DTD de los documentos XML y el *XMLDBSchemaMapping* (esquema de correspondencias), que contiene al información para migrar la base de datos a un documento XML y viceversa. Sin embargo, esta migración de la base de datos no tiene porque ser sólo a un determinado DTD, es decir, a un determinado tipo de documento XML, pero sí que tiene que existir un esquema de correspondencias y un esquema con la información del DTD, *XMLDTD*, para cada DTD distinto. Por otra parte, también puede ocurrir que quieran convertirse diferentes bases de datos, cada una con su esquema, al mismo DTD, por ejemplo porque pertenecen al mismo dominio de información, en este caso también es necesario un *DBSchema* y un esquema de correspondencias, para cada base de datos.

El *DBSchema* contiene información sobre cada una de las relaciones de la base de datos, *DBRelations*, de los atributos que las componen, *DBAttributes* y sobre conexiones entre relaciones, *DBRelationShip*.

El *XMLDTD* guarda información sobre el DTD al que se quiere migrar a través de sus elementos, *XMLElementType*, y los atributos de estos elemento, *XMLAttributes*. Los elementos pueden ser simples, compuestos o vacíos.

Finalmente, el *XMLDBSchemaMapping*, establece las correspondencias entre los dos esquemas anteriores, tal que por ejemplo, un atributo de la base de datos puede estar asociado a un elemento simple, a otro atributo, o con el contenido de un elemento compuesto, del documento XML resultante.

En este modelo la correspondencia entre XML y la base de datos relacional, gira en torno al establecimiento de una relación base, que es la que se va a tomar como raíz del árbol, y que va a condicionar el cómo se ve el resto de la base de datos desde esta relación, y por tanto esto supondrá una limitación.

En general XML es ante todo una forma de almacenar y modelar datos. Así Peter McBrien y Alexandra Polovassilis [McBrien y Poulovassilis (1) 1999] [McBrien y Poulovassilis (2) 1999] [McBrien y Poulovassilis 2001] en trabajos relacionados con el tema, utilizan un formato gráfico especial denominado *HDM (Hypergraph Data Model)* formado por tres elementos: nodos, arcos y restricciones. El manejo de este grafo y sus elementos lo hacen a través de una serie de primitivas que permiten consultar, añadir o eliminar elementos de dicho gráfico.

Estableciendo la correspondencia entre los elementos de cualquier modelo de datos (DER, UML, Modelo Relacional, ...) y los elementos de *HDM*, se pueden modificar los datos; es decir,

se puede pasar de unos modelos a otros usando HDM como intermediario. Esto supone, que también a través e *HDM* se puede pasar del Modelo XML al Relacional y viceversa. Para ello parte de un esquema inicial de un determinado modelo de datos, y se quiere obtener otro distinto en otro modelo, y siempre que ambos pertenezcan al mismo universo de datos. Esto se consigue renombrando nodos y arcos y añadiendo y/o borrando nodos y arcos del grafo origen para obtener él de destino. Una vez pasado de un modelo de datos a otros, o incluso estando en el mismo modelo, por ejemplo porque lo que se quiere es adaptar la información de un esquema a otro del mismo universo y modelo de datos, se trata de migrar todas las operaciones, por ejemplo las consultas o actualizaciones, que se hacían sobre el sistema original y que queremos que se sigan haciendo sobre el destino.

Christophides y Cluet [Christophides y Cluet 2000] proponen utilizar XML como herramienta de integración en un sistema de conversión de datos semiestructurados denominado Yat, sobre los que utilizan un lenguaje de consulta especial, el YatL.

Este sistema integrador Yat, representa la información a varios niveles: (Modelo, Esquema y Datos) y está formado por unos wrappers y mediadores que se comunican a través de estructuras, datos y operaciones que utilizan una sintaxis XML.

Las consultas hechas en lenguaje YatL utilizan tres tipos de sentencias: *MATCH*, que asocia una variable a cada campo que participa en el filtro de la consulta, *MAKE*, que construye el resultado y *WHERE*, que añade las condiciones.

Además añaden al sistema una serie de funciones algebraicas en XML, lo que se conoce como el álgebra Yat, que se suman a las operaciones de álgebra relacional tradicionales, como *Select* y *Join,*que siguen soportándose igual. Estas tres nuevas operaciones algebraicas son:

- *BIND:* obtiene la estructura de las tablas que se representa en forma arborescente.
- *TREE*: selecciona ciertos subárboles de la estructura obtenida anteriormente.
- *Funciones Skolem*: sirve para crear nuevos identificadores, una especie de campos calculados

Aunque la representación de las operaciones es arborescente, también se incluyen herramientas para ejecutar consultas tanto en YatL, como en XML.

SINGAPORE [Dittrich y Domenig 1999] [Dittrich y Jonscher 2000] es otro sistema que va más allá de las bases de datos relacionales, al proponer un método de integración y consulta tanto para datos estructurados de distintos tipos, bases de datos relacionales u orientadas a objetos, como para datos semiestructurados. Para conseguir esta integración usa una estructura especial

denominada mediador, donde se implementa un modelo de datos global que permite homogeneizar toda la información. La consulta se hace en un lenguaje especial, SOQL, basado en SQL, pero más potente. Para crear el metaesquema en el que guardar la información de cualquier fuente de datos que se desee integrar en el sistema, se utilizan conceptos ontológicos, estudio de perfiles de usuario, etc.

SilkRoute [Fernández y Tan 2000] es una herramienta que permite consultar en XML una base de datos, con un lenguaje propio, el RXL.

Con la primera consulta RXL se crea una vista de parte de la base de datos, sólo de la afectada por esa consulta, con los datos ya en XML. Es decir, a XML no se pasan todos los datos de la base de datos, sino sólo los que interesan. Esto limita las posteriores consultas de usuario a esa vista virtual de la base de datos (*XML View*).

La arquitectura y procesamiento de una consulta en este sistema, se podría resumir de la siguiente forma:

Con la consulta RXL se crea la *XMLView* de la base de datos, que es lo que ve el usuario. Las consultas siguientes se hacen en XML-QL, se pasan al "Componedor de Consultas", que en función de la *XMLView*, obtiene la "Consulta Ejecutable" o compuesta, otra vez en RXL. Esta consulta será la combinación de la consulta inicial RXL que da lugar a la *XMLView* y la emitida por el usuario en XML-QL. Una consulta RXL tiene una parte *from* y otra *where* como las de SQL y una parte *construct* como la de XML-QL.

Finalmente, la consulta ejecutable es convertida, por el "Traductor", a una o más sentencias SQL que son ejecutados sobre la base de datos real por el motor de BD y los resultados obtenidos se traducen a XML por el "Generador XML"

M. Carry, D. Florescu et al. [Carry et al. 2000] en su proyecto XPERANTO permiten consultar datos almacenados en bases de datos Objeto-Relacionales en un lenguaje de consulta XML y devuelven los resultados también en XML, es decir, todo se hace en un formato XML puro.

El sistema se subdivide en una serie de componentes:

- Traductor de consultas, que es el componente más importante.
- Traductor de los resultados de la consulta de la base de datos a XML.
- Generador del esquema XML (XMLSchema), a partir del esquema de la base de datos.
- Servicios de la XML-View, que nos ayudan a obtener el XQGM (Modelo gráfico de consulta en XML) a partir de una consulta en XML-QL y la información del XMLSchema.

El Traductor de Consulta, el componente más importante, a su vez se divide en las siguientes partes: el parser XML-QL, que recibe la consulta XML-QL y obtiene el gráfico XQGM a partir de la información del XMLView. El Optimizador de la Consulta, que obtiene el XQGM ya optimizado para la consulta concreta realizada. El Traductor a SQL, que traduce a SQL la consulta reflejada en el XQGM que se recibe.

Una base de datos Objeto-Relacional se caracteriza porque además de los conceptos típicos de una base de datos Relacional como tablas, vistas, esquemas, columnas, tipos básicos de datos, añade otros conceptos propios de la Orientación a Objetos como herencias, referencias, "tipos" de tablas y vistas, ...

El proceso de traducción del esquema de la base de datos a un XMLSchema de XML, que sería el XMLView de esa base de datos, sería el siguiente:

- Los tipos estructurados de la base de datos objeto-relacional se convierten a "ComplextypeElement" en el XMLSchema.
- Las tablas normales de la BD se convierten a "element" en el XMLSchema.
- Para determinar que una tabla es un conjunto de elementos de un tipo estructurado, también se usa un "ComplexDataType", diciendo que esa tabla esta formada por un conjunto de ocurrencias (maxoccurs='*') del tipo estructurado.

Para pasar la consulta realizada de XML-QL a SQL se usa como elemento intermedio el gráfico XQGM, que facilita la traducción a SQL al optimizar el XML-View en función de la consulta XML-QL concreta realizada.

Este gráfico se parece al Modelo Gráfico de Consulta (QGM) del sistema de base de datos objeto-relacional de DB2 UDB, aunque se puede usar luego en otros sistemas. EL gráfico optimiza la información del XML-View para adaptarlo a la consulta XML-QL realizada, para lo cual se elimina del XML-View los elementos o atributos que no van a aparecer en el resultado de la consulta porque no se piden en la consulta real realizada.

Finalmente, para pasar a SQL la consulta final, y devolver los resultados en el orden que interesa para XML, cada subconsulta WHERE de la consulta XML-QL, se convierte a una subconsulta SQL y el resultado es la UNION de todas las subconsultas SQL. Para que quede clasificado como interesa en el XML-Result, se usa el ORDER-BY.

El orden de las subconsultas es desde la tabla principal de la consulta, a las secundarias, usando como claves las que se han obtenido en las tuplas devueltas de la tabla principal.

Además de los trabajos mencionados también han aparecido otros que desarrollan una serie de lenguajes de consulta, que se van convirtiendo más o menos en estándares según el éxito de cada uno, y que sirven para acceder a información que ya está en XML, aunque originalmente puede provenir por ejemplo de una base de datos relacional.

Maier presenta en [Maier 1998] algunos de los requisitos que debería tener cualquier lenguaje de consulta para XML. Entre ellos destacan: la presentación del resultado en XML, la posibilidad de seleccionar, extraer, combinar y transformar el contenido del documento XML (en este caso basta con que lo pueda consultar, no es necesario que también lo pueda modificar), la capacidad de actuar sobre datos sin esquema predefinido (en el modelo planteado si se va a obligar a que el documento este validado contra un determinado DTD o Esquema) y la posibilidad de que los resultados sean representados en XML.

M. Fernández, J. Simeon y P. Wadler [Fernández et al. 1999] también presentan las características que debería tener un lenguaje de este tipo y además ponen diversos ejemplos y comparan algunos de dichos lenguajes entre sí. Algunos de estos lenguajes proceden del entorno de BD y tienen más similitudes con otros lenguajes de este entorno como SQL. Dentro de este tipo está Lorel, XML-QL, YatL o Quilt. Otros lenguajes de consulta proceden del entorno de XML, como XSL, XQL o XQuery, y otros son lenguajes gráficos como XML-GL [Ceri et al. 1999].

XQuery [DeRose 1998] es un lenguaje que tiene influencias de lenguajes procedentes del entorno de BD como Quilt, que a su vez también está influenciado por YatL y Lorel, y del propio SQL, utilizando el típico patrón de sentencia SELECT-FROM-WHERE. Además está influenciado también por otros lenguajes de consulta del entorno XML, como XQL y XPath para la sintaxis de caminos jerárquicos, y de XML-QL para crear variables de enlace. Por todos estos motivos podría llegar a convertirse en el lenguaje estándar de consulta XML, el problema es que, a día de hoy, todavía sigue siendo un lenguaje de laboratorio que se sigue refinando y para el que aún no existen herramientas de implementación.

Uno de los lenguajes más utilizados es Lorel[Abiteboul et al. 2000] [McHugh 1997] [Abiteboul et al. 1997], que tiene influencias de OQL, un lenguaje de consulta para bases de datos orientadas a objetos y añade nuevas características como la búsqueda por aproximación (exclusivas de documentos semiestructurados como los de XML). Lorel usa los *Dataguide* imitando al esquema de una base de datos.

XML-QL [Deutsch 1999] es otro lenguaje proveniente del entorno de las bases de datos, también muy utilizado como Lorel. Una consulta consiste en una cláusula *WHERE*, que indica

lo que se quiere seleccionar, y una cláusula *CONSTRUCT*, que especifica el resultado a devolver.

Quilt es otro lenguaje de consulta a documentos XML que suelen contener información procedente de base de datos relacionales, y con influencias de otros lenguajes como Lorel, Xpath, ... En [Manolescu et al. 2001] se dan las pautas para convertir una consulta en Quilt sobre un documento XML a SQL, y que sea ejecutada ya sobre la base de datos asociada al documento XML consultado. Sin embargo, en este trabajo no se da ninguna solución al proceso previo de traducción de la información de la base de datos relacional a XML. Para conseguirlo se podría usar cualquiera de los métodos propuestos para lograr esta traducción, por ejemplo SilkRoute, si sólo se quiere tener una vista de la base de datos, o X-Ray si se quiere pasar toda la base de datos a XML. En cualquier caso, se sigue partiendo de que la consulta se hace sobre datos ya en XML.

Los lenguajes XSL y XQL [Robie 1999], que proceden del entorno XML, se basan en la utilización de caminos o *path* para indicar lo que se busca, y filtros para establecer las condiciones de la búsqueda.

Otro método de acceso a las bases de datos relacionales es a través de páginas web con ASP (de Microsoft), JSP (en JAVA) o PHP (las más genéricas). Todas siguen la misma filosofía aunque utilicen distintos lenguajes de implementación. Este método permite consultar las BD a través de SQL, el lenguaje habitual de consulta de las bases de datos, y después convierten a XML los resultados devueltos en un "recordset", pero no permiten hacer consultas en lenguaje XML, ni a documentos XML.

Por otro lado, algunos de los sistemas gestores de base de datos más importantes como Microsoft SQL-Server o Oracle9i, disponen ya de herramientas que permiten hacer consultas a las bases de datos en SQL y devuelven los resultados en XML.

2.1. Esquema Comparativo de los Diferentes Sistemas que Relacionan XML y Bases de Datos Relacionales.

Los trabajos expuestos, sistemas que relacionan XML y bases de datos relacionales, se pueden clasificar en cuatro categorías, los que permiten hacer conversiones entre distintos formatos, de XML a base de datos y viceversa. Los que además de permitir la conversión, incorporan un lenguaje para consultar los datos ya en XML, los que analizan una serie de lenguajes de consulta de información almacenada en XML y finalmente, los métodos que permiten hacer

consultas a la base de datos en SQL, pero que transforman y devuelven los resultados en formato XML.

1. SISTEMAS DE CONVERSIÓN DE INFORMACIÓN ALMACENADA EN DISTINTOS FORMATOS.

Dentro de esta categoría se distinguen dos tipos:

a) *Sistemas que permiten convertir información en XML a Base de Datos.* Se caracterizan por almacenar la información, que inicialmente estaba en XML, a una BD, de forma que puede ya ser consultada con un lenguaje de base de datos como SQL.

   Dentro de este grupo estarían, Monet [Schimidt 2000] y el sistema de Shanmugasundaram [Shanmugasundaram et al. 1999].

b) *Sistemas que permiten convertir información en XML a Base de Datos y viceversa.* Permiten la traducción en ambos sentidos mediante algún componente especial.

   Así por ejemplo, en X-Ray [Kappel et al. 2000] se incluye un metaesquema especial denominado *XMLDBSchemaMapping.;* en los trabajos de Peter McBrien y Alexandra Poulovassilis [McBrien y Poulovassilis (1) 1999] [McBrien y Poulovassilis (2) 1999] [McBrien y Poulovassilis 2001], el componente que ayuda a hacer la conversión entre diferentes modelos, es un modelo de datos gráfico especial denominado *Hypergraph Data Model (HDM).*

2. SISTEMAS QUE LLEVAN INTEGRADO UN SISTEMA DE CONVERSIÓN Y UN LENGUAJE DE CONSULTA PARA LOS DATOS YA CONVERTIDOS A XML.

Este tipo de sistemas van a permitir por un lado, la conversión entre datos de distintos formatos, que no tiene porque ser de base de datos relacional a XML, y por otro lado, proponen un lenguaje de consulta para el tratamiento de la información ya traducida a XML. Dentro de este grupo están:

- El modelo de conversión Yat [Christophides y Cluet 2000] que sólo trata datos semiestructurados, normalmente ya en XML y un lenguaje de consulta propio para los datos ya convertidos, denominado YatL.
- El modelo de integración SINGAPORE [Dittrich y Domenig 1999][Dittrich y Jonscher 2000] que además de tratar datos semiestructurados, como los de formato XML y BDR, también permite integrar datos de bases de datos orientadas a objetos. Incluye un lenguaje propio denominado SOQL, que además de usarse para consultas, también permite realizar actualizaciones sobre la información ya transformada.

- SilkRoute [Fernández y Tan 2000] permite consultar información que inicialmente procede de una base de datos, en un lenguaje propio denominado RXL. Para poder realizar la consulta la información de la base de datos tiene que ser traducida previamente a XML, aunque sólo la parte afectada por la consulta (*XMLView*). Para realizar la conversión se permite utilizar cualquiera de los métodos anteriores, con tal que al final la información éste en XML para su consulta.
- XPERANTO[Carry et al. 2000] permite consultar datos de bases de datos Objeto-Relacionales en un lenguaje de consulta XML, y los resultados los devuelve también en XML. El proyecto se basa en aplicar un proceso de traducción de la consulta inicial en lenguaje XML-QL a SQL, a través de un gráfico intermedio denominado XQGM. Antes de realizar la consulta hay que obtener el XMLSchema en XML, asociado a la Base de Datos.

3. LENGUAJES DE CONSULTA PARA INFORMACIÓN EN XML

Si la información original procede de BD, estos lenguajes suelen tener una sintaxis más parecida al lenguaje SQL, y sino usan una sintaxis más semejante a XML con patrones y caminos. Estos lenguajes se pueden clasificar por tanto, según los dos entornos fundamentales de los que puede derivar, el entorno de bases de datos o del entorno XML.

a) *Lenguajes de Consulta procedentes del entorno de Base de Datos.*

- YatL del sistema Yat [Christophides y Cluet 2000]
- SOQL, del sistema SINGAPORE [Dittrich y Domenig 1999][Dittrich y Jonscher 2000]
- RXL, del sistema SilkRoute [Fernández y Tan 2000]
- XML-QL [Deutsch 1999]
- Lorel [Abiteboul et al. 2000] [McHugh 1997] [Abiteboul et al. 1997] que tiene influencias de OQL, un lenguaje de consulta del entorno de bases de datos orientadas a objetos.

b) Lenguajes de Consulta procedentes del entorno de XML.

- El único lenguaje gráfico, XML-GL [Ceri et al. 1999].
- Xpath, lenguaje de patrones y caminos.
- XSL que permite crear hojas de estilo e internamente también utiliza Xpath.
- XQL [Robie 1999] que es la combinación de XSL y XPath y esta claramente orientado a la consulta.

- XQuery con influencias tanto de lenguajes de consulta basados en XML, como de BD. Es quizás el más completo de todos, aunque aún esta en proceso de definición.

4. HERRAMIENTAS DE CONSULTA DE INFORMACIÓN DE BD, QUE DEVUELVEN LOS RESULTADOS DE LA CONSULTA EN XML.

Este tipo de herramientas permiten consultar una base de datos en SQL, pero obtienen los resultados de la consulta en XML. Dentro de este grupo hay dos tipos, aquellas que incorporan los propios gestores de bases de datos y las que están más orientadas a obtener la información en un formato Web.

a) *Obtención de la información de consulta en XML a través de paquetes especiales de los motores de búsqueda o sistemas de gestión de base de datos.*

Los motores de búsquedas o Sistemas de Gestión de Base de Datos potentes como Oracle o SQL Server, incorporan en sus aplicaciones, herramientas especiales para el tratamiento de información en XML. Oracle9i tiene por ejemplo un paquete especial (XML Development Kit o XDK9i) que permite, entre otras cosas, hacer una consulta a una base de datos en SQL y obtener el resultado en XML, a través de lo que denominan páginas XSQL.

b) Obtención de la información de consulta en XML a través de páginas web.

Se trata de un tipo especial de páginas Web, que permiten consultar la base de datos en SQL y obtener el resultado en XML en una página Web especial, que se denomina página "Activa". Dentro de este tipo están las páginas ASP (de Microsoft), las páginas JSP (implementación en Java) y páginas PHP (implementación en cualquier lenguaje).

3. Limitaciones de los sistemas planteados

En la introducción anterior del estado del arte, se hace un recorrido por diversas técnicas o modelos que directa o indirectamente relacionan las bases de datos en general, con XML, y por diferentes lenguajes de consulta de datos en XML.

El tema se pretende centrar al final en tres puntos fundamentales, un tipo de bases de datos, las relacionales, por ser uno de los métodos de almacenamiento y tratamiento de datos tradicionalmente más extendido. Su relación con XML, por ser éste otro método de almacenamiento de datos, pero más actual y que cada día cobra más fuerza. Finalmente, si se tiene información de los dos tipos, cómo poder consultarla, que este acceso resulte eficiente e incluso transparente para el usuario final, es decir, que éste simplemente consulte, sin saber en realidad cuál es el sistema real de almacenamiento de estos datos.

Con las consideraciones planteadas, se trata de hacer un análisis de los sistemas mencionados y estudiar cuales son sus limitaciones en relación al tipo de sistema que se desea, cuyas características se acaban de exponer. Al mismo tiempo sin embargo, también pueden resultar útiles en otros aspectos, las ideas en las que estos métodos se apoyan.

Por ejemplo, SINGAPORE o Yat, son más modelos de integración de datos de diferentes tipos, estructurados, como las BDR, o no estructurados, que modelos de consulta, por tanto van más allá del simple acceso a las BDR e información en XML, que es lo que se pretende.

El gráfico HDM se utiliza sobre todo para hacer conversiones de información de distintos formatos pero perteneciente al mismo universo de datos, es decir, información referente más o menos al mismo tema, pero perteneciente a distintas fuentes que al mismo tiempo tienen que colaborar entre sí. En el modelo planteado no importa que los temas de la información a consultar sean diferentes.

En la tecnología ASP, JSP o PHP y en las nuevas herramientas de SGBDR como SQL-Server o Oracle9i, las consultas se hacen directamente en SQL, no en XML, y sólo sirven para bases de datos relacionales, no también para consultar al mismo tiempo documentos XML.

X-Ray permite convertir los datos de la base de datos a XML, aunque con la limitación de que hay que establecer una determinada relación de la BD como eje para convertirla a XML, y una vez que los datos estén en este formato, se puede consultar con cualquier lenguaje de consulta XML. Su limitación esta en esta conversión, pues se trata de que los datos reales de la base de datos relacional no tengan que traducirse en ningún momento ni a XML, ni a ningún otro entorno, que permanezcan en su estado original.

SilkRoute se acerca a uno los objetivos principales, consultar una BDR con un lenguaje de consulta XML, pero para conseguirlo, una parte de la base de datos (*XML View*), la que interese según la consulta, es traducida previamente a XML para permitir su acceso. Lo ideal sería, como se ha dicho, evitar cualquier tipo de conversión de la información almacenada.

XPERANTO plantea una idea bastante similar a la expuesta, pero las bases de datos con las que trabajan no son relaciónales puras, sino objeto-relacionales, con lo que aparecen nuevos conceptos a tener en cuenta en el modelo de datos, como los de herencia y tipos especiales de datos. Este proyecto puede aportar alguna idea, aunque por ejemplo, falta aclarar como se hace exactamente la conversión de la base de datos al XMLSchema de XML. Por otro lado, podría resultar más práctico no tener que usar ningún gráfico intermedio par hacer las conversiones, sino que se puedan hacer directamente desde la información que se deriva del esquema de base de datos que se obtiene del propio sistema gestor de base de datos.

El objetivo final sería permitir la consulta de información almacenada en una Base de Datos Relacional o en un documento XML, por medio de un lenguaje de consulta basado en XML y retornando los resultados de la consulta también en XML. De esta forma todo el entorno tendría un aspecto XML para el usuario, aunque internamente este no fuera el formato real de los datos. Además se trata de lograr todo esto evitando los problemas que hemos comentado, por ejemplo que los datos no tengan que ser convertidos a ningún formato intermedio, que permanezcan siempre en su formato original.

4. Conclusiones y Líneas Futuras

El estudio de las limitaciones de los modelos descritos justifica aún más la necesidad de crear un sistema que permita consultar datos que pueden pertenecer a Bases de Datos Relaciónales o a documentos XML, pero además con el mismo sistema de consulta, y basado en XML, para que todo el entorno tenga la misma apariencia.

Lo que se plantea como trabajo futuro es establecer por un lado, el modelo que nos permita migrar la Base de Datos a un formato que la haga accesible desde XML, pero sin tener que hacer ningún tipo de conversión con los datos o registros de información de la base de datos, que era uno de los problemas de algunos de los sistemas anteriores. La idea es migrar el modelo de datos que define a la base de datos, su modelo Entidad-Relación, no sus datos, a un modelo semiestructurado, como son los DTD o XMLSchema de XML.

La segunda tarea fundamental es establecer el lenguaje de consulta, basado en XML, que se va a usar este entorno y sobre todo, plantear como se pasaría de este lenguaje a SQL, que es el lenguaje con el que al final se va a acceder a la base de datos. No se trata de inventar un nuevo lenguaje, se puede reutilizar alguno de los lenguajes expuestos procedentes del entorno de XML, para que todo tenga el mismo aspecto, y adaptarlo, si se requiere, a nuestras necesidades específicas.

5. Bibliografía

S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan-Kaufmann 2000

S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener. *The Lorel Query Language for Semistructured Data.* International Journal on Digital Libraries, vol. 1, no. 1, pg (68-88). Abril 1997.

T. Bray et al. *eXtensible Markup Language (XML) 1.0.* World Wide Web Consortium. Disponible en. http://www.w3c.org/TR/REC-xml 1998

M. Carry, D. Florescu, Zachary Ives, Ying Lu et al. *XPERANTO: Publishing Object-Relational Data as XML.* WebDB (Informal Procedings), pg. 105-110. Disponible en http://citeseer.nj.nec.com/christophides00wrapping.html. 2000

Vassilis Christophides, Sophie Cluet. *On wrapping query languages and efficient XML integration.* Disponible en http://citeseer.nj.nec.com/christophides00wrapping.html. 2000

S. Ceri, s. Comai, E. Damiani, P. Fraternali, S. Paraboshi, L. Tanca. *XML-GL: A Graphical Language for Quering and Restructuring XML Documents.* Proceedings of the International WWW Conference. 1999

S.J. DeRose. *XQuery: A Unified Syntax for Linking and Querying General XML.* Proc. of the Query Languages Workshops (QL98), Cambridge, http://www.w3.org/TandS/ql/QL98/xquery.html, Diciembre 1998.

Klaus R. Dittrich,, Ruxandra Domenig. *Towards Explotation of the Data Universe.* 3rd International Conference on Bussiness Information Systems, Poznan, Springer, Abril 1999

A. Deutsch, M. Fernández, D. Florescu, A. Levy, D. Suciu. *A query language for XML.* International WWW Conference. 1999

Klaus R. Dittrich,, Dirk Jonscher. *All Together Now: Towards Integrating the World's Information Systems.* V Jornadas de Ingeniería del Software y Bases de Datos., Valladolid (España). Noviembre 2000.

Mary Fernandez, Jerome Simeón, PhilipWadler. *XML Query Languages: Experiences and Exemplars.* Disponible en http://wwwdb.research.bellabs.com/user/simeon/xquery.ps , 1999.

Getti Kappel, Elisabeth Kapsammer, Werner Retschitzegger. *X-Ray: Towards Integrating XML and Relational Database Systems.* International Conference on Conceptual Modeling, the Entity Relation Ship Approach (pag. 339- 353). 2. Julio 2000

D. Maier. *Database Desiderata for an XML Query Language.* W3C Workshop on Query Languages for XML. 1998

Michael Morrison, et al. *XML Al descubierto.* Prentice Hall 2000

J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom. *Lore: A Database management system for semistructured data.* ACM SIGMOD Record, vol. 26, no. 3, 1997

Ioana Manolescu, Daniela Florescu, Donald Kossmann *Pushing XML Queries inside Relational Database.* Tech. Report no. 4112, INRIA. Disponible en www.caravel.inria.fr/Epublications.html, 2001

P. McBrien, A. Poulovassilis. *Automatic Migration on Wrapping of Database Applications. A schema Transformation Approach.* Procedings of ER99, vol. 1728 of LNCS, PG. (96-113). Springer-Verlag 1999

P. McBrien, A. Poulovassilis.*A uniform Approach to Inter-model Transformations.* 11th International Conference on Advanced Information Systems Engineering (CAiSE'99), vol. 1626 og LNCS pg. (333-348). Springer Verlag 1999

P. McBrien, A. Poulovassilis. *A Semantic Approach to integrating XML and Structured Data Sources.* 13th International Conference on Advanced Information Systems Engineering (CAiSE'01). 2001

Mary Fernandez, Wang-Chiew Tan. *SilkRoute. Tradding between Relations and XML.* WW9/Computer Networks journal, vol. 33, no. 1-6, pg (723-745). 2000

J. Robie. *The design of XQL.* Disponible en http://www.textel.no/whitepapers/xql-design.html. 1999

Albretch Schimidt, Martin Kersten, Menzo Windower, Florian Lucas. *Efficient Relational Storage and Retrieval of XML documents.* WebDB 2000, pg.(47-52). 2000

Jayavel Shanmugasundaram, Kristing Tufte, H. Gang, Chung Zhang. *Relational Databases for Querying XML documents: Limitations and Opportunities.* The VLDB journal on the Proceedings of the 25th VLDB Conference, pg. (302-314). 1999

. Direcciones de Internet:

- http://www.w3.org/: World Wide Web Consortium
- http://www.w3.org/xml: W3C Extensible Markup Language (XML) 1.0
- http://www.w3.org/TR/xpath: XML Path Language (XPath)
- http://www.w3.org/TandS/QL/QL98/pp/xql.html: XML Query Language. (XQL).

- http://www.w3.org/TR/xqueryl: W3C XQuery 1.0: An XML Query Language
- http://www.w3.org/Style/XSL: W3C XSL and XSLT.

O XML nos curriculae das licenciaturas em informática

José Carlos Ramalho
jcr@di.uminho.pt
DI/UM

10 de Fevereiro de 2003

Resumo

Devido ao grande sentido ecuménico assumido cada vez mais por esta tecnologia, a sua inclusão em várias disciplinas leccionadas ao longo da Licenciatura em Engenharia de Sistemas e Informática e na Licenciatura em Matemática e Ciências da Computação foi inevitável.

No entanto, a evolução das tecnologias à volta do XML e o cada vez maior número de aplicações à volta desta norma forneceram o contexto ideal para a criação de uma disciplina à volta deste tema. E é assim que, desde 2000, uma disciplina com a designação de ”Processamento Estruturado de Documentos”tem vindo a ser oferecida como disciplina de Opção às duas licenciaturas acima referidas, ao Mestrado em Informática, ao Curso de Especialização em Informação, e ainda como um dos módulos do programa de Formação Contínua do Departamento de Informática / Escola de Engenharia da Universidade do Minho.

Esta disciplina tem registado uma enorme adesão em todas as suas instâncias. O seu sucesso pode ser medido pela quantidade e qualidade dos projectos desenvolvidos: esta informação está acessível a partir da página do autor: `http://www.di.uminho.pt/~jcr`.

Os projecto têm abordado várias áreas: a publicação electrónica, ontologias, WebServices, animação gráfica, construção dinâmica de websites, intercâmbio de informação, transformação de bases de dados, ...

Na apresentação a realizar na Workshop, serão apresentadas algumas das aplicações desenvolvidas, nomeadamente: a publicação electrónica de edições esgotadas do Arquivo Distrital de Braga, a produção de Guiões Online para aulas práticas e a geração automática de Websites a partir de ontologias especificadas em XML.

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

13 Fev.	S2(11.00h) XML e Bases de Dados	[*XML - Solução de Integração de Sistemas para uma Gestão Distribuída de Seguros*] - **Jorge Miranda**, I2S; [*Comparação de Várias Estratégias para Armazenamento de XML*] - **Rui Cerveira Nunes**, IST; **Miguel Mira da Silva**, IST; [*Abordagens para armazenamento de metadata em Datawarehouses usando XML*] - **Hugo Alhandra**, IST; **Miguel Mira da Silva**, IST;

XML - Solução de integração de sistemas para uma Gestão Distribuída de Seguros

Jorge Miranda Jorge.Miranda@i2s.pt
Miguel Moreira Miguel.Moreira@i2s.pt
Sector de Soluções eBusiness
Informática - Sistemas e Serviços, s.a.

1 Introdução

O presente documento, elaborado no âmbito da *Workshop* "XML – Aplicações e Tecnologias Associadas", pretende expor, de forma resumida, o papel do XML num projecto de implementação de Gestão Distribuída de Seguros, dando particular ênfase à flexibilidade proporcionada pelo XML nas diversas soluções encontradas e demonstrando que o XML é já uma realidade como tecnologia aplicada em larga escala em projectos empresariais de grande envergadura.

2 Apresentação do projecto

A nova economia veio revolucionar a forma de fazer negócios, e abrir novos horizontes a sectores com uma longa tradição, como é o caso da área seguradora, constituindo um enorme desafio para inúmeras empresas.

Uma grande parte das seguradoras que possuem um saber acumulado ao longo da sua história vêem-se agora confrontadas com a necessidade de se reposicionarem no mercado e, principalmente, reformular toda a sua lógica de funcionamento interno, de forma a poderem atingir uma maior eficácia e rapidez de actuação no mercado, conjuntamente com uma preocupação acrescida no que respeita ao relacionamento com o cliente e com a qualidade dos serviços que lhe são prestados.

Dentro deste cenário de rápidas mudanças, a I2S, enquanto *software house* especializada no fornecimento de soluções para gestão de seguros, líder no mercado nacional, foi confrontada, no ano de 1999, por um dos seus clientes mais importantes, com o desafio de desenvolver uma solução para venda e gestão de seguros aos balcões de um dos mais importantes grupos bancários nacionais.

O projecto envolveria várias entidades: uma seguradora cujo negócio era já gerido pelo ERP para Seguros fornecido pela I2S (o GIS® – Gestão Integrada de Seguros®), 3 bancos com quem a Seguradora tem parcerias estabelecidas e empresas de consultoria contratadas pelos bancos.

2.1 Objectivos do projecto inicial

O projecto em causa pretendia criar uma solução total de gestão de seguros (Banca-Seguros), funcionando nos balcões do já referido Grupo Bancário, mas completamente integrada com os vários sistemas nos quais se encontravam as informações críticas para o negócio, nomeadamente:

- Sistema central do Banco;
- Sistema central da Seguradora.

Um objectivo essencial definido para o projecto era que a solução final integrasse, de forma transparente e amigável para o utilizador, a informação proveniente dos vários sistemas.

2.2 Objectivos complementares definidos pela I2S

Tendo identificado este projecto como uma importante oportunidade de modernizar e alargar o âmbito das suas soluções, a I2S baptizou o projecto com o nome de "eGIS", tornando-se desde logo uma peça fundamental na estratégia de evolução definida para a sua solução GIS®.

No âmbito dessa estratégia foram definidos objectivos internos da I2S, que incluíam os seguintes pontos:

- Conceber uma solução generalista, e não à medida do cliente;
- Implementar uma solução suficientemente flexível para permitir a sua utilização em diferentes canais de comercialização de seguros, nomeadamente na Internet, nas agências das seguradoras e em balcões de bancos;
- Definir uma arquitectura altamente modular e com elevado grau de isolamento que permitisse a sua adaptação a diferentes cenários sem grandes necessidades de desenvolvimento.

3 Análise do Problema

Enquadrada pelos objectivos atrás mencionados, pelos condicionalismos tecnológicos existentes e pelas preocupações de segurança dos Bancos, foi definida uma Arquitectura Global. Esta definição foi, numa primeira fase, de alto nível, pois existiam ainda muitos pontos a estudar e investigar antes de poder ser detalhada nos seus vários componentes.

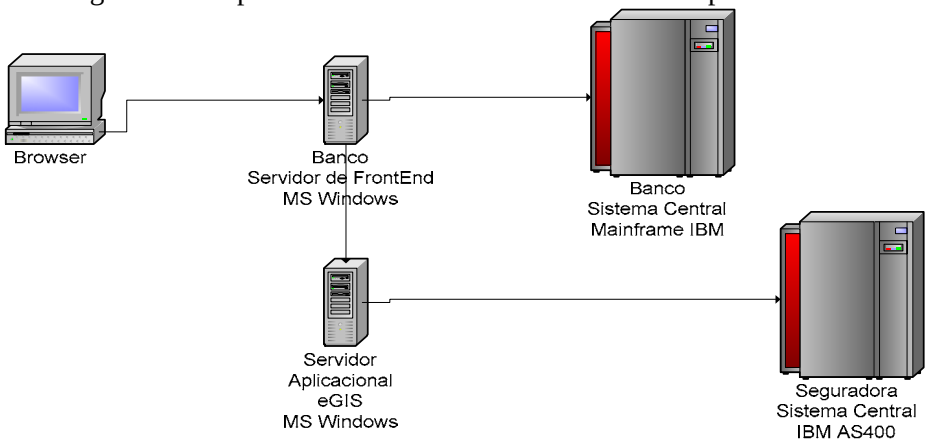


Figura 1 – Arquitectura geral

Os componentes "base" da solução eram:

- "Servidor de *FrontEnd*", com Microsoft Windows 2000, servidor da rede bancária, responsável pela recepção, transformação e encaminhamento da informação trocada entre os balcões e os vários sistemas;
- "*Mainframe*", com IBM MVS, sistema central de gestão do banco, com o qual seriam efectuadas transacções relativas a dados dos clientes e débitos a contas;
- "eGIS", com Microsoft Windows 2000, servidor aplicacional, contém as aplicações da I2S, com toda a lógica de negócio e de integração;
- "AS400 / iSeries", com IBM OS/400, sistema central de gestão da seguradora, utilizado em consultas e na integração final das transacções.

Partindo desta definição passou-se então à análise mais detalhada de todo o sistema, que iremos seguidamente abordar nas suas várias vertentes.

3.1 Condicionalismos estratégicos e tecnológicos

A arquitectura apresentada na secção anterior, conjuntamente com alguns pressupostos definidos inicialmente, resulta em alguns condicionalismos que se viriam a revelar fundamentais na definição do caminho a seguir na implementação do projecto:

- Todas as funcionalidades de "interface homem/máquina" deveriam estar devidamente isoladas;
- Toda a lógica de negócio deveria estar no servidor aplicacional eGIS;
- Todas as interacções com os sistemas do banco, nomeadamente consultas a dados de clientes e débitos bancários, seriam efectuadas recorrendo a API cujo desenvolvimento seria independente da I2S. Assim sendo, seria necessário isolar este tipo de interacções num módulo com funcionamento independente dos sistemas com o qual irá comunicar.

3.2 Problemas a resolver

3.2.1 Definição dinâmica de ecrãs

A complexidade do negócio segurador é muito elevada, com uma grande variedade de produtos (tipos de seguro) cobrindo situações muito diversas. Com este projecto, pretendia-se que, sempre que um novo tipo de seguro fosse parametrizado no sistema central da seguradora, ele ficasse imediatamente disponível nos balcões dos bancos, sem a necessidade de qualquer desenvolvimento adicional. Desta vontade resultou a necessidade de ter uma interpretação das parametrizações dos produtos capaz de gerar e validar dinamicamente a interface com o utilizador.

Adicionalmente a esta situação, dois dos condicionalismos atrás mencionados contribuíam para aumentar a complexidade do problema. Pretendendo-se simultaneamente que a interface com o utilizador fosse desenhada e controlada pelo Banco, e que a lógica do negócio estivesse centralizada nos componentes desenvolvidos pela I2S, rapidamente se chegou à conclusão que seria necessário definir uma forma para que os componentes I2S informassem os componentes do banco sobre:

- Qual a informação a incluir em cada interacção com o utilizador;
- Qual a formatação dessa informação;
- Quais as validações e/ou valores possíveis para a informação em causa.

3.2.2 Troca de informação entre sistemas

Pretendendo-se criar uma solução em que o utilizador não se apercebesse do sistema do qual é proveniente a informação, e simultaneamente reutilizar a maior quantidade de dados possível, desde logo foram identificadas algumas situações em que era necessário interagir com o *Mainframe* do Banco, quer para aceder à informação lá disponível, quer para efectuar transacções com esse sistema. Daqui resultou a necessidade de definir um protocolo de troca de informação entre o eGIS e sistemas externos.

3.2.3 Persistência da informação das transacções

Ao terminar uma sessão de um utilizador na aplicação, tendo sido concluída uma operação, é necessário armazenar os respectivos dados para posterior integração no sistema central da seguradora.

A informação a armazenar varia muito consoante o tipo de transacção efectuada; por exemplo, a informação relativa à criação de um contrato é extremamente complexa, enquanto que a informação relativa a um pedido de anulação de um contrato é extremamente simples. Para um mesmo tipo de operação, a informação varia também bastante, dependendo do tipo de seguro.

Pretendia-se criar uma base de dados capaz de armazenar esta variedade de situações, e que simultaneamente tivesse a simplicidade inerente a uma base de dados onde a informação é persistida de forma transitória, até ser integrada definitivamente no sistema central da seguradora.

Paralelamente os dados arquivados seriam utilizados para fornecer informação para a impressão de documentos.

3.2.4 Integração no sistema central

O processo de integração da informação no sistema central deveria ser capaz de garantir a coerência do sistema de Back Office, fazendo cálculos ou validações adicionais, para além de todas as que já tinham sido efectuadas no eGIS. No âmbito de outros projectos, para os quais também era necessário garantir esta coerência, estavam já em desenvolvimento um conjunto de objectos de negócio, desenvolvidos em Java, tendo sido convencionado que toda a informação proveniente de sistemas externos e a integrar no GIS® deveria passar pelas validações associadas a esses objectos de negócio.

Havia portanto a necessidade de conseguir passar a informação guardada na base de dados do *MiddleWare* eGIS para os objectos de negócio.

3.2.5 Consultas

Uma das funcionalidades fundamentais da solução era proporcionar a consulta *online* à informação existente no sistema central da seguradora. Um dos princípios desde logo definido para estas consultas era que deveria existir uma independência total das consultas efectuadas via eGIS relativamente à base de dados do sistema central, para que sempre que essa base de dados fosse alterada não fosse necessário alterar componentes do eGIS. Considerando a já referida existência de objectos de negócio desenvolvidos ao nível do sistema central, pretendia-se utilizá-los também para extracção de informação.

4 eGIS, as Soluções

Complementarmente à arquitectura simplificada já apresentada, foi definida a estrutura de comunicação entre os vários componentes.

Face aos condicionalismos expostos no ponto 3.1, rapidamente se concluiu da excessiva rigidez apresentada pelas soluções comummente usadas na comunicação entre sistemas das seguradoras e dos bancos, quer ficheiros com formato predefinido (*flat files*), quer API. Por outro lado, a utilização dos padrões de EDI existentes (como o UN/EDIFACT) resultava em projectos de implementação muito demorada.

No período em que o projecto arrancou verificava-se uma grande euforia relativamente a uma panóplia de novas tecnologias. Nesta torrente de propostas de soluções que surgiam no fim do milénio, o XML destacava-se como uma solução com um potencial elevado de flexibilidade e rapidez na concepção e implementação da aplicação eGIS. Como tecnologia que permite transmitir dados e a sua descrição, permitia-nos simultaneamente:

- especificar a informação a incluir nos ecrãs e respectivas validações;
- efectuar pedidos de fornecimento de informação a sistemas externos;
- persistir informação de uma sessão de trabalho entre iterações;
- suportar a transferência de informação entre sistemas.

4.1 Definição dinâmica de ecrãs

Para resolver o problema da definição das páginas/ecrãs a apresentar ao utilizador, a melhor solução encontrada foi a definição de uma especificação que permitisse a descrição dos respectivos ecrãs.

O API que já existia, desenvolvido pela I2S e baseado em *arrays*, foi substituído por XML, tendo sido criado um XML Schema em que era especificada a forma como os ecrãs seriam definidos.

O mecanismo de interacção pressupunha que o eGIS enviava um pedido de construção de ecrãs a um componente que foi baptizado de "formatador de interface", que seria responsável por converter o XML de especificação do ecrã numa página HTML e enviá-la para o browser. Posteriormente, quando o resultado da interacção do utilizador com a página era devolvido ao servidor HTTP, os dados provenientes do browser eram convertidos para o XML no formato predefinido e enviados para o componente eGIS para processamento.

Desta forma, a forma como as páginas HTML são desenhadas e controladas é completamente independente do eGIS, podendo existir várias implementações com diferentes interfaces, baseadas no mesmo núcleo do eGIS.

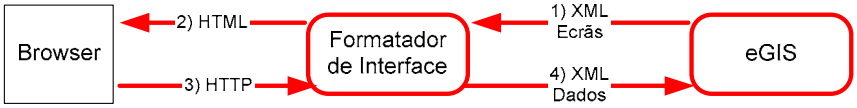


Figura 2 – Fluxo de criação de ecrãs

Durante a elaboração do protótipo inicial do projecto foi estudado e definido um XML Schema de troca de informação que permite a definição de:

- Pedidos de construção de ecrãs e respectiva resposta com os dados introduzidos;
- Pedidos de informação, enviando os dados pretendidos e obtendo os resultados fornecidos pelos sistemas destino dos pedidos;
- Pedidos de construção de relatórios, fornecendo a informação necessária para o seu preenchimento.

Com os padrões ainda em elaboração pelo W3C e face à implementação XML oferecida pela Microsoft, foram adoptados alguns padrões XML que o tempo entretanto já decorrido se encarregou de tornar obsoletos: XDR (XML-Data Reduced) na definição de Schemas e XSLT com XSL Patterns na pesquisa de informação. A sua evolução para padrões mais recentes tem sido gradual, de forma a garantir a estabilidade da aplicação em funcionamento.

4.1.1 Linguagem de especificação de Ecrãs

O XML Schema de especificação de páginas criado, cujos exemplos se apresentam na formatação Microsoft BizTalk definida à data de arranque do projecto, contém os seguintes conceitos:

pEcran – Pedido de construção de ecrã

Estes pedidos contêm o XML de definição das páginas / ecrãs e são enviados para o Formatador de interface, responsável por gerar as páginas HTML.

Foi definido que um ecrã poderá conter:

- Agrupamentos de campos, área "lógica" do ecrã
- Campos
- Conjuntos de campos
- Conjunto de relatórios para impressão, de forma a permitir também apresentar páginas HTML formatadas para impressão
- Botões

<**pecran**
idecran = *string*
titulo = ***string***
>

many	[<agrupamento>] * [<campo>] * [<conjunto_campos>] * <lista_botoes> [<conjunto_relatorios_impressao>] *

mixed content
</**pecran**>

Agrupamento – Área lógica de um ecrã, com um cabeçalho (etiqueta); pode agrupar vários campos ou tabelas.

Um agrupamento poderá conter:

- campos ou conjuntos de campos
- tabelas
- listas de botões

<**agrupamento**

```
[  etiqueta = string ]
   tipo = enumeration: ecra_ant | ecra_act | erros
>
many | [ <campo> ] *
     | [ <conjunto_campos> ] *
     | [ <tabela> ] *
     | [ <lista_botoes> ]
</agrupamento>
```

Conjunto de campos – permite agrupar campos de alguma forma relacionados para serem apresentados ao mesmo nível (por exemplo, valores monetários em Escudos e em Euros).

```
<conjunto_campos
 [  separador = string ]
>
seq | <campo>
    | <campo>
</conjunto_campos>
```

Campo – dado simples, ao qual poderão estar associadas as seguintes propriedades:

- tipo – indica se é um campo de texto, lista, visualização, etc.
- etiqueta – o que é apresentado ao utilizador a pedir a informação
- prefixo
- sufixo
- descritivo – pequeno texto de ajuda
- tamanho
- lista de valores válidos – para campos do tipo lista, por exemplo, meses
- valor inicial
- lista de validações a efectuar
- mensagem de erro – indicar valor mal preenchido na página anterior
- mensagem de aviso

```
<campo
 [   nome = id ]
 [   obrigatorio = boolean ]
 [   ref = idref ]
 [   tipo = enumeration: alfanumerico | sim_nao | data | numerico | visualizacao | erro | lista |
     escondido | aviso ]
>
many | [ <etiqueta> ]
     | [ <prefixo> ]
     | [ <sufixo> ]
     | [ <descritivo> ]
     | [ <tamanho> ]
     | [ <lista_valores> ]
     | [ <valorinicial> ]
     | [ <lista_validacoes> ]
     | [ <msg_erro> ]
     | [ <msg_aviso> ]
 mixed content
</campo>
```

Validação – Validação associada a um campo, destinada a especificar um pequeno texto em VBScript ou Javascript a executar quer para validar o conteúdo do campo (por exemplo, ver se é uma data válida) quer para alterar o valor de outro campo que dependa do actual.

<**validacao**
[idval = *id*]
[ref = *idref*]
[carregar_pagina = *enumeration*: sim | **não**]
[sair_campo = *enumeration*: **sim** | nao]
[sair_pagina = *enumeration*: sim | **nao**]
>
mixed content
</**validacao**>

Botão – "Botões" a apresentar ao utilizador para enviar a página ao servidor.

<**botao**
accao = *string*
etiq_botao = *string*
/>

Resposta:

rEcran – XML contendo os dados introduzidos pelo utilizador e as acções seleccionadas

<**recran**
accao = *string*
idecran = *string*
>
many [<campo>] *
</**recran**>

Exemplos de uma interacção, com o envio de um pEcran, e a recepção do rEcran:

pEcran enviado:

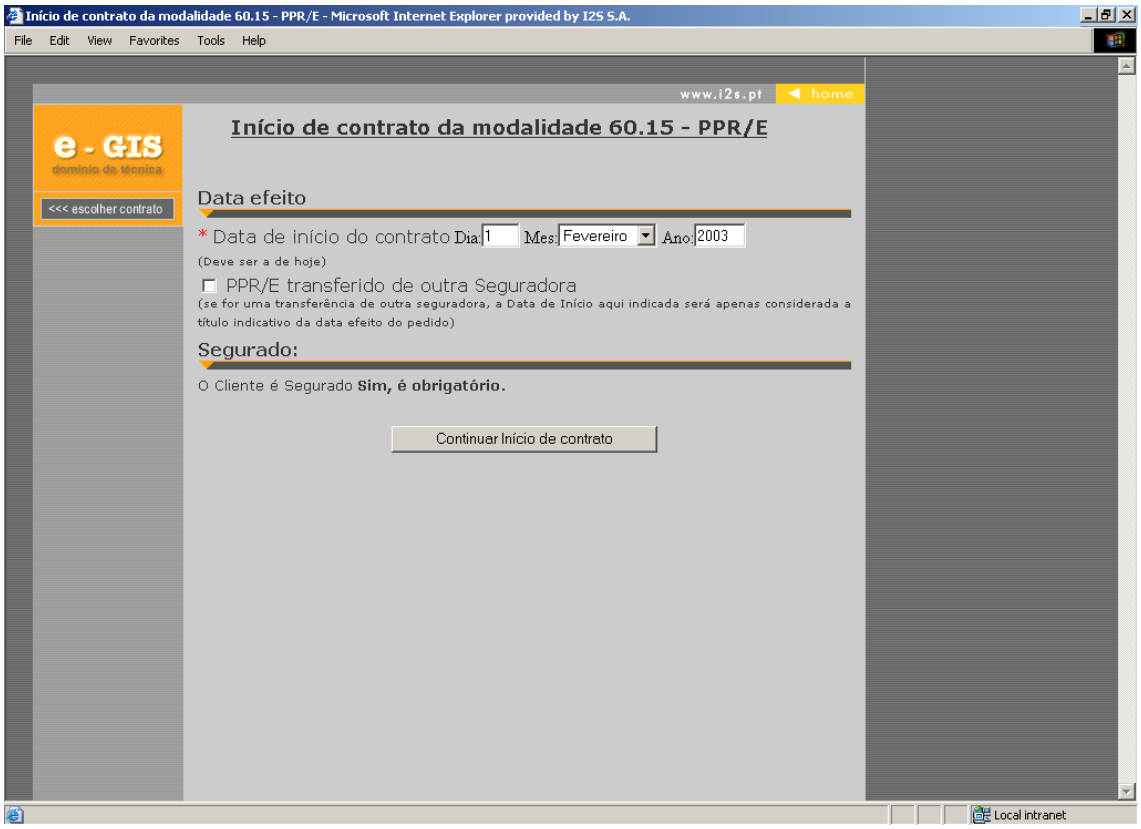
```
<pecran titulo="Início de contrato da modalidade 60.15 - PPR/E"
   idecran="6016">
  <agrupamento tipo="ecra_act" etiqueta="Data efeito">
    <campo tipo="data" nome="DataInicio" obrigatorio="1">
      <valorinicial>01-02-2003</valorinicial>
      <etiqueta>Data de início do contrato</etiqueta>
      <descritivo>Deve ser a de hoje</descritivo>
      <lista_validacoes>
        <validacao idval="DataInicio__DATABASEVAL" carregar_pagina="nao"
         sair_campo="nao" sair_pagina="sim">
          <![CDATA[ DateMustBeValid( DataInicio,
              'Não é uma data válida.' )]]>
        </validacao>
      </lista_validacoes>
    </campo>
    <campo tipo="sim_nao" nome="PPRTransfer">
      <valorinicial>nao</valorinicial>
      <etiqueta>PPR/E transferido de outra Seguradora</etiqueta>
      <descritivo>se for uma transferência de outra seguradora, a Data de
        Início aqui indicada será apenas considerada a título indicativo da
        data efeito do pedido
      </descritivo>
    </campo>
  </agrupamento>
  <agrupamento tipo="ecra_act" etiqueta="Segurado:">
    <campo tipo="visualizacao" nome="cli_pessoa_segura_mos">
      <etiqueta>O Tomador (cliente) é Segurado</etiqueta>
      <valorinicial>Sim, é obrigatório.</valorinicial>
    </campo>
  </agrupamento>
```

```
  <lista_botoes>
    <botao etiq_botao="Continuar Início de contrato" accao="Continuar"/>
  </lista_botoes>
  <![CDATA[
    function DateMustBeValid( data, msg ){
      try{
        var dateStr = new String( data.value );
        if ( dateStr.length == 0 ) return true;
        var ano = dateStr.substr(6,4);
        var mes = dateStr.substr(3,2);
        var dia = dateStr.substr(0,2);
        var date1 = new Date( ano, mes - 1, dia, 0, 0, 0, 0 );
        if ((date1.getMonth() != mes - 1) || (date1.getFullYear() != ano))
          return ShowErrorMsgAndFail( msg );
      }
      catch( e ){
        return ShowErrorMsgAndFail( msg );
      }
      return true;
    }
    function ShowErrorMsgAndFail( msg ) {
      var stCaller = ShowErrorMsgAndFail.caller + "";
      alert( msg );
      if ( stCaller.substr(0,18) == "function anonymous" )
        return true;
      else return false;
    }]]>
</pecran>
```

rEcran recebido

```
<recran idecran="6016" accao="Continuar">
  <campo nome="DataInicio">19/03/2002</campo>
  <campo nome="PPRTransfer">nao</campo>
  <campo nome="cli_pessoa_segura">Sim, é obrigatório.</campo>
</recran>
```

Usando um XSL para converter o XML de pEcran para HTML podemos ver um ecrã gerado:



4.2 Troca de informação entre sistemas

Para permitir a troca de informação com outros sistemas foi criado um XML Schema de definição de pedidos de serviço que seriam enviados aos sistemas externos, sendo estes responsáveis por devolver uma resposta ao serviço de acordo com as regras predefinidas. A arquitectura definida não permitia a comunicação directa do servidor eGIS com o *gateway* de acesso ao *mainframe* IBM, pelo que a solução encontrada passou pela troca de pServicos / rServicos numa filosofia semelhante à dos ecrãs.

Mais uma vez o XML foi o suporte ideal para esta comunicação. Os componentes eGIS são responsáveis por fornecer um XML de pedido de serviço, que é enviado para um componente baptizado de "formatador de dados bidireccional", e aguardando uma resposta desse componente que terá a implementação específica e o conhecimento necessário para efectuar a ligação aos sistemas externos.

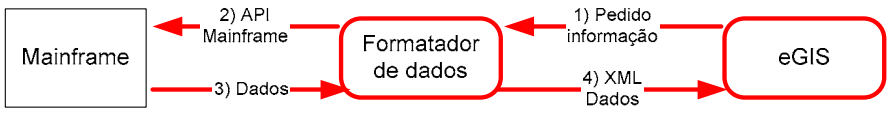


Figura 3 – Fluxo de troca de informação

4.2.1 Linguagem de troca de informação

O XML Schema criado contém os seguintes conceitos:

- pServico – XML pedindo o fornecimento de informação
- rServico – resposta ao pedido de informação, contendo os dados pretendidos.

<**pservico**
  classe = *enumeration*: I2SI | I2SIO | I2SBF
  id = *string*
>
many [<arg>] *
</**pservico**>

<**rservico**
  classe = *enumeration*: I2SI | I2SIO | I2SBF
  id = *string*
  status = *string*
>
many [<arg>] *
</**rservico**>

Exemplos de uma interacção, com o envio de um pServico, e a recepção do rServico:

pServico enviado:

```
<pservico classe="I2SBF" id="0001">
  <arg nome="ContaDebito">123456789001</arg>
  <arg nome="ContaCredito">023541180001</arg>
  <arg nome="DataValor">20030201</arg>
  <arg nome="ValorMovimentado">1000000</arg>
  <arg nome="MoedaUtilizada">EUR</arg>
  <arg nome="Contrato">00001236015000012345</arg>
  <arg nome="Operacao">1234567890</arg>
  <arg nome="TipoOperacao">IC</arg>
  <arg nome="Produto">000575</arg>
</pservico>
```

rServico recebido:

```
<rservico classe="I2SBF" id="0001" status="OK">
  <arg nome="Cod_Erro">00</arg>
  <arg nome="Desc_Erro"/>
  <arg nome="NomeTitular">MARIA ALBERTA SILVA FERNANDES</arg>
  <arg nome="CodTrans">7576</arg>
</rservico>
```

4.3 Persistência de informação das transacções

Tendo em conta a já referida variedade de informação a guardar por transacção, e não sendo objectivo que a base de dados do eGIS fosse uma réplica da complexa base de dados do sistema central da seguradora, achou-se por bem representar toda a informação de negócio relativa a uma transacção através de um XML, com uma estrutura predefinida baseada nos conceitos de negócio, capaz de suportar as situações existentes.

A gravação deste XML foi efectuada numa base de dados relacional que suporta os dados chave da transacção, conseguindo-se aliar as virtudes das bases de dados relacionais e do XML.

O Schema do XML usado foi simplificado ao máximo, para que a sua pesquisa e utilização na geração de relatórios fosse relativamente simples.

4.3.1 Gerador de informação para Relatórios

Para permitir a obtenção de relatórios sem efectuar desenvolvimentos específicos, foi criado um gerador que permite parametrizar a informação que se pretende incluir num determinado documento. Foram também criados um conjunto de conceitos, e, para cada um destes, o XPath necessário para extrair a respectiva informação no XML de *MiddleWare*.

Para cada documento são depois definidos quais os agrupamentos de impressão, e os conceitos a incluir em cada um dos agrupamentos, sendo o XML de impressão gerado automaticamente com base na configuração efectuada, de acordo com o XML Schema de "pEcran" anteriormente mencionado e que contém os seguintes etiquetas específicos de relatórios:

```
<conjunto_relatorios_impressao >
 many | <relatorio_impressao> +
</conjunto_relatorios_impressao>

<relatorio_impressao >
 seq | <agrupamento_impressao> +
</relatorio_impressao>

<agrupamento_impressao
   etiqueta = string
 [ nome = string ]
>
 one | <campo_impressao> +
     | <tabela_impressao>
</agrupamento_impressao>

<campo_impressao
   nome = string
>
     | <valorinicial>
     | [ <etiqueta> ]
 seq | [ <prefixo> ]
     | [ <sufixo> ]
</campo_impressao>
```

<**tabela_impressao**
[nome = *string*]
>
seq <cabecalho_impressao>
<linha_impressao> +
</**tabela_impressao**>

Exemplo de pEcran relativo a um documento de impressão:

```
<pecran titulo="Início de contrato da modalidade 60.15 - PPR/E"
idecran="ECIM">
  <agrupamento tipo="ecra_act" etiqueta="Contrato criado com sucesso">
    <campo tipo="visualizacao" nome="num_contrato">
      <etiqueta>Contrato nº</etiqueta>
      <valorinicial>00001236015000012345</valorinicial>
    </campo>
  </agrupamento>
  <agrupamento tipo="ecra_act" etiqueta="Avisos">
    <campo tipo="visualizacao" nome="aviso_1">
      <etiqueta/>
      <valorinicial>1ª via para cliente, 2ª via para balcão e posterior
        arquivo central, 3ª via (quando impressa) para a Seguradora, Notas
        Informativas e Condições Gerais (quando impressas) para cliente,
        Questionário Clínico (quando impresso) para a Seguradora.
      </valorinicial>
    </campo>
  </agrupamento>
  <conjunto_relatorios id="id">
    <relatorio id="PPR1" copias="2">
      <campo_simples nome="numero_contrato">00001236015000012345
      </campo_simples>
      <campo_simples nome="codigo_produto">000575</campo_simples>
      <campo_simples nome="nome_produto">PPR/E</campo_simples>
      <campo_simples nome="numero_conta">123456789001</campo_simples>
      <campo_simples nome="tomador_nome">MARIA ALBERTA SILVA FERNANDES
      </campo_simples>
      <campo_simples nome="tomador_num_cliente">123456789</campo_simples>
      <campo_simples nome="tomador_morada">RUA EXEMPLO, 345 - 2ºFRENTE
      </campo_simples>
      <campo_simples nome="tomador_cod_postal">4400-996 PORTO PORTUGAL
      </campo_simples>
      <campo_simples nome="tomador_data_nasc">01/02/1970</campo_simples>
      <campo_simples nome="tomador_profissao">DESCONHECIDA</campo_simples>
      <campo_simples nome="tomador_num_contrib">123456789</campo_simples>
      <campo_simples nome="forma_pagamento">ANUAL</campo_simples>
      <campo_simples nome="premio">10.000,00 EUR</campo_simples>
      <campo_simples nome="entrega_adicional"/>
      <campo_complexo nome="beneficiarios_vida">
        <esquema>
          <coluna_esquema nome="beneficiario"/>
        </esquema>
        <dados>
          <linha_dados>
            <coluna_dados nome="beneficiario">SEGURADO / PESSOA SEGURA
            </coluna_dados>
          </linha_dados>
        </dados>
      </campo_complexo>
      <campo_complexo nome="beneficiarios_morte">
        <esquema>
          <coluna_esquema nome="beneficiario"/>
        </esquema>
        <dados>
          <linha_dados>
            <coluna_dados nome="beneficiario">HERDEIROS DO(S) SEGURADO(S) EM
              PARTES IGUAIS
            </coluna_dados>
```

```
        </linha_dados>
       </dados>
     </campo_complexo>
     <campo_simples nome="impressao_2_via"/>
   </relatorio>
   <relatorio id="PPR1CG" copias="1">
     <campo_simples nome="nome_produto">PPR/E</campo_simples>
   </relatorio>
   <relatorio id="PPR1NI" copias="1">
     <campo_simples nome="nome_produto">PPR/E</campo_simples>
   </relatorio>
  </conjunto_relatorios>
</pecran>
```

4.4 Integração no sistema central

Para que uma transacção seja considerada pela Seguradora é necessário integrar o XML da transacção arquivada na base de dados do eGIS (XML *MiddleWare*) no sistema central da Seguradora.

Este processo obriga à transformação desse XML no momento da integração, pois os processos Java disponíveis no sistema central reconhecem um XML Schema bastante mais abrangente e complexo do que o utilizado no eGIS. Para resolver este problema recorreu-se à utilização de XSL. Através da aplicação desse XSL é obtido o XML Back Office, no formato reconhecido pelos objectos de negócio.

No entanto, o processo não termina aqui, pois para que a informação seja processada e registada no sistema central, é necessário que seja passada para os objectos de negócio, que contêm regras que irão garantir a validade da informação, e se tudo estiver correcto, se encarregarão de efectuar a gravação dos dados na base de dados relacional.

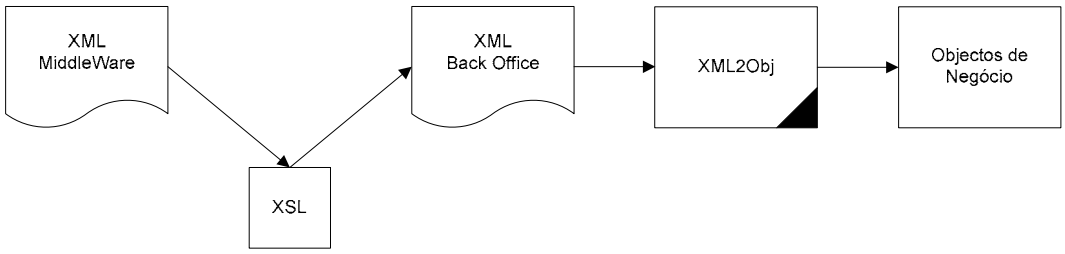


Figura 4 – Fluxo de integração de transacções

4.4.1 Conversão de XML para Objectos – XML2Obj

Para conseguir integrar a informação no sistema central, com base no XML em formato reconhecido pelos objectos de BackOffice, é activado um processo de abastecimento desses objectos.

Para cada objecto de negócio foi criada uma classe responsável pelo seu abastecimento a partir de um XML (por exemplo: à classe Apolice, corresponde uma classe XML2Apolice).

O processo é baseado num componente chamado de "XML Engine" que é responsável por efectuar o *parsing* do XML recorrendo ao DOM4J, e para cada uma das etiquetas e atributos existentes, fazer a respectiva instanciação e atribuição aos objectos XML2Obj. Por cada etiqueta que encontra no XML, o XML Engine procura identificar se essa etiqueta corresponde a uma classe ou não.

Caso corresponda, acrescenta a identificação dessa classe à *stack* que guarda a pilha de classes que vai sendo percorrida.

Se a etiqueta não corresponder a uma classe, significa que é um atributo da classe que está no topo da *stack*, sendo que neste caso o XML Engine irá invocar os métodos:

- ClasseTopoStack.strTag
- ClasseTopoStack.setTag
- ClasseTopoStack.endTag

Quando as validações de negócio (ao efectuar o *set* dos valores nos objectos) provocam erros, é invocada a classe ErrorContainer, sendo guardado o XPath da etiqueta ou atributo em causa, e a respectiva mensagem.

Se no final do processo existirem erros, é executado o processXMLLog, que efectua o *merge* do XML original com as mensagens de erro guardadas no ErrorContainer.

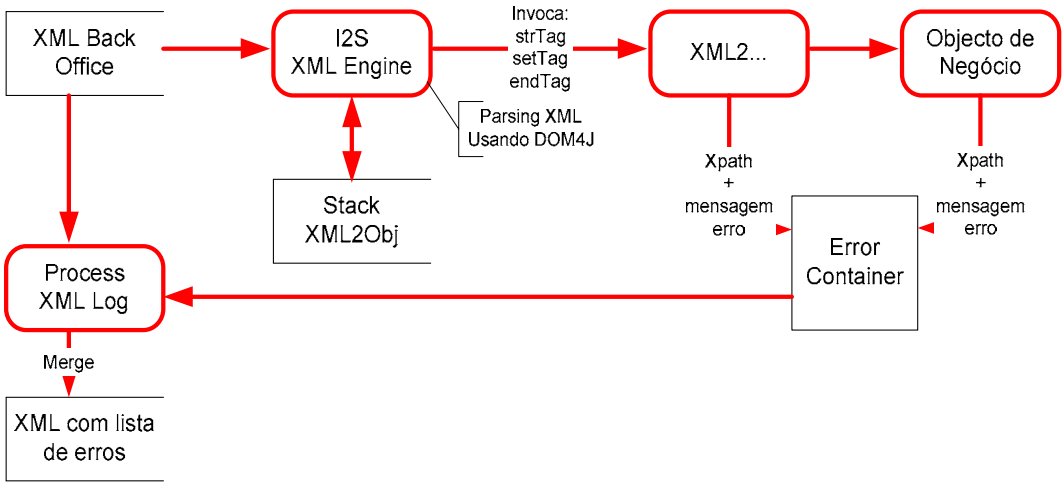


Figura 5 – Conversão de XML para Objectos

4.5 Consultas

Existe o processo inverso: com base em objectos de negócio, gerar XML que serve para alimentar o eGIS para fornecer consultas *online* aos clientes, com base na informação do sistema central da seguradora.

4.5.1 Conversão de Objectos para XML – Obj2XML

Para conseguir gerar um XML que correspondesse à informação existente em cada objecto de negócio, seguiu-se uma lógica semelhante à usada no processo inverso, de conversão de XML para objectos.

Assim sendo, o XML Engine coloca o objecto principal a consultar na *stack* de classes percorridas, sendo que para a classe que estiver no topo da *stack*, o XML Engine é responsável por invocar os seguintes métodos para cada um dos elementos do XML a criar, acrescentando os respectivos elementos ao XML de consulta:

- getChecked – responsável por verificar se um determinado elemento deve ser incluído no XML;
- getAttribute – responsável por criar os atributos do elemento no XML;
- getValue – responsável por colocar os valores dos elementos no XML a partir dos atributos da classe.

Caso o objecto principal consultado contenha referências a outros objectos, o objecto em causa é colocado no topo da *stack* e o processo é iniciado para a classe em causa.

Após ter percorrido toda a árvore de objectos a consultar, é efectuada a serialização do XML e enviado o XML resultado da consulta para o eGIS.

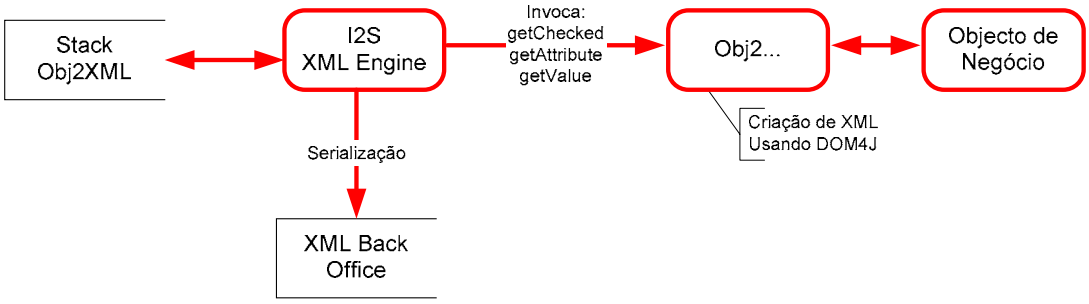


Figura 6 – Conversão de Objectos para XML

4.6 Arquitectura global

As soluções atrás mencionadas levaram à construção de um sistema com a seguinte arquitectura:

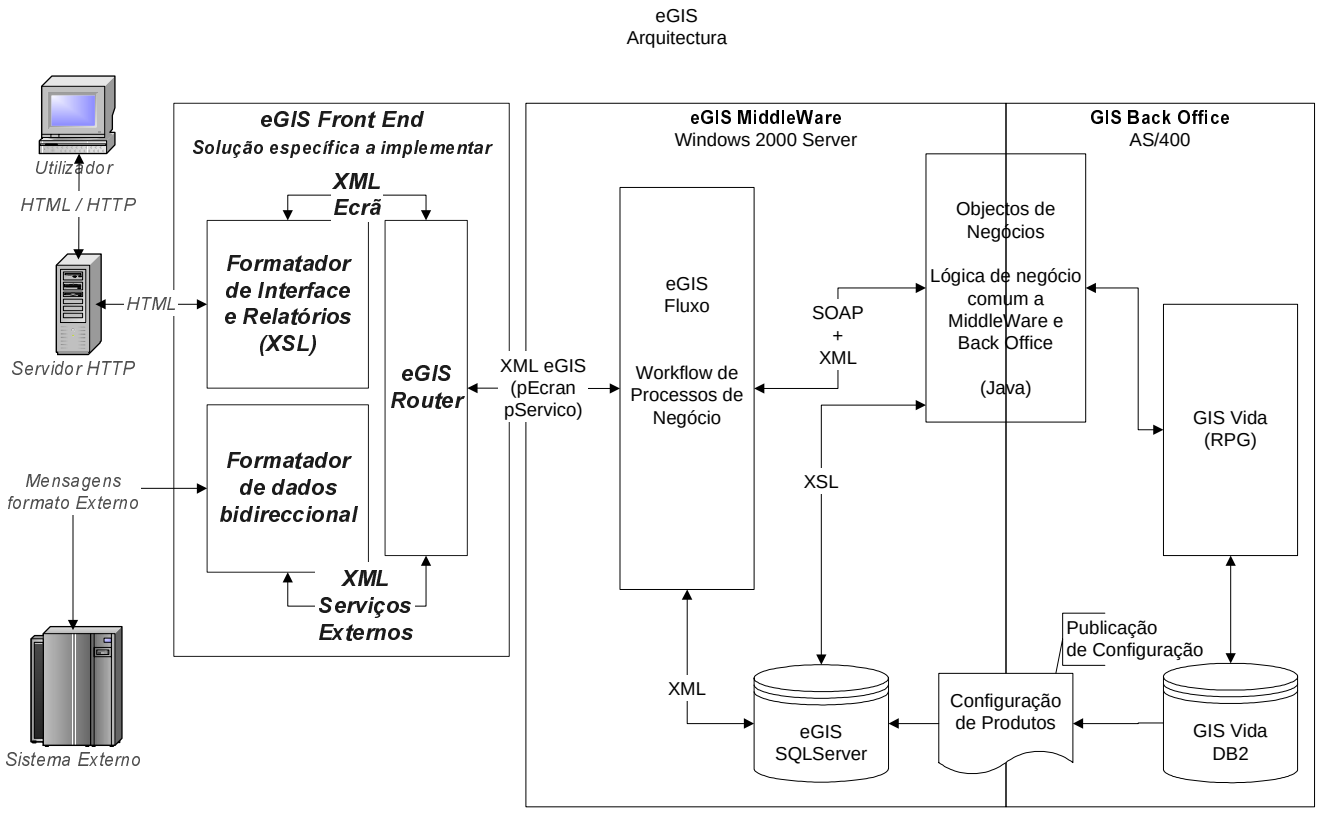


Figura 7 – Arquitectura eGIS

5 Conclusões

5.1 Solução comprovada

A solução encontra-se actualmente em funcionamento num dos maiores grupos financeiros nacionais, tendo taxas de utilização diárias de mais de um milhar de transacções (novos contratos ou alterações), e vários milhares de consultas, estando disponível em cerca de 800 balcões.

Outras soluções semelhantes estão a ser usadas em soluções integradores, incluindo componentes desenvolvidos por várias empresas e usando várias tecnologias.

5.2 Balanço

Os resultados da aposta no XML foram extremamente positivos, tendo esta tecnologia revelado ser extremamente interessante ao nível do potencial de *Enterprise Application Integration* e como complemento das tradicionais bases de dados relacionais.

Uma das grandes incógnitas iniciais do projecto estava relacionada com a maturidade e performance das tecnologias (XML, SOAP, etc...) para funcionarem em soluções empresariais de grande dimensão, principalmente em situações de sobrecarga de utilização. Tal como demonstrado no ponto 5.1, esta questão foi ultrapassada com total sucesso.

5.3 Acções futuras

Tendo já usado uma vasta gama de tecnologias associadas ao XML, nomeadamente DTD, XML Schemas (várias versões), XSL Patterns, XSLT, XPath, DOM, SAX, DOM4J e SOAP, continuamos a investigação das tecnologias existentes nesta área, de forma a conseguir daí tirar o maior partido possível.

Uma das áreas onde julgamos que esta tecnologia poderá ter particular interesse será na "persistência" da informação no sistema central, de forma a facilitar consultas que actualmente obrigam a pesquisas em base de dados relacionais altamente complexas e dispersas.

No seguimento da já longa tradição que a I2S mantém de ligação ao mundo académico, estão em curso alguns projectos ligados a mestrados da Universidade do Minho, em que estão envolvidos colaboradores da I2S, e onde estão a ser estudadas ferramentas para utilização de XML em bases de dados relacionais e soluções de bases de dados XML.

De entre as áreas actualmente em estudo, podemos enumerar:

- BPEL, Business Process Execution Language 4 Web Services, linguagem de especificação de fluxos de negócio que tem como mentores a IBM e Microsoft, pretendendo substituir as especificações WSFL da IBM e XLANG da Microsoft.
- Ferramentas de processamento de XML em bases de dados, nomeadamente o DB2 XML Extender
- Web Services
- Bases de dados XML – Apache XIndice

COMPARAÇÃO DE VÁRIAS ESTRATÉGIAS PARA ARMAZENAMENTO DE XML

Rui Cerveira Nunes
INETI – Departamento de Optoelectrónica, Estrada Paço Lumiar, 1649-038 Lisboa, Portugal
Email: rnunes@dop.ineti.pt

Miguel Mira da Silva
Instituto Superior Técnico – Departamento de Eng. Informática, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
Email: mms@dei.ist.utl.pt

Keywords: XML, Base Dados, Armazenamento, Integração

Abstract: À medida que aumenta a quantidade e complexidade de documentos XML que é necessário armazenar, aumentam também os requisitos dos sistemas de armazenamento para inserir, pesquisar e recuperar esses documentos. Por exemplo, já não é possível utilizar ficheiros do sistema operativo pois tornou-se inaceitável armazenar documentos XML sem sistemas de indexação que permitam pesquisar eficientemente grandes quantidades de dados neste formato. Este artigo foca-se nos resultados obtidos pelo nosso trabalho de investigação em estratégias de armazenamento de XML desde que publicámos os nossos resultados pela última vez [artigo aceite na ICEIS 2003]. Em particular, apresentamos neste artigo os resultados obtidos em experiências realizadas com vários produtos de bases de dados que armazenam XML em conjunto com uma implementação de referência que foi desenvolvida para efeitos de teste e protótipagem para um projecto. As grandes novidades deste artigo em relação ao artigo anterior são as diferenças fundamentais entre as várias estratégias de armazenamento, os tempos de inserção de grandes quantidades de documentos, e os testes com novas quantidades e novos tamanhos de documentos.

1. INTRODUÇÃO

À medida que a adopção do XML como formato de especificação de dados continua a ganhar popularidade de forma exponencial, surge cada vez mais a necessidade de mecanismos que permitam o seu armazenamento, pesquisa e recuperação. Apesar dos sistemas de ficheiros, bases de dados relacionais e sistemas de gestão orientados a objectos terem satisfeito os requisitos no passado, as quantidades crescentes de documentos XML impõem novos requisitos que – como nós descobrimos – não são suportados convenientemente por todos os sistemas que listam suporte para XML. O objectivo deste artigo é explorar (através de produtos representativos), as diferentes estratégias para armazenar XML. Este estudo foi desenvolvido parcialmente no contexto do projecto de investigação Europeu: *Euclid RTP11.13 – Realizing the Potential of Networked Simulations in Europe* [EUCLID 2001], no qual os requisitos são um pouco diferentes dos tipicamente encontrados em ambientes empresariais. O capítulo 2 irá conter uma breve introdução a esses requisitos. No capítulo 3 irão ser revistas as várias estratégias existentes para armazenar informação no formato XML, e quais são aparentemente as suas vantagens e desvantagens. No capítulo 4 é apresentado o sistema de referência desenvolvido assim como os produtos seleccionados para efeitos de teste. Finalmente no capítulo 5 são apresentados alguns resultados dos testes efectuados a esses produtos representativos de cada estratégia, onde mostraremos que nem sempre a prática bate certo com a teoria.

2. REQUISITOS ESPECIFICOS

Tipicamente em ambientes empresariais predomina o factor da grande quantidade de

instancias de dados do mesmo tipo, tais como: facturas, recibos, clientes, etc.

Normalmente estes tipos de dados tem uma estrutura relativamente simples, pelo que a melhor escolha para o seu armazenamento é em bases de dados relacionais que oferecem um desempenho impar nas pesquisas sobre grandes volumes de dados deste tipo. Tipicamente tambem são conhecidos os campos (ou colunas) desses mesmo dados que são normalmente pesquisáveis (tais como o identificador do clientes, da factura, nome do cliente, etc.) pelo que é possível definir índices específicos para optimizar as buscas.

No nosso projecto [EUCLID 2001], que consiste na definição de um processo para o desenvolvimento de ambientes sintéticos, tivemos a necessidade de desenvolver um repositório para informação codificada em XML, no qual temos um conjunto especifico de requisitos que podem ser sintetizados na seguinte lista:

1. O repositório tem que suportar dados semi-estruturados. A arquitectura do sistema apenas define os **campos de topo** dos documentos a armazenar.
2. Neste artigo apenas irão ser consideradas um subconjunto das buscas disponíveis no repositório: as denominadas **buscas por caracterização de entrada**.
3. Nesse contexto, a informação a pesquisar está localizada debaixo de um elemento especifico, o de caracterização.
4. Não se sabe a partida quais os caminhos ao longo do elemento de caracterização que irão ser frequentemente pesquisáveis.
5. O repositório **irá conter um elevado numero de entradas de um determinado tipo** (i.e. com a mesma estrutura definida por um DTD ou Schema).
6. O repositório irá conter **um elevado numero de tipos de entrada diferentes**.

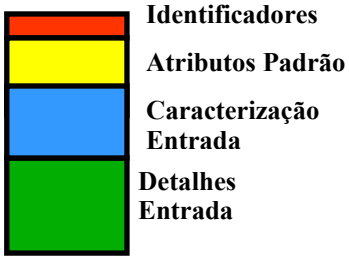


Figura 1 – Estrutura de uma entrada

Com este conjunto de requisitos, impõe-se uma revisão das estratégias existentes para o armazenamento de dados no formato XML.

3. ESTRATÉGIAS DE ARMAZENAMENTO XML

O mercado das bases de dados com suporte para XML está em crescendo. De acordo com [Staken 2001] os requisitos básicos para este tipo de base de dados são:

- define um modelo para um documento XML utilizando (no mínimo) elementos, atributos, PCDATA, e ordem dos elementos no documento;
- o documento XML é a unidade (lógica) fundamental de armazenamento.

No entanto, não é requerido que o modelo de armazenamento físico subjacente seja de um tipo especifico, podendo ser construído sobre uma base de dados relacional, hierárquica, orientada a objectos, ou mesmo até utilizando um sistema de armazenamento proprietário com ficheiros indexados.

As bases de dados que suportam XML podem ser segmentados em três categorias ou estratégias [Bourret 2002]:

- **Sistemas XML Nativos**;
- **Extensões XML** (tipicamente aos sistemas RDBMS e OODBMS já existentes);
- **Sistemas XML Virtuais** (consiste em middleware entre aplicações e bases de dados, tipo ODBC).

3.1 Sistemas XML Nativos

Existem produtos (tal como o Tamino da Software AG) que são desenvolvidos de raiz para suportar documentos XML. Normalmente estes produtos não são capazes de receber outro tipo de informação que não esteja no formato XML, mas são altamente optimizados para lidarem com este tipo de dados. Nem todas as bases de dados XML nativas são criadas por igual. Para os nossos testes seleccionamos duas amostras desta categoria: O **Tamino** [SOFTWAREAG 2002] e o **Xindice** [APACHE 2002].

3.2 Extensões XML e Sistemas XML Virtuais

As **extensões XML** são apenas uma evolução dos produtos RDBMS e OODBMS que foram adaptados para aceitar/exportar informação no formato XML, tal como no princípio dos anos 90 os sistemas relacionais foram estendidos para suportar funcionalidades dos sistemas orientados a objectos.

Os **sistemas XML virtuais** oferecem uma solução atractiva para armazenar XML em bases de dados existentes, sem acrescentar ou alterar absolutamente nada nessas bases de dados. Servem ainda para exportar em formato XML os dados relacionais já existentes, nomeadamente dados tradicionais como por exemplo encomendas. Na prática, a diferença entre estas duas categorias reside no nível de integração entre a própria extensão e a base de dados. Sendo que muitas vezes a extensão é desenvolvida por outra entidade que não o fabricante da base de dados.

3.3 Implementação de Extensões XML

Actualmente, a maioria dos sistemas que armazenam XML são baseados quer na aproximação de extensões XML ou então nos sistemas virtuais. Ou seja, utilizam uma base de dados relacional ou orientada a objectos para armazenar os documentos XML. Devido à sua importância, vamos explicar mais em pormenor como se pode fazer o mapeamento dos dados XML quer para o modelo relacional, quer para o modelo orientado a objectos.

3.3.1 Mapeamento para o Modelo Relacional

[YoshiKawa 2001] distingue as possibilidades de mapeamento de dados XML para o modelo relacional em duas grandes categorias:

- Mapeamento por Estrutura (em que o esquema das tabelas relacionais é dependente do DTD).
- Mapeamento por Modelo (em que o esquema das tabelas relacionais é independente do DTD).

3.3.1.1 Mapeamento por Estrutura

Nesta aproximação, o desenho da base de dados relacional é baseado na estrutura do documento XML de entrada, tipicamente definida por um DTD ou XML Schema.

Dado um DTD arbitrário, é possível [Williams 2000] seguir 18 regras algorítmicas para chegar a um modelo de tabelas relacionais. Estas tabelas serviriam de suporte ao armazenamento de qualquer documento XML que respeite a estrutura definida no DTD original. Existem produtos de *middleware* que tentam automatizar este processo tais como o Microsoft SQLXML [MICROSOFT 2002] e o Oracle XSU [ORACLE 2002].

É importante referir que nem todos os produtos suportam todas as características desejáveis, não sendo capazes de processar DTD com complexidade mais elevada. Idealmente, o *middleware* deverá ser capaz de transformar pesquisas numa linguagem tipo de XML (XPath) em instruções de SQL que serão enviadas para o produto RBDMS subjacente.

Para o nosso desenvolvimento, a hipótese de mapeamento por estrutura foi rejeitado, dado que isso implica que exista uma grande confiança na capacidade do *middleware* que efectua o mapeamento de estruturas XML arbitrariamente complexas para tabelas relacionais. A solução da Microsoft [MICROSOFT 2002] por exemplo não conseguiu processar com sucesso o nosso DTD complexo para testes, dado que não suporta a situação em que um elemento é filho de elementos diferentes e que estejam encadeados, o erro obtido foi o seguinte: '*The same table tableXXX can not be child table in two nested relations*'.

3.3.1.2 Mapeamento por Modelo

A segunda aproximação para o mapeamento de dados XML no modelo relacional consiste no chamado mapeamento por modelo [YoshiKawa 2001]. Em que as tabelas são criadas, não para suportar uma estrutura de um DTD em particular, mas sim para suportar todo o modelo do XML (ou transformação equivalente) em si. Por outras palavras, a partir do modelo XML (constituído por elementos, atributos, CDATA, etc.) são criadas tabelas genéricas para o suportar.

Em [Florescu 1999], foi feito um estudo comparando diferentes estratégias para construir esquemas relacionais genéricos de suporte ao modelo XML.

Um esquema de mapeamento neste contexto é constituído pelo conjunto de tabelas que serão criadas na base de dados relacional, pelos respectivos índices que irão ser construídos, e pela forma como são armazenadas os elementos internos do documento (objectos), os atributos (edges) e os valores de texto (folhas). Nesse trabalho são analisadas quatro variantes sobre como representar os atributos e duas variantes sobre como representar valores de texto, resultando em 8 combinações de implementação.

O problema desta implementação reside no tempo de reconstrução e recuperação de um documento, que no caso de uma instância de 79 MB demora entre 32 minutos a 1h e 40 minutos para diferentes implementações.

Dado que este tempos desta ordem são inaceitáveis para os requisitos do nosso projecto, decidimos implementar um sistema de referencia de mapeamento por modelo que utiliza uma base de dados relacional e o sistema de ficheiros. Este sistema servirá como **referência** e será descrito em detalhe na secção seguinte do artigo.

3.3.1.3 Armazenamento em Coluna

Para alem dos dois modelos de mapeamento para relacional apresentados nas secções anteriores, alguns dos vendedores de produtos de bases de dados relacionais mais conhecidos estão a optar por estenderem os seus produtos oferecendo um tipo de dados de coluna capaz de receber um documento XML e efectuar pesquisas sobre caminhos específicos e até suportar a definição de índices sobre caminhos XPath específicos.

Iremos seleccionar para os testes, como representante desta classe, uma base de dados relacional de topo de gama, que por impedimento não podemos divulgar o nome. Esta base de dados suporta este tipo de armazenamento através da utilização de uma coluna especial para armazenar XML.

3.3.2 Mapeamento para Bases de Dados OO

Enquanto o mapeamento de XML para bases de dados relacionais é uma tarefa complexa, dado o desalinhamento de impedância existente entre as relações hierárquicas complexas entre os elementos (e atributos) constituintes dos documento XML e os modelos mais planos de tabelas e colunas das bases de dados relacionais. O modelo de dados XML mapeia melhor num modelo orientado a objectos. Alguns vendedores de bases de dados orientadas a objectos a incluir extensões aos seus produtos de forma a proporcionar suporte de XML.

Iremos seleccionar também um produto de base de dados orientado a objectos que possua suporte para XML para a nossa analise detalhada, neste caso o **Ozone** [OZONE 2000].

4. PRODUTOS REPRESENTATIVOS

4.1 Sistema de Referência

Para ter uma base de comparação, optámos por desenvolver um sistema de base adaptado aos requisitos do sistema. Este sistema utiliza mapeamento de XML para relacional (por modelo) e servirá como base para comparação dos restantes produtos.

O sistema de referência utiliza *parsing* SAX [SAX 2001] para efectuar de uma forma célere e eficiente todos os elementos por baixo do elemento de caracterização, e armazena os dados de uma forma não normalizada em tabelas relacionais utilizando JDBC. Existe obviamente um factor de mérito entre a escolha de um modelo para as tabelas que minimize o espaço ocupado pelos dados e o tempo posterior de busca. No modelo de referência optamos por minimizar o tempo de busca, mantendo os dados não normalizados.

Dado que a informação que o repositório tem que armazenar é semi-estruturada. Os principais campos de informação são a caracterização e os detalhes. Apenas armazenamos a informação de caracterização nas tabelas relacionais. Sendo que o resto é armazenado num ficheiro externo.

Esta opção reduz o problema da reconstrução de documentos das bases de dados relacionais a custa da duplicação de espaço, e permite assim comparar as restantes soluções com tempos equivalentes ao de uma leitura de ficheiro.

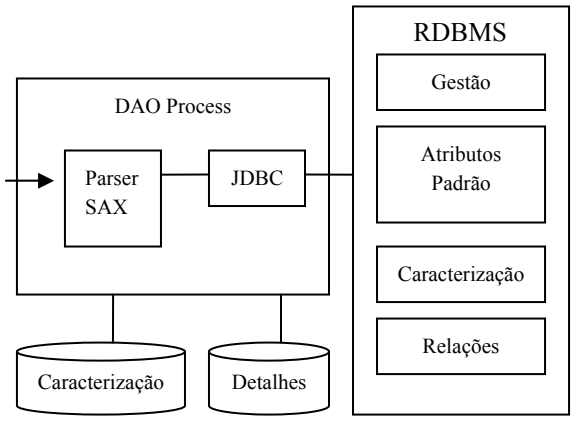


Figura 2 – Sistema de Referência

4.1.1 Notas de Implementação

O sistema de referência utiliza JDBC para conseguir alguma portabilidade entre sistemas de gestão de bases de dados relacionais diferentes.

Apesar de só por si uma implementação em JDBC não garantir que a aplicação funcione sobre qualquer base de dados que suporte este padrão, desde cedo no desenvolvimento tentámos manter o sistema de referência compatível com as seguintes bases de dados relacionais:

- Base de dados comercial de topo de gama.
- SAP/DB Server 7.3.0.21.

Utilizando o menor denominador comum sempre que uma dada funcionalidade, característica, ou opção de implementação não funcionava em um dos sistemas. Infelizmente, descobrimos numa fase posterior (fase de testes) que existe uma limitação na criação de índices do produto SAP/DB, o que impede a indexação total dos elementos de XML armazenados pelo nosso modelo. Esta limitação não ocorre no sistema de topo de gama.

4.2 Outros Produtos Testados

Adicionalmente, seleccionamos também um conjunto de produtos comerciais ou *freeware* representativos de algumas das restantes categorias aplicáveis para efeitos de teste:

- Apache Xindice [Base de dados XML nativa *freeware*]
- Tamino da Software AG [Base de dados XML nativa comercial]
- Base de dados relacional de topo de gama com extensões XML [Representante da categoria de armazenamento em coluna]
- Ozone OODB [Base de dados orientada a objectos com extensões XML]

Uma análise mais detalhada das funcionalidades suportadas por estes produtos, assim como das suas interfaces de acesso pode ser encontrada em [Nunes 2003].

5. ANÁLISE DE DESEMPENHO

Nesta secção analisa-se em maior detalhe, o desempenho dos produtos seleccionados. Para o efeito descreve-se primeiro o ambiente de testes desenvolvido para exercitar os produtos referidos, sendo apresentado posteriormente os resultados desses testes.

5.1 Desenvolvimento de um Framework de Testes

O ambiente de testes foi desenvolvido em Java e utiliza o servidor aplicacional *JBoss*. Este *framework* de testes encontra-se documentado em maior detalhe em [Nunes 2002]. O objectivo é a execução de casos de teste genéricos (que consistem em classes Java). As entidades que executam um determinado caso de teste são bipartidas num componente cliente e num componente servidor dentro do servidor aplicacional. Apresenta-se de seguida o fluxo de execução típico:

- É definido o numero de clientes que irão executar.
- É criado um objecto da classe Monitor para sincronizar as *threads* cliente.
- Cada *thread* cliente executa o método do caso de teste **clientInit** () que dependendo do caso de teste pode consistir por exemplo numa leitura de dados do sistema de ficheiros.
- Cada *thread* cliente, inicializa o seu parceiro dual no lado do servidor, transferindo para lá o caso de teste e os dados respectivos. Este parceiro dual consiste num *statefull session bean*.
- O objecto cliente mantém uma referencia para o componente servidor que fica pronto e em espera.
- Após todo o processo de inicialização de todas as instancias estar concluída, é iniciada a execução dos casos de teste, quer em paralelo, ou série, sendo este um parâmetro de controlo do ficheiro de configuração global.
- A *thread* cliente indica ao seu respectivo parceiro dual no servidor para inicializar a execução.
- Assim que o lado servidor termina a execução, notifica a *thread* cliente, que por sua ver notifica o **Monitor**.
- Quando o **Monitor** detecta que todas as *threads* terminaram a execução, é iniciado o processo de *debriefing*.
- No processo de *debriefing*, cada *thread* contacta o seu parceiro dual do lado do servidor para obter a estrutura de dados chamada **TimerLogger** que regista métricas de tempo ao longo da execução do caso de teste.

- São calculadas as métricas para cada evento, tais como os tempos mínimos, tempos máximos, tempos médios, etc.

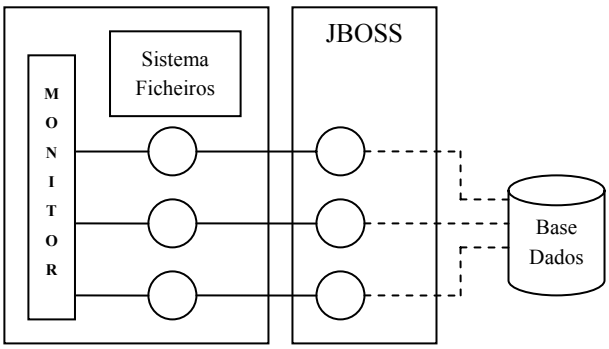


Figura 3 – Framework de Testes

De notar que todas as operações realizadas durante a execução de um caso de teste, são executadas contra uma interface de base de dados comum, que contem métodos genéricos tais como:

- Inserir documento
- Recuperar documento
- Actualizar documento
- Remover documento
- Efectuar Pesquisa
- Etc

É utilizado o *bridge pattern* [Gamma 1995] para mapear estes métodos genéricos em implementações concretas para os vários produtos de bases de dados testados.

5.2 Pesquisa de Documentos

5.2.1 Condições de Teste

Este teste consiste na execução em série por 10 clientes, de uma pesquisa de XPath efectuada contra o framework. É mostrado o tempo médio de execução destes 10 clientes. Antes de cada execução a base de dados é populada com um numero variável de documentos gerados sinteticamente e que diferem no valor do caminho especifico que está a ser pesquisado (ou seja, cada documento inserido tem o caminho especifico a ser pesquisado com um valor diferente).

Apenas um desses documentos conterá o valor que está a ser pesquisado. Neste teste a dimensão dos documentos é de cerca de 4~5 KB.

5.2.2 Resultados

Os resultados dos tempos desta busca já foram parcialmente publicados em [Nunes 2003] para alguns dos produtos, pelo que aqui se opta por fazer apenas uma revisão das principais conclusões, assim como da análise do desempenho do novo sistema de referência.

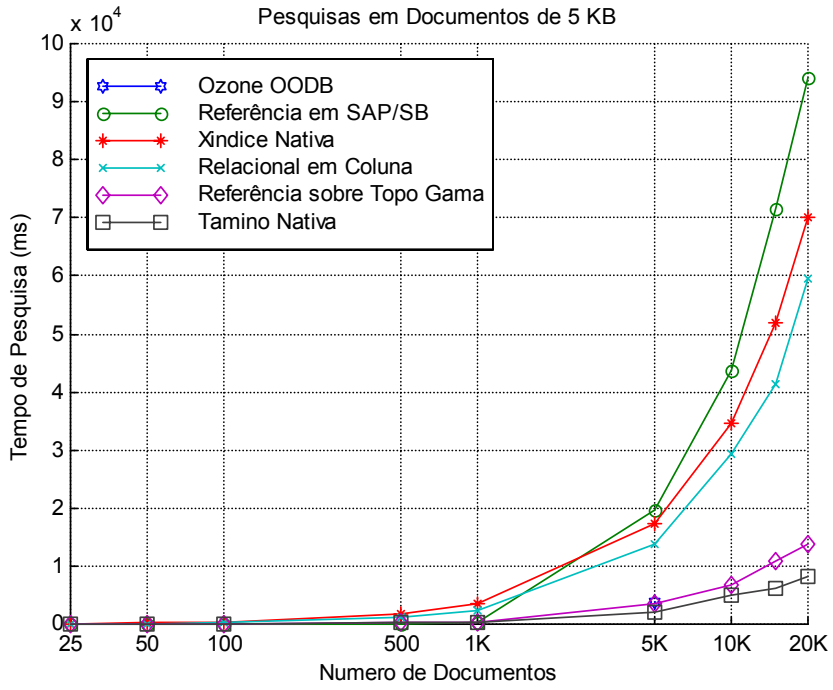


Figura 4 – Tempos de Pesquisa para Documentos de 5 KB

O Xindice e o sistema de armazenamento de coluna tem um desempenho similar. Apesar destes produtos suportarem indexação de caminhos XPath específicos, essa funcionalidade não foi utilizada devido aos nossos requisitos iniciais, em que não se conhece à partida quais os caminhos específicos mais utilizados nas buscas. No produto Xindice, foi testada informalmente a opção de indexar todos os elementos à entrada, e o resultado das buscas é ligeiramente pior do que sem indexação (a indexação apenas dos caminhos específicos é obviamente bastante mais rápida).

Por omissão, o Tamino indexa todos os elementos à entrada utilizando o seu motor proprietário baseado na base de dados hierárquica Adabas, e como resultado o tempo para localizar elementos arbitrários dentro dos documentos XML é de longe o melhor do conjunto.

Em relação à base de dados orientada a objectos, Ozone, e apesar de listar suporte para XML nas suas funcionalidades, ela apenas permite efectuar buscas dentro de um objecto desenvolvido para encapsular um documento, chamado XML *container*. A nossa aproximação foi a de utilizar um *hashmap* para gerir estes conjuntos de XML *containers* que funciona bem enquanto o numero de documentos é relativamente pequeno, mas cresce rapidamente para mais de 5000 documentos devido a passivação/activação de objectos dentro do servidor. Apenas apresentamos no gráfico o ponto correspondente aos 5000 documentos.

Em relação ao sistema de referência, a implementação nos dois RDBMS é radicalmente diferente. A indexação efectuada pelo sistema de topo de gama é bastante superior à efectuada pelo sistema SAP/DB.

De facto, em termos de buscas, a implementação sobre o sistema de topo de gama apesar de inferior ao Tamino, tem um desempenho bastante aceitável, ocupando confortavelmente o segundo lugar.

Esta analise fornece uma boa ideia sobre o tempo que cada um destes sistemas demora até localizar informação especifica no interior de documentos XML. A secção seguinte do artigo, analisará em maior detalhe, qual é exactamente a penalização a pagar em termos de desempenho de inserção de documentos (resultante da indexação efectuada à entrada) para que as buscas posteriores sejam tão rápidas.

5.3 Inserção de Documentos

5.3.1 Condições de Teste

Este teste consiste no registo do tempo de execução de todas as pré-inserções antes das fases dos testes posteriores de pesquisa. Verificamos em média o que tempo que cada base de dados demora até que o processo de inserção esteja concluído (incluindo possíveis indexações) para documentos que variam de dimensão entre 5 KB e 5 MB

5.3.2 Limitações Identificadas

Como pri meira nota, é importante referir que o sistema de armazenamento em coluna tem como mecanismo de acesso ao seu sistema, instruções de SQL estendidas do tipo objecto-relacional.

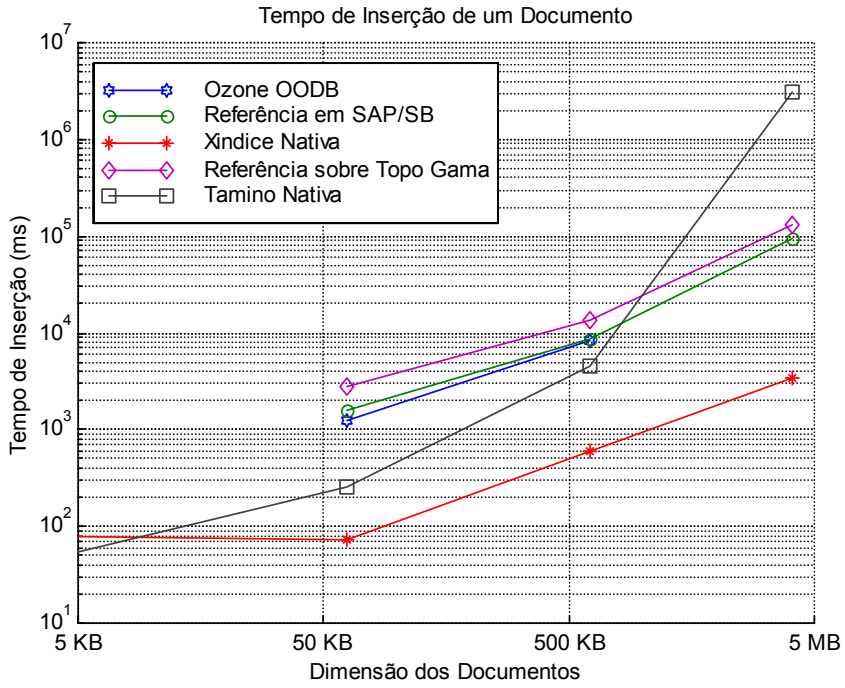


Figura 5 – Tempos de Inserção Variando a Dimensão dos Documentos

A dimensão máxima de uma *string* de SQL para este sistema é de 4000 caracteres, pelo que não é trivial a inserção de documentos de XML maiores do que esta dimensão.

A base de dados Xindice também revelou problemas de execução no servidor para documentos superiores a 5 MB.

5.3.3 Resultados

Em relação ao sistema de referência implementado sobre as duas bases de dados relacionais, observa-se que existe mais trabalho de indexação efectuado pela base de dados de topo de gama do que no sistema SAP/DB, mas esse processamento é largamente compensado nas fases posteriores de buscas.

O sistema Xindice não efectua nenhum tipo de indexação a entrada por defeito e por esse facto é o sistema mais rápido a inserir documentos.

Apesar de efectuar indexação a todos os elementos, o Tamino tem um desempenho excelente de inserção até documentos da ordem dos 500 KB. Para documentos de uma ordem de grandeza superior, ele revela graves problemas dado que o tempo de inserção ronda os 51 minutos! comparativamente por exemplo com os 2,18 minutos da implementação de referência sobre a base de dados topo de gama, e dos 3,4 segundos do sistema Xindice.

Este facto pode indiciar que o sistema utiliza uma implementação de DOM para efectuar o *parsing* ao XML recebido, antes de escrever a informação no seu motor hierárquico (esta hipótese não foi testada). No entanto essa implementação é bastante optimizada, dado que um *parser* de DOM típico não tem um desempenho tão bom com ficheiros da ordem de 500 KB.

De notar que o ficheiro de 5 MB utilizado tem uma estrutura muito complexa. É provável que se o documento de XML tivesse a mesma dimensão, mas fosse constituído por elementos com blocos maiores de CDATA, numa estrutura hierárquica mais simples, o desempenho tivesse sido melhor (Esta hipótese também não foi testada).

6. CONCLUSÕES

Neste artigo foram revistas as estratégias existentes para armazenar informação em XML e foram seleccionados produtos representantes das categorias relevantes que fossem adequados aos requisitos do nosso projecto.

A grande conclusão é que: Ao contrario das bases de dados relacionais em que provavelmente é possível identificar produtos que sejam melhores do que os restantes em todas as categorias, no que concerne ao armazenamento de informação em XML isso não acontece actualmente.

Terá que ser efectuada uma analise detalhada aos requisitos da aplicação em causa para se determinar qual das aproximações e estratégias se adequa mais ao cumprimento desses mesmos requisitos.

Por exemplo, se a aplicação envolve a escrita frequente de documentos de grande dimensão e complexidade, então provavelmente o Tamino não será a escolha ideal, mas se esses mesmos documentos forem para ser escritos apenas uma vez e tipicamente existam apenas pesquisas então o Tamino supera todos os outros produtos.

Nos requisitos do nosso projecto, não se sabe a partida quais os caminhos dentro do XML que serão pesquisados mais frequentemente, o que não acontece nas aplicações empresariais típicas em que existem caminhos/campos específicos tal como o identificados da factura ou do cliente que são frequentemente utilizados como a chave de pesquisa.

Nessas condições, então talvez o armazenamento por coluna, ou a utilização do Xindice com a definição de índices específicos seja uma aproximação mais adequada.

Para o nosso projecto, o requisito de existirem muitas estruturas diferentes para os documentos XML não aconselha a utilização do mapeamento por estrutura, mas em aplicações em que exista apenas um DTD conhecido a partida, então poderá ser compensador o investimento de tempo na definição de uma mapeamento especifico para esse DTD.

Cada responsável pelo desenvolvimento de um projecto deverá identificar quais as interacções típicas das suas aplicações com os repositórios de XML e efectuar um compromisso de escolha ao longo das curvas de desempenho dos vários produtos, em conjunto com requisitos adicionais tais como integridade transaccional, preço, etc.

REFERÊNCIAS

[APACHE 2001a] Staken, K. 2001-2002, *Xindice Users Guide 0.7*, The Apache Software Foundation 2001-2002. Disponível em: http://xml.apache.org/xindice/UsersGuide.html

[APACHE 2001b] The Apache Software Foundation 2001-2002, *Xindice Developers Guide 0.7*,. Disponível em: http://xml.apache.org/xindice/DevelopersGuide.html

[APACHE 2002] Apache Xindice. Disponível em : http://xml.apache.org/xindice/

[Bourret 2002] Bourret, R. 2002. *XML Database Products*. Disponível em: http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[EUCLID 2001] Euclid RTP11.13 – *Realising the Potential of Networked Simulations in Europe* HomePage. Dísponível em: http://www.euclid1113.com

[Florescu 1999] Daniela Florescu, Donald Kossman, 1999, *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*, INDRIA

[Gamma 1995] Gamma E. et al, 1995, *Design Patterns, Elements of reusable Object-Oriented Software*, Addison-Wesley

[MICROSOFT 2002] Microsoft Corporation. *Microsoft SQL Server XML Extensions (SQLXML) 3.0 SP1. Disponível em:* http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/824/msdncompositedoc.xml

[MICROSOFT 2003] Microsoft Corporation. *Microsoft SQL Server Home*. Disponível em: http://www.microsoft.com/sql/default.asp

[Nunes 2002] Nunes, R., da Silva, M., *Comparação de Bases de Dados para Armazenamento XML*, INESC-ID, Relatório Técnico 007/2002, Lisboa, 2002.

[Nunes 2003] Nunes, R., da Silva, M., 2003. *A comparison of database systems for storing XML documents*. Accepted to the 5th International Conference on Enterprise Information Systems (ICEIS 2003), April 2003, École Supérieure d' Électronique de l' Ouest, Angers, France

[ORACLE 2002] Oracle Corporation. *Oracle XML SQL Utility – XSU*. Disponível em: http://technet.oracle.com/tech/xml/oracle_xsu/content.html

[OZONE 2000] Ozone, the Open Source Java ODBMS. Disponível em: http://www.ozone-db.org

[SAP 2001] SAP/DB - The FREE Enterprise Open Source Database. Disponível em:http://www.sapdb.org

[SAX 2001] SAX. *The Simple API for XML*. Disponível em: http://www.saxproject.org/

[Staken 2001] Staken, K., 2001. *Introduction to Native XML Databases.* Disponível em: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[SOFTWAREAG 2002] Software AG, 2002. *Tamino Documentation,* CD ROM Tamino XML Server Version 3.1.1

[Williams 2000] Williams, K. et al, 2000. *Professional XML Databases*, Wrox Press.

[YoshiKawa 2001] YoshiKawa, M., Amagasa, 2001. XRel: *A path-based approach to storage and retrieval of XML documents using relational databases*. Nara Institute of Science and Technology.

Abordagens para armazenamento de metadata em Data Warehouses usando o XML

Hugo Alhandra Miguel Mira da Silva

alhandra@mail.com mms@dei.ist.utl.pt

Instituto Superior Técnico, Portugal

Resumo

Este artigo procura descrever os problemas que actualmente existem na integração dos vários componentes de uma Data Warehouse. É proposto um modelo que utilizando CWM e uma base de dados nativa XML simultaneamente resolve os problemas de armazenamento do CWM, e troca de metadata.

Palavras Chaves: Data Warehouses, Metadata, CWM, XML, e Bases de dados

1. INTRODUÇÃO

Uma Data Warehouse é um dos sistemas de informação mais importantes de uma organização. É quase impensável hoje em dia, imaginar que uma organização de média ou grande dimensão não possua um sistema que integre os vários sistemas informáticos que a constituem, permitindo ao gestor ter uma visão global do estado da organização. A informação disponibilizada por uma Data Warehouse é de tal modo valiosa que são feitos avultados investimentos para construir, manter e usar esse sistema.

Uma Data Warehouse é constituída por vários componentes (Ferramentas de modelação, ETL, Ferramentas de análise de dados e produção de relatórios, entre outras). Cada um desses componentes necessita armazenar e utilizar metadata para cumprir os seus objectivos. Além disso cada um desses componentes necessita trocar metadata com o componente seguinte na "linha de montagem" que vai desde a importação dos dados dos sistemas operacionais até à disponibilização da informação aos utilizadores finais.

Com o amadurecimento das Data Warehouse, verificou-se que existiam dificuldades de integração na metadata usada por cada uma das ferramentas. Algumas dificuldades eram por exemplo as seguintes: saber a data de carregamento da Data Warehouse para exibir num relatório para o utilizador final, determinar os tipos de dados presentes nos sistemas operacionais que alimentavam a Data Warehouse, ou então produzir um documento actualizado com as regras de mapeamento utilizadas para carregar a Data Warehouse. Esses e outros problemas levaram à criação de algumas normas de metadata para Data Warehouse, nomeadamente o CWM e o OIM.

Este artigo apresenta os resultados obtidos no nosso trabalho de investigação sobre armazenamento de metadata em Data Warehouse. Focaremos a nossa atenção no armazenamento das regras de mapeamento (nomeadamente as utilizadas nas ferramentas de ETL). A metadata será armazenada utilizando o CWM sendo um foco deste trabalho determinar se esses metadados deverão ser armazenados numa base de dados relacional ou numa base de dados nativa XML.

A solução tradicional consiste em armazenar a metadata de cada uma das ferramentas de modo proprietário e não partilhado. Assim neste artigo propomos uma nova arquitectura para Data Warehouse que permita uma mais fácil integração dos seus componentes.

Na próxima secção explicitamos mais claramente o papel a metadata na Data Warehouse.

2. DATA WAREHOUSES E METADATA

2.1 Data Warehouses

De acordo com [Bert97] "Uma Data Warehouse é um ambiente. É uma arquitectura de um sistema de informação que fornece aos utilizadores dados actualizados e históricos que são difíceis de aceder ou que estão presentes num conjunto de relatórios de um sistema operacional". Uma Data Warehouse é composta de vários componentes conforme veremos nos próximos parágrafos.

Primeiro, necessitamos de ferramentas de modelação que são usadas para desenhar o modelo de dados da Data Warehouse. Tradicionalmente os modelos em estrela ("*Star Schemas*") e floco-de-neve ("*Snowflake*") são os mais comuns numa Data Warehouse. Os esquemas em estrela são constituídos por uma tabela de facto e várias tabelas de dimensão; cada dimensão representando uma perspectiva para análise. Tendo em conta que uma Data Warehouse mantém informação histórica, quase sempre teremos uma dimensão tempo. Para melhorar o desempenho do sistema, as hierarquias (que porventura existam) são previamente "desdobradas" em todas as combinações possíveis, minimizando deste modo o número de "*joins*" necessários para reconstituir a informação armazenada. Os esquemas em floco-de-neve são basicamente esquemas em estrela em que as hierarquias estão normalizadas.

Após o desenho do modelo de dados é necessário implementá-lo, o que é conseguido usando ferramentas ETL (Extract Transform and Load). Estas ferramentas necessitam conhecimento sobre o sistema operacional de onde provêm os dados e conhecimento acerca das tabelas presentes na base de dados de Data Warehouse. As ferramentas ETL utilizam metadata (contendo a informação de mapeamentos) para gerar automaticamente um programa de carregamento das tabelas daquela Data Warehouse.

Assim que a base de dados onde reside a Data Warehouse é correctamente actualizada e a qualidade da informação é verificada (possivelmente usando ferramentas de análise de qualidade de dados) é possível passar à fase de análise de dados. Aqui podem ser usadas ferramentas OLAP para gerar cubos multi-dimensionais que permitem o estudo da informação de várias perspectivas. Numa Data Warehouse típica serão também usadas algumas ferramentas para gerar relatórios. Alguns utilizadores mais experientes submeterão interrogações directamente para o SGBD, permitindo-lhes descobrir correlações entre os dados. Em Data Warehouses mais sofisticadas serão utilizadas também ferramentas de Data Mining que de modo heurístico ajudarão a descobrir as relações entre os dados.

Por último numa Data Warehouse real, é imprescindível a existência de algum tipo de administração e monitorização do sistema. Isso pode ser feito através de scripts ou através de ferramentas comerciais.

Conforme veremos nas secções seguintes, cada um dos componentes de uma Data Warehouse utiliza um determinado tipo de metadata. Além disso existe necessidade de trocar metadata entre cada um desses componentes. Assim surgirão dois problemas: como armazenar a metadata utilizada em cada um desses componentes e como trocar a informação entre os vários componentes da Data Warehouse.

2.2 Metadata

A metadata ganhou uma nova importância ultimamente com os sistemas de suporte à decisão porque, ao contrário dos sistemas operacionais em que os operadores dos sistemas executam as tarefas de modo rotineiro, nos sistemas de apoio à decisão os utilizadores muitas vezes têm de analisar os dados pela primeira vez e de modo mais criativo. Por isso, a metadata pode ser uma ajuda valiosa para entender os dados com que se está a lidar. Além disso como a Data Warehouse integra dados de muitas fontes, pode acontecer que não haja ninguém que conheça todos os pormenores do sistema, uma vez que as pessoas conhecem bem apenas partes do sistema.

Há duas facetas importantes na metadata de qualquer sistema de apoio à decisão. A metadata de negócio (ou de utilizador) e a metadata técnica.

A metadata de negócio procura descrever os dados do negócio de modo independente da tecnologia que a suporta. Este tipo de metadata é usado pelos analistas do negócio e pelos

utilizadores (não técnicos) para obterem uma descrição das entidades informacionais. Ajuda-os a perceber, localizar, e aceder à informação que está na Data Warehouse. Um exemplo de metadata de negócio seria a descrição do atributo TOT_EMP como "descreve o número total de empregados em determinada sucursal da nossa empresa". Um papel muito importante deste tipo de metadata é aumentar a confiança dos utilizadores nos dados!

A metadata técnica, por outro lado, fornece informação aos programadores e utilizadores técnicos sobre o sistema de apoio à decisão e os sistemas operacionais, para além do que será necessário fazer para manter o sistema a funcionar. Esta metadata é um grande auxílio na resolução de muitos problemas técnicos.

Passamos agora a descrever alguns exemplos de metadata.

2.3 Exemplos de Metadata

A metadata técnica contém, portanto, informações sobre a fonte dos dados, sobre o destino dos dados; e sobre as regras que foram usadas para a extracção, limpeza, e transformação dos dados. A metadata técnica é normalmente mantida pelos SGDBs (por exemplo, na recolha de estatísticas), pelas ferramentas de ETL (por exemplo regras de transformação), e pelo administrador da Data Warehouse (por exemplo na definição de horários de carregamento).

Por outro lado a metadata de negócio procura responder às seguintes perguntas:

- "Onde..?" é um tipo de pergunta que se pode responder recorrendo a uma interface visual que permita pesquisas na metadata.
- "Como…?" é um tipo de pergunta cuja resposta se poderá ser obtida através das regras de negócio que estejam armazenadas como parte da Metadata

Após termos analisado nesta Secção, os diferentes tipos de metadata existente, passamos agora a analisar quais as soluções actuais para armazenamento de metadata.

3. SOLUÇÕES ACTUAIS PARA ARMAZENAMENTO DE METADATA

3.1 Repositórios

Para a maior parte das organizações, um repositório de metadata único (abordagem centralizada) é suficiente para suportar os requisitos de todos os departamentos dentro da organização. A aproximação centralizada fornece grandes vantagens relativas à administração e partilha de metadata entre aplicações porque é muito mais fácil manter a uniformidade e consistência da metadata desta forma.

Mas para algumas organizações poderá ser necessário utilizar vários repositórios de metadata. Por exemplo, no caso de holdings, poderia existir um repositório de metadata por empresa do grupo. A utilização desta arquitectura distribuída implica, no entanto, a consolidação dos repositórios de nível inferior, e portanto mais possibilidades de inconsistência na metadata e mais esforço de administração. Devido a esses factores, a nossa visão é que esta abordagem distribuída deverá ser evitada.

Uma razão extra para a escolha da abordagem centralizada é o próprio conceito de Data Warehouse: uma base de dados central que apresenta uma visão coerente e global da informação presente numa organização. Uma vez que os dados são consolidados na Data Warehouse faz todo o sentido que a metadata sobre esses dados também seja consolidada numa base de dados central.

3.2 Normas de Metadata

Em geral, a troca de dados entre aplicações ainda é muito difícil quando comparada por exemplo, com o desenvolvimento dessas mesmas aplicações onde existem inúmeras metodologias e ferramentas. No mundo das Data Warehouses este problema é particularmente grave. Para resolver (ou, melhor dizendo, atenuar) este problema utilizam-se normalmente uma série de tecnologias proprietárias para ligar um sistema directamente a outro, uma solução ad-hoc que não é escalável. O ideal seria existirem normas para todos os tipos de dados.

Para ajudar a resolver estes problemas de integração entre ferramentas, tanto o MDC (Meta Data Coalition) como a OMG (Object Management Group) criaram normas para partilhar metadata entre diferentes fabricantes e nesta Secção vamos analisar várias normas específicas para depois atacar finalmente o problema do seu armazenamento.

O CWM [CWM](Common Warehouse MetaModel) foi desenvolvido pela OMG [OMG] com o objectivo de gerir diversos tipos de dados empresariais para facilitar a sua integração. A modelação por objectos e o UML (Unified Modeling Language) estão entre as melhores tecnologias para projectar e implementar aplicações complexas, para além do UML também ter sido proposto pelo OMG e com grande sucesso, por isso o CWM é baseado em UML. Com o CWM "todo este poder de modelação fica disponível para os administradores de Data Warehouses, para os 'arquitectos dos SI', e para os gestores das empresas" [CWMSpec].

Resumidamente, o CWM fornece um enquadramento para representar a metadata das fontes de dados, a metadata do destino dos dados (normalmente corresponde à base de dados que suporta a Data Warehouse), as regras de transformação e análise dos dados, e os processos e operações que permitem criar, gerir e fornecer informação sobre o uso dos dados.

O CWM, quando usado em conjunto com outras tecnologias, permite uniformizar a recolha da informação porque a metadata permite "explicar" a informação que está a ser recolhida. Esta simplificação reduz custos uma vez que não é mais necessário construir interfaces proprietárias entre fontes de dados. Por exemplo, as ferramentas de ETL vêem a sua tarefa facilitada devido à existência de uma norma para definir os mapeamentos de dados, e podem, por exemplo importar/exportar regras de transformação de outra ferramenta.

Para resumir, o CWM fornece a metadata comum a cada tipo de ferramenta que existe numa Data Warehouse, seja essa ferramenta de carregamento ETL, seja uma ferramenta de análise, seja uma ferramenta de monitorização. Um dos metamodelos em que focamos mais a nossa atenção é o *Transformation*. Este metamodelo serviu de base para as experiências que estão descritas na Secção 4.

Passamos agora à análise de outra norma de metadata existente.

Ao contrário do CWM, que se tenta concentrar no problema específico da metadata nas Data Warehouses, o OIM é muito mais abrangente, desde as tecnologias CASE até à modelação de sistemas distribuídos (como sejam os usados na Internet).

No entanto, tal como o CWM, o OIM é baseado no UML, propondo submodelos específicos para determinadas áreas; tais como "análise", "engenharia de negócio", "objectos e componentes", "DataWarehousing" e "gestão de conhecimento".

O modelo "Database and Warehousing Model" é aquele que mais nos interessa, pois providencia conceitos de metadata úteis para desenho da base de dados, reutilização do esquema da base de dados e DataWarehousing.

Neste modelo existem vários pacotes específicos, entre os quais:

- informação sobre o esquema da base de dados (tabelas, vistas, interrogações, etc);
- esquemas OLAP para descrever bases de dados multi-dimensionais (incluindo a descrição de cubos, dimensões, e agregações);
- transformações de dados (as regras de mapeamento entre os dados da DataWarehouse e os dados do sistema operacional).;
- elementos para descrever os dados guardados na forma de ficheiros, e em bases de dados não relacionais (não inclui, no entanto, informação sobre como os dados estão armazenados fisicamente);
- definições para relatórios, representando a informação necessária para essa classe de ferramentas.

Uma vez que o pacote das transformações, é que o que mais nos interessa para este artigo, destacaremos alguns detalhes.

No OIM, o pacote "*Data Transformation"* foi criado com o objectivo de disponibilizar às ferramentas ETL um modelo de dados para armazenarem a sua informação de mapeamentos, permitir troca de informações de mapeamento entre ferramentas de Data Warehouse, permitir

descobrir as dependências existentes entre os dados, e disponibilizar um mecanismo de armazenamento de metadata para aplicações.

Existem algumas semelhanças com o modelo das transformações do CWM. Uma transformação mapeia um conjunto de objectos fonte num conjunto de objectos destino. As transformações podem ser definidas em vários graus de granularidade, podendo se necessário armazenar o programa (código fonte) que executa a transformação. É também possível agrupar as transformações em *Transformation Tasks*, de modo que se definam sequências de passos de transformações a aplicar (*Transformation step)* para executar uma determinada tarefa. Não existe no entanto o conceito de transformações de caixa branca.

Enquanto por um lado o CWM concentra-se em DataWarehouses, o OIM procura tem uma abrangência muito maior tentando incluir todas as fases de desenvolvimento dos sistemas de informação. Ambas as normas têm por base o UML, uma linguagem de modelação extremamente popular hoje em dia. Outra semelhança entre as duas normas é a linguagem usada para definir a: XML. Em Setembro de 2000 os grupos de trabalho do CWM e do OIM juntaram-se à volta da norma CWM.

Passemos agora a uma análise breve de como os produtos comerciais suportam estas normas.

3.3 Produtos Comerciais

O Oracle Warehouse Builder (OWB [OWB00]) é o produto fornecido pela Oracle para construir Data Warehouses. No OWB, O CWM é utilizado como formato para troca de metadata entre as aplicações que compõem o pacote. É no entanto visível que nos ficheiros CWM foram usadas muitas extensões proprietárias, o que é aceitável apenas por este ficheiro ser para uso 'interno' do pacote Oracle.

Em versões futuras, o OWB poderá usar mais extensivamente o CWM para exportar informação relativa aos processos de ETL e respectivo escalonamento. Utilizando uma bridge CWM também é possível extrair metadata relativa às ferramentas de modelação Oracle.

Conforme podemos observar, a metadata armazenada pelo OWB é a relativa à definição das tabelas, aos mapeamentos da Data Warehouse e à sequência de carregamento de dados sendo todo este tipo de informação armazenada suportada pelo CWM.

Ao contrário da Oracle, a IBM optou por uma estratégia descentralizada em relação à metadata na qual cada ferramenta pode ser autónoma e ter a sua própria metadata, embora haja um conjunto de metadata que é comum a todas as ferramentas da IBM. Além disso a oferta da IBM na área de Business Intelligence é bastante mais completa que aquela fornecida pela Oracle

Infelizmente, a versão que a IBM disponibilizou para testes (7.1), não tinha ainda suporte para CWM embora o site da IBM indique que a versão 8.0 suporta CWM. A documentação IBM incluía, no entanto, excelente informação sobre a metadata exportada num formato ASCII proprietário.

Mesmo assim, é razoável pressupor, que a maior parte da informação sobre metadata presente no Information Catalog seja exportável em formato CWM. A informação relativa ao escalonamento no Process Model também se ajusta muito bem aos modelos *Warehouse Process* e *Warehouse Operation* por isso seria relativamente fácil exportar essa informação para CWM. Uma excelente utilização do CWM seria a importação do esquema das bases de dados dos sistemas operacionais.

4. SOLUÇÕES INOVADORAS PARA ARMAZENAMENTO DE METADATA

4.1 Introdução ao caso de estudo escolhido

Uma das mais importantes informações de metadata, para construir e manter uma Data Warehouse são aquelas sobre os mapeamentos dos sistemas fontes para as tabelas do sistema de apoio à decisão. Sem este conhecimento não é possível garantir qualidade na informação fornecida nem perceber os dados armazenados. Embora este não seja o tipo de metadata que mais interessa aos utilizadores finais na exploração da Data Warehouse, é essencial para aqueles que têm de manter a Data Warehouse a funcionar produzindo resultados correctos.

As transformações ETL estão entre os sistemas fonte e a Data Warehouse e podem ser especificadas basicamente de duas maneiras: a um nível de granularidade mais alto (designadas por caixa preta) ou mais detalhado (designadas por caixa branca). As transformações de caixa preta são algumas vezes o único tipo de especificação possível para um sistema fonte sobre o qual não se tem muito conhecimento. No entanto, se o objectivo é gerar automaticamente um programa que faça o carregamento da Data Warehouse, é indispensável o uso de expressões de caixa branca (excepto se tanto o gerador da metadata e como o (s) receptor (es) partilharem a mesma linguagem utilizada na expressão de caixa de preta).

Na Secção seguinte mostramos como armazenar este tipo de metadata em base de dados relacionais.

4.2 Armazenamento de metadata em base de dados relacionais

As expressões CWM são relativamente simples para as transformações do tipo caixa preta. Por isso, basta criar uma coluna por cada atributo utilizado nas expressões XML. Obviamente esta implementação apenas suporta alguns exemplos, na implementação genérica para qualquer tipo de representação de caixa preta seria mais complexo.

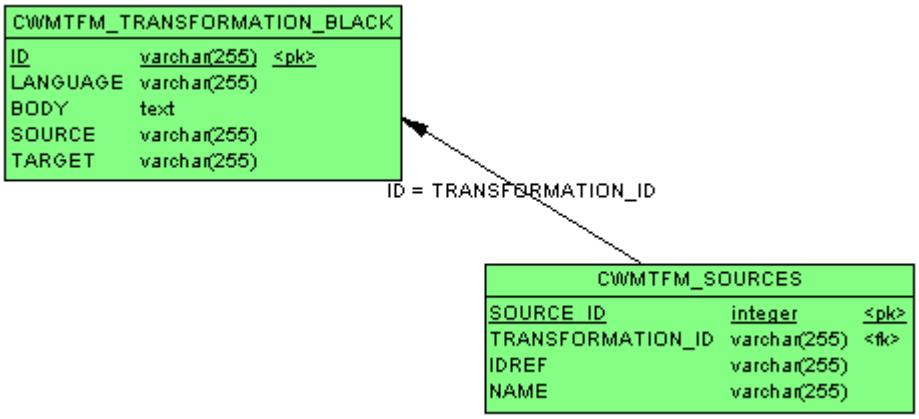


Figura 1 – Modelo relacional para transformações tipo caixa preta

Apresentamos em baixo, o conteúdo das tabelas para alguns exemplos:

ID	LANGUAGE	BODY	SOURCE	TARGET
1	SQL	select target=source	_2233	_3322
2	SQL	select target=a+b from tab_origem	_3322	null
3	SQL	select id from clientes where exists (select * from tab_pagamentos where cliente=:x and pago=false)	_3322	null
4	SQL	select convert(int, "2002")	_3322	null
5	SQL	if primeiro_nome(CLIE.NOME) in (select nome from nomes_pessoas where genero='masculino') then "M" else if primeiro_nome(CLIE.NOME) in (select nome from nomes_pessoas where genero='feminino') then "F" else if ultima_letra(primeiro_nome(CLIE.NOME)='a' then "F" /* em geral se a ultima letra do primeiro nome termina em a é de mulher */ else if ultima_letra(primeiro_nome(CLIE.NOME)='o' then "M" /* em geral se a última letra do nome termina em o é masculino" else NULL /* não é possível determinar com segurança */	_5566	_3322
6	SQL	CLIE.ID	_3432	_5566
7	SQL	SELECT NAME FROM COUNTRIES WHERE COUNTRIES.ID=:source	_5423	_5566
8	SQL	SELECT convert(inteiro, convert(text, clie.data_nascimento_seculo-1)+ convert(text, cliente.data_nascimento_ano))	null	null

Tabela 1 - conteúdo da Tabela CWTFM_TRANSFORMATION_BLACK

SOURCE_ID	TRANSFORMATION_ID	ID_REF	name
1	8	_5877	cliente.data_nascimento_século
2	8	_5878	cliente.data_nascimento_ano

Tabela 2 – conteúdo da tabela CWMTFM_SOURCES

Conforme se pode observar na tabela CWM_TRANSFORMATION_BLACK, os atributos das expressões CWM dão origem a colunas no modelo relacional. No campo *source* guardamos uma referência para o modelo relacional da fonte dos dados. Assim o valor 2233 corresponde ao identificador de um campo definido no *CWM Relational Model* (atributo `xmi.id` da tag `CWMRDB:Column`) para aquele SGBD. O campo *Target* corresponde à definição do campo destino. Para as transformações mais complexas nas quais estão envolvidas várias fontes de dados é necessário preencher a tabela CWMTFM_SOURCES.

As Transformações, propriamente ditas ficam guardadas na tabela `CWTFM_TRANSFORMATION_BLACK` no campo `BODY` na linguagem especificada no campo `LANGUAGE`. Para cada mapeamento que esteja definido teremos uma linha nesta tabela.

Em baixo apresentamos uma representação muito simplificada para um modelo relacional que suporta algumas expressões de caixa branca. No desenho deste modelo relacional mantiveram-se apenas os atributos essenciais, uma vez que mesmo assim já é possível estudar as limitações desta abordagem.

O esquema relacional da Figura 2 foi desenhado manualmente a partir do DTD do CWM por não se ter encontrado nenhuma ferramenta que o fizesse convenientemente de modo automático (a directiva ANY que é usada no DTD do CWM tornava impossível determinar algumas construções impossíveis na prática). Além disso, pretende-se trabalhar com um modelo simplificado porque bastava armazenar apenas os exemplos das Secções anteriores. Como existem algoritmos para gerar esquemas relacionais a partir de um DTD, utilizamos um retirado de [ATG00].

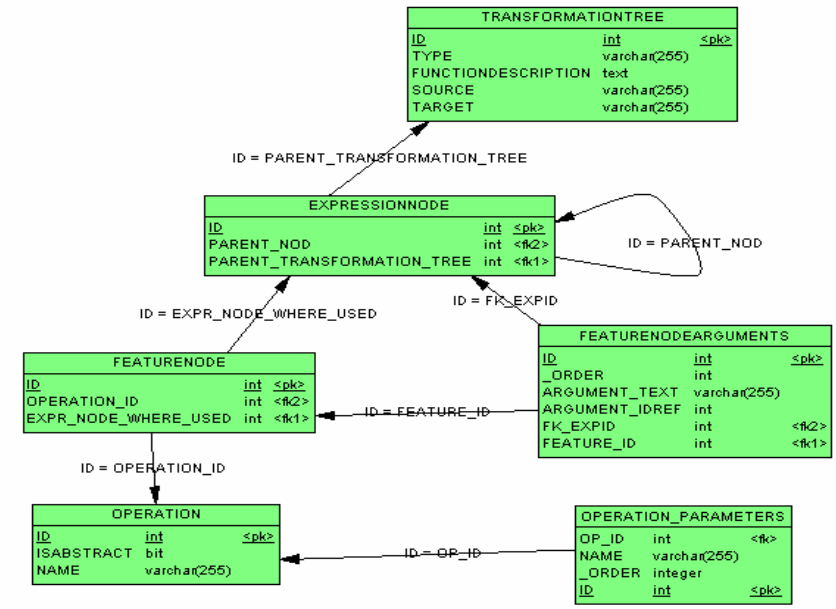


Figura 2 Modelo relacional simplificado para transformações de tipo caixa branca

4.3 Armazenamento de metadata em bases de dados nativas XML

O Armazenamento do CWM em bases de dados nativas XML é completamente transparente sendo suficiente enviar o documento a armazenar, não há portanto necessidade de uma camada de software intermédia como no caso do armazenamento no modelo relacional. Além disso as linguagens de interrogações XML permitem lidar facilmente com estruturas hierarquizadas.

No entanto, de acordo com [Bourret00] (e confirmado pela experiência prática que tivemos) a actualização de dados em base de dados XML ainda é uma área em desenvolvimento. Hoje em dia, ainda existem inúmeras estratégias para actualização e remoção de documentos, desde simplesmente substituir o documento DOM armazenado até bases de dados que suportam linguagens proprietárias para modificar fragmentos de um documento. Foi aceite no entanto, uma tentativa de normalizar essas linguagens de actualização de XML com a norma XUpdate [Xupdate].

4.4 CWM é muito mais do que apenas XML

Conforme vimos a norma CWM é baseada em XML. No entanto, as experiências que realizámos não são apenas um estudo acerca de armazenamento de XML em bases de dados. São muito mais do que isso porque a metadata para Data Warehouses apresenta características únicas: complexidade alta, estrutura bastante hierárquica e necessidade de ser facilmente trocada entre ferramentas. É por isso que o XML parece ser um formato interessante para armazenamento de metadata em Data Warehouse, pois lida bem com dados hierárquicos e nasceu para troca de informação entre aplicações.

Por exemplo, comparando o DTD de uma factura simples com a norma CWM [CwmSpec] a diferença é abismal em relação a um típico DTD. O CWM é um modelo de dados (ou melhor um metamodelo) criado para descrever a metadata em Data Warehouses. O CWM é integrado (por exemplo, na definição de um modelo de transformações podemos fazer referências a um modelo sobre a base de dados relacional na qual aplicamos as regras) mas, ao mesmo tempo, modular para facilitar a implementação de apenas partes do modelo.

Os exemplos que se seguem devem ser vistos como representações de um modelo de dados para metadata em Data Warehouses. O CWM não é apenas um exemplo de XML, o CWM permite definir a informação de uma Data Warehouse para ser um ambiente integrado e propicio para análise ad-hoc.

4.5 Comparação dos dois métodos

A utilização da base de dados XML permite simplificar muito a manipulação do CWM. A grande vantagem da utilização de uma base de dados XML para armazenamento do CWM é que nos permite lidar melhor com a complexidade. O modelo de base de dados XML permite armazenar todos os documentos CWM, no entanto, a complexidade do modelo de dados apenas vai ficando visível à medida que é necessária.

Quando armazenamos o XML, numa BD relacional, fazemos corresponder cada entidade a uma tabela diferente. Para fazer o relacionamento entre essas entidades é obrigatória a utilização de *ids* que irão permitir os *joins*. No modelo XML, o aninhamento dos vários elementos XML, reduz a necessidade de fazermos *joins*.

Normalmente é mais simples apresentar os dados se estes estiverem em XML do que armazenados em tabelas relacionais, uma vez que facilmente transformamos o XML em HTML. No entanto pode ser também uma desvantagem: suponhamos que apenas se pretende saber os nomes das tabelas existentes numa base de dados relacionais para os exibir numa 'listbox' em que o utilizador seleccionava uma das tabelas. Podemos usar um ficheiro CWM para trocar esta informação, mas teríamos o *'overhead*' suplementar de percorrer a árvore DOM para obter cada um dos valores, quando talvez fosse mais simples obter com uma interrogação SQL os nomes das tabelas de uma determinada base de dados. (Em Sybase poderia ser algo do género: *select name from sysobjects where type="U")*. Este problema acontece porque as bases de dados nativas XML retornam apenas XML, mesmo quando não precisamos de XML. Se a aplicação necessitar dos dados noutro formato, tem de fazer o "*parsing*" do XML antes de poder usar os dados. Embora uma interrogação XQL possa retornar uma "lista de valores" em rigor é sempre necessário fazer o *parsing* do documento XML que é a resposta da BD XML.

No modelo relacional a construção das tabelas para armazenar o CWM é uma tarefa tediosa e sujeita a erros. É preciso desenhar as tabelas para que possa guardar o documento CWM mais complexo que possa existir, quando na maioria dos casos apenas uma parte do esquema será usado. Por exemplo, uma ferramenta poderá estar interessada em exportar apenas o esquema de uma base de dados, outra poderá ter interesse em exportar apenas as regras de transformação para que se visione numa ferramenta de *reporting* a origem de cada um dos

dados. No entanto, ambas as aplicações têm de ser complicadas sem nenhum benefício aparente por eventualmente preencherem tabelas que não têm nada a ver com o problema em questão.

Além disso o CWM é composto por 204 classes (ver [Chang02] pág. 108) o que significa que sem contar com todas as relações possíveis entre as classes, teremos no mínimo 204 tabelas para suportar a metadata num modelo relacional. O que acontece é que a metadata é semi-estruturada e tem uma estrutura regular, mas os mapeamentos para uma base de dados relacional resultam num número grande de colunas com valores nulos e utiliza uma quantidade grande de tabelas, o que complica ainda mais a sua utilização.

As interrogações SQL para obter uma sub-expressão armazenada (em caixa branca) em tabelas relacionais são mais complexas do que as interrogações XQL para adquirir a mesma informação. Consequentemente, o tempo para reconstruir o documento XML é maior do que se fosse armazenado numa base de dados XML onde o documento inteiro ficasse junto fisicamente (ver [Bourret00], secção "Text-Based Native XML Databases"). Além disso as interrogações em XQL para obter determinadas partes do documento são mais simples de escrever. Para armazenamento de expressões em caixa preta é indiferente o tipo de armazenamento usado devido à sua grande simplicidade.

5. MODELO PROPOSTO E TRABALHOS FUTUROS

5.1 Explicação do modelo proposto

De acordo com as experiências realizadas na Secção 4, é possível concluir qual o tipo de armazenamento mais adequado para um repositório de metadata. As experiências centraram-se no problema de armazenamento dos mapeamentos dos sistemas fonte para os sistema destino numa DataWarehouse, mas é possível extrapolar os resultados para os restantes tipos de metadata.

1) No armazenamento de mapeamentos de caixa preta, tanto o armazenamento numa base de dados relacional como numa base de dados XML são adequados; os dados armazenados têm uma estrutura muito simples. Conforme vimos no modelo relacional bastou criar duas tabelas.

2) No armazenamento de expressões de caixa branca, o modelo relacional tem uma complexidade alta independentemente do tipo de mapeamento a armazenar. Além disso o modelo de base de dados relacional criado cobre apenas as funcionalidades básicas do modelo das transformações do CWM. 3)Se utilizarmos uma base de dados nativa XML não existe necessidade de realizar um mapeamento entre o formato de troca de dados (CWM/XML) e o formato de armazenamento.

4) Se utilizar uma base de dados relacional é necessário realizar uma série de *joins*, ou até escrever um pequeno programa para reconstruir o documento CWM trocado entre os vários componentes da arquitectura de uma DataWarehouse. A base de dados XML permite automaticamente reconstruir o documento além disso preservando a propriedade round-trip.

5) Ao se utilizar o CWM como formato de troca de metadata entre os vários componentes da DataWarehouse, obtemos um sistema mais aberto, mas ao mesmo tempo integrado.

A Figura 3, apresenta uma proposta para uma nova arquitectura para armazenamento de metadata que resultou das experiências que foram apresentadas na Secção anterior. A ideia base desta nova arquitectura é utilizar uma base de dados XML única para armazenar a metadata de uma Data Warehouse em formato CWM, que será utilizada pelas várias ferramentas que compõem ou utilizam a Data Warehouse. Esta decisão é baseada no facto da norma CWM ser baseada em XML, uma linguagem para a qual já existem diversas normas e produtos tanto para armazenamento (por exemplo, Tamino) como para troca de dados entre aplicações (por exemplo, SOAP).

O próprio CWM já prevê (através do XMI [OMG99]) a sincronização incremental entre várias fontes de metadata, pelo que provavelmente seria desejável ter algum processamento na base de dados central para aplicar essas alterações, simplificando dessa forma as interrogações ao repositório.

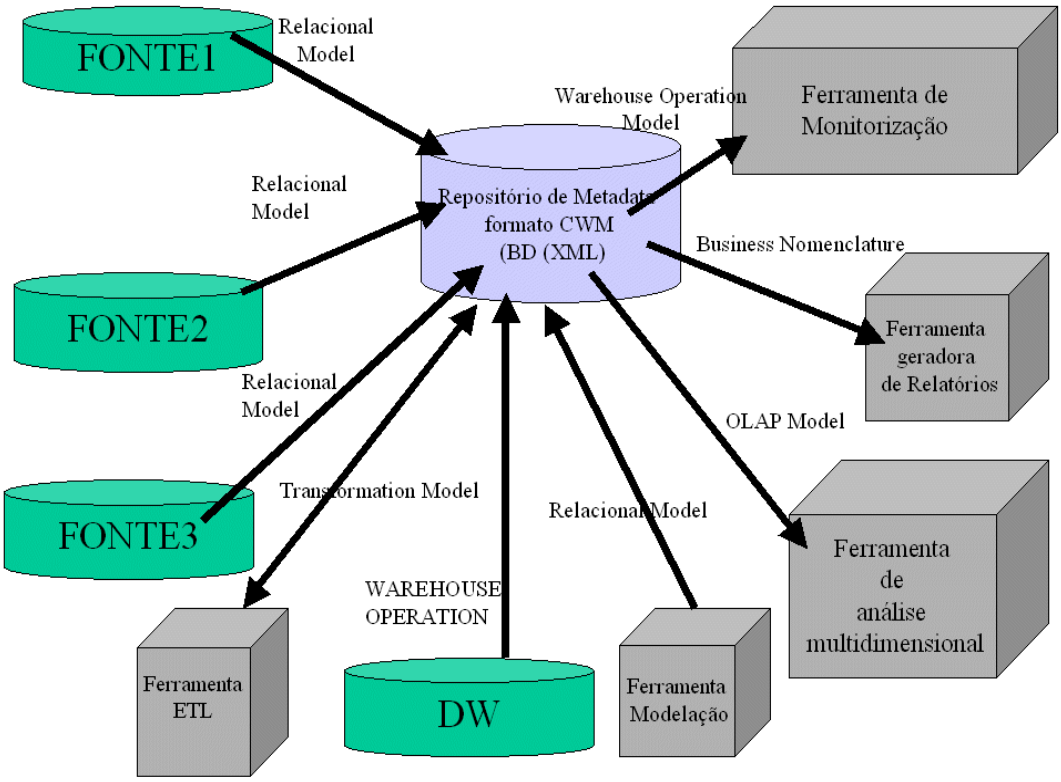


Figura 3 - Modelo Proposto

Cada tipo de ferramenta exporta para o repositório a sua metadata usando o submodelo CWM mais apropriado. No caso das ferramentas de ETL é usado extensivamente o modelo das transformações e no caso das ferramentas de monitorização o submodelo "Warehouse Operation". No entanto a integração é facilitada graças ao CWM.

Por outro lado a metadata sobre as estruturas das bases de dados operacionais pode ser importada pelas ferramentas de ETL para facilitar aos utilizadores a especificação dos mapeamentos.

A utilização de uma base de dados XML para armazenar a metadata, facilita o processamento da metadata exportada pelas ferramentas e permite controlar a complexidade, uma vez que as ferramentas não precisam usar todas as funcionalidades fornecidas pelo CWM, nem os utilizadores têm de saber todas as propriedades existentes de determinado submodelo.

5.2 Exemplos de utilização do modelo proposto

Para explicar melhor a ideia base da arquitectura proposta enumeramos em seguida alguns exemplos de utilização.

- Cada uma das fontes da Data Warehouse exporta a sua metadata utilizando um documento XML, onde deverá estar incluído (pelo menos) o modelo "CWM Relational" (Esta obrigatoriedade significa inserir no repositório toda a informação relativa a nomes de tabelas e colunas das bases de dados operacionais)
- A ferramenta ETL exporta a informação de mapeamentos utilizando o modelo das transformações.
- Um "*script*" interpreta o resultado dos procedimentos de carregamento da Data Warehouse e exporta os resultados usando o modelo das Operações Warehouse.
- Por sua vez, a ferramenta de monitorização acede ao repositório de metadata para obter o estado do carregamento da Data Warehouse.

- As ferramentas utilizadas para análise multidimensional acedem ao repositório de metadata para saberem como construir os cubos multidimensionais.
- A ferramenta ETL pode detectar erros que poderão acontecer devido a mudanças nos sistemas operacionais. Por exemplo, o modelo das transformações usa o modelo relacional, se uma das colunas da *fonte1* alterar o seu tipo é possível fazer uma análise de impacto.
- A ferramenta de modelação de Data Warehouse exporta informação sobre o esquema utilizado na Data Warehouse, e eventualmente efectua validações lendo informações da própria Data Warehouse.

5.3 Trabalhos Futuros

A utilização do CWM possibilita que se desenvolvam ferramentas de monitorização genéricas para Data Warehouses utilizando o modelo Warehouse Operation. Por isso, um dos trabalhos futuros é estender uma ferramenta de monitorização existente com essa funcionalidade ou mesmo construir uma de raiz. O objectivo seria explorar até que ponto é possível construir uma ferramenta genérica.

Outro tema que poderia ser estudado é a reengenharia inversa de metadata presente nas rotinas de carregamento da Data Warehouse. Ou seja, o objectivo seria desenvolver um programa que extraísse as regras de mapeamento dado o conjunto de "*stored procedures*" usados no carregamento de uma Data Warehouse para gerar uma representação usando o modelo de transformações. Esta ferramenta teria um campo de aplicação vastíssimo nas Data Warehouses existentes, que por vezes são difíceis de perceber e alterar.

Outro tema vastíssimo que pode ser explorar é estender esta proposta apresentada para um repositório que além de armazenamento oferece serviços. Poderiam ser utilizados *WebServices* implementando por exemplo serviços de sincronização, consolidação, e subscrição (para replicação) de metadata. Poderiam ser desenvolvidos protótipos bastante potentes e úteis

6. CONCLUSÕES

A análise das normas de metadata como o CWM, levou-nos à conclusão da importância central da metadata na integração das ferramentas numa Data Warehouse. Foi também entendida a importância dos modelos do CWM como seja o modelo "Warehouse Process" e o "Warehouse Operation". O Warehouse Process é muito útil para definir os processos de ETL de uma Data Warehouse, ao definir a ordem das tarefas de transformação e as respectivas dependências. O "Warehouse Operation Model" pode ser usado como auxílio a ferramentas de monitorização.

Tomando o modelo das transformações definido no CWM como base para as experiências, definiu-se um modelo relacional para suportar a metadata dessas representações. A conclusão que se tirou foi que, embora exista um algoritmo definido a reconstrução do documento XML original não é uma tarefa simples. Outra conclusão foi que normalmente o armazenamento de XML em bases de dados relacionais não goza da propriedade *round-trip* (i.e. o documento extraído não é exactamente igual ao documento armazenado).

Resumindo todo o artigo, podemos afirmar que as bases de dados nativas XML são uma boa solução para o armazenamento da metadata de uma Data Warehouse, e que o CWM fornece uma óptima estrutura para modelos de metadata. No entanto, uma arquitectura como aquela proposta na Secção 5 permite tirar ainda mais proveito da metadata ao mesmo tempo que facilita a sua utilização pelos vários tipos de ferramentas.

Referências

[Alh00] Hugo Alhandra, *Report of the Final Year Project*, Instituto Superior Técnico, 2000.

[Alh02] Hugo Alhandra, *MsC Thesis*, Submitted. Instituto Superior Técnico, 2002.

[App00] Applied Technologies Group, *Putting Metadata to Work in the Warehouse*, 2000.

[ATG00] Applied Technologies Group, 2000. "Putting Metadata to Work in the Warehouse" – A White Paper

[Ber97] Alex Berson, Stephen J. Smith, "DataWarehousing, Data Mining, and OLAP"

[Bourret00] Ronald Bourret, 2000. “XML and Databases”

[Chang02] Chang, Dan 2002. “Common Warehouse Metamodel“ – An introduction to the Standard for Data Warehouse integration

[CWM] CWM. http://www.omg.org/technology/cwm/

[CwmSpec] OMG, 2000. Common Warehouse Metamodel (CWM) Specification

[OMG] OMG. http://www.omg.org

[OMG99] OMG, 1999. Especificação XMI 1.1. http://www.omg.org/cgi-bin/doc?ad/99-10-02

[OWB00] Oracle Warehouse Builder: A Technical Overview - Fevereiro 2000

[Xupdate] XML:DB Initiative. XUpdate – XML Update Language http://www.XMLdb.org/xupdate/xupdate-wd.html -

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

13 Fev.	S3(14.30h) Publicação Electrónica de Conteúdos	[*Museu da Pessoa*] - **Alberto Simões**, DI-UM; [*O XML no software de gestão de bibliotecas: GiB*] - **Manuel Alves**, Biblioteca Pública de Braga; **Adalberto Ferreira**, LibWare; [*No passado será ... XML*] - **Luis Ferreira**, IPCA / Sidereus;

Histórias de Vida + Processamento Estrutural = Museu da Pessoa

Alberto Manuel Simões and José João Almeida

Museu da Pessoa
Departamento de Informática
Universidade do Minho
{albie@alfarrabio.|jj@}di.uminho.pt
http://natura.di.uminho.pt

Resumo Este artigo apresenta a arquitectura actual do Museu da Pessoa, contemplando a forma como os documentos estão a ser editador, catalogados, arquivados, e processados para a criação das estruturas necessárias ao Museu.

1 Introdução

O Museu da Pessoa nasceu no Brasil, S. Paulo, de um grupo de historiadores que resolveram compilar a história do país usando depoimentos do cidadão comum. Depois de iniciado o projecto começaram a surgir outras ideias, e outros projectos. Foram-se construindo acervos não só da história do país, mas também da história de determinadas profissões, festas populares, instituições ou empresas.

Entretanto um docente da Universidade do Minho, depois de ter assistido a uma apresentação do Museu da Pessoa - Brasil, resolveu criar um núcleo em Portugal: o Núcleo Português do Museu da Pessoa[1].

Em Portugal, o projecto está a ser mantido pelo Departamento de Informática que tenta manter o espírito do Museu da Pessoa original:

- é urgente recolher histórias da pessoa comum;
- sempre que possível, uma história deve ser contada na primeira pessoa, com toda a sua riqueza adicional;
- as histórias estar acessíveis em vários media (Internet, réplicas, livros, etc.), uma vez que se trata de um património comum;
- as regras de ética e direitos de autor têm de ser respeitadas;
- usar abordagens o mais metódicas possível;

No entanto, ao estar sediado num departamento de informática um dos principais objectivos é criar uma base informática para a gestão do acervo.

O projecto está a tentar evoluir para uma rede de Museus da Pessoa a nível mundial.

A secção 2 apresenta uma introdução ao ciclo de vida ou modelo de tratamento das histórias de vida recolhidas. De seguida, na secção 3, apresenta-se as tecnologias usadas para arquivar os documentos recolhidos enquanto que as secções seguintes apresentam as várias tecnologias usadas para a geração dos vários media.

2 Tratamento de Histórias de Vida

As histórias de vida são recolhidas e tratadas segundo o modelo apresentado na figura 1.

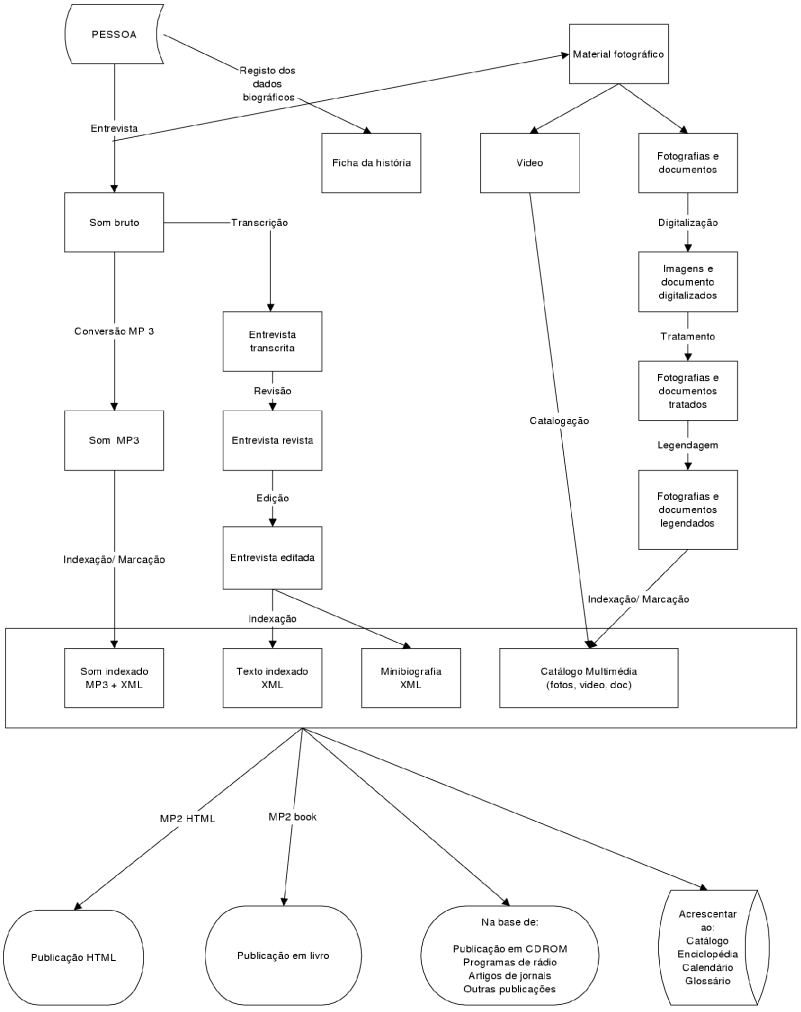


Figura 1. Modelo de tratamento da História de Vida

Resumidamente:

– entrevista e recolha de documentos junto do público — inclui a entrevista e gravação vídeo (ou apenas sonora) da entrevista, registo dos dados bi-

ográficos (e questões jurídicas, como direitos de autor) e recolha de material vídeo ou fotográfico como sejam cartões, fotografias e outros;
- digitalização dos documentos encontrados — digitalização de vídeos e dos documentos, tratamento gráfico, legendagem em XML e inclusão no catálogo multimédia;
- transcrição do documento para XML — conversão do som da entrevista para formato electrónico de forma a facilitar a transcrição em computador utilizando *software* desenvolvido no Museu;
- revisão da entrevista — correcção de erros ortográficos, edição de histórias, marcação XML de zonas interessantes. Criação de uma mini-biografia do depoente. Indexação da entrevista no catálogo geral; Este conceito de revisão leva à existência de mais do que um documento por entrevista, como seja um em formato de entrevista e um outro na primeira pessoa, como se o próprio depoente o escrevesse.
- publicação web do material recolhido — transformação de XML em HTML, divisão das histórias em pequenas porções, criação de álbum de fotografias, extracção de pequenos vídeos e várias outras operações.
- publicação em papel — criação de colectâneas de histórias de vida, transformação do XML para LaTeX, criação de livros em PostScript para *download* ou impressão.

3 Suporte Informático

Embora o projecto conte actualmente com mais de dois anos de existência, o seu suporte informático ainda não foi totalmente decidido. De facto, ao longo do seu desenvolvimento foram tidos em conta os seguintes objectivos:

- utilizar formatos abertos, de modo a facilitar o intercâmbio de histórias entre ferramentas, permitir independência de plataformas e de aplicações;
- disponibilização dos documentos em formatos ricos, capazes de estabelecer relações entre os mesmos;
- utilizar estruturas classificativas comuns entre todos os documentos;
- disponibilização de ferramentas construídas e de formatos propostos;
- privilegiar a automatização;

No entanto, só alguns dos pontos relativos à escolha do suporte informático foram completamente definidos, enquanto que outros vão evoluindo à medida que a necessidade surge.

3.1 Armazenagem

Em relação ao formato em que se deveria armazenar o acervo, a sua escolha não foi complicada. A possibilidade de se definir estruturas complexas em XML e de ser um formato aberto e facilmente processável fez-lo a escolha do Museu.

O XML permite que vários dos objectivos de desenvolvimento de Museu da Pessoa se concretizem mas, por si só, não resolve todos os problemas. Em particular, foi necessário definir uma forma de arquivar os vários documentos em algum sítio. Com este fim surgiram duas hipóteses:

– utilizar uma base de dados, estilo Oracle que, nas suas versões mais recentes, inclui suporte para campos de tipo XML, permitindo a realização de *queries* XQL sobre estes;
– criar um documento XML por entrevista e, de alguma forma, utilizar o sistema de ficheiros do sistema operativo para guardar estes documentos;

A primeira hipótese obriga à utilização de ferramentas mais pesadas e, na sua maioria, comerciais. Sem dúvida que a utilização de uma base de dados permite ao museu crescer facilmente, mas a falta de fundos obrigou ao uso temporário de uma estrutura mais ou menos rígida sobre o sistema de ficheiros.

Um outro problema surgiu ao usar esta abordagem: várias pessoas podem vir a editar o mesmo documento ao mesmo tempo ou, determinada edição pode a não fazer sentido, sendo necessário quer gerir conflitos entre as várias edições, quer ter capacidade de recuar edições.

Para resolver este tipo de situações optou-se por incluir todos os documentos XML num repositório CVS (Concurrent Version System) que permite de uma forma bastante eficiente permitir que vários utilizadores possam editar documentos mantendo sempre o histórico das alterações. Inclui também, a capacidade de juntar versões paralelas e só em casos extremos requerer a intervenção humana.

A organização do acervo no sistema de ficheiros obrigou à definição de determinadas regras para manter limpeza e coerência na sua arrumação:

– cada directoria na raíz do repositório corresponde a um projecto;
– directorias a níveis superiores podem corresponder a sub-projectos ou a depoimentos dentro desse projecto;
– cada directoria de projecto (ou sub-projecto) pode conter um documento XML com uma pequena sinopse com uma introdução e objectivos do projecto, várias directorias de depoentes ou sub-projectos, e uma directoria com fotografias;
– cada directoria de depoente pode ter várias versões da entrevista em XML, um documento com a identificação do depoente e uma directoria com fotografias;
– Cada directoria de fotografias contém imagens e um documento XML com a legendagem das mesmas;

A figura 2 mostra esta estrutura de uma forma gramatical.

$$Projecto \leftarrow Projecto^{\star} \times Entrevista^{\star} \times Album \times SinopseXML$$
$$Entrevista \leftarrow DocXML^{\star} \times BiXML \times Album$$
$$Album \leftarrow LegendaXML \times Foto^{\star}$$

Figura 2. Estrutura de directorias do repositório

3.2 Etiquetação

A informação relativa ao depoente está toda a ser guardada na sua directoria dentro do projecto em que se insere. Esta informação está dividida em três partes:

- mini-biografia e dados pessoais, como sejam o nome, data e local de nascimento, e profissão. Esta informação é colocada num documento próprio, denominado de BI (Bilhete de Identidade — na figura 2, denominado por BiXML);
- várias versões da entrevista, em documentos XML. Cada nome do documento deve reflectir o seu tipo (entrevista, editado);
- informação relativa a documentos digitalizados encontra-se no documento de legenda dentro da directoria de fotografias;

Estes documentos são processados automaticamente sempre que é necessário voltar a publicar determinado projecto. Este processamento constrói o catálogo (índice) de documentos do projecto e extrai um conjunto extra de informação.

Bilhete de Identidade

Como foi referido, o bilhete de identidade do depoente está a ser armazenado em XML. No entanto contém informação de tal forma rígida que não é mais do que uma tabela de base de dados armazenada em formato XML.

O seu conteúdo é:

- nome, profissão, data e local de nascimento;
- mini-biografia para ser apresentada como resumo da história de vida;
- informação sobre uma fotografia *escolhida* para ser mostrada juntamente com a mini-biografia;

A figura 3 mostra um bilhete de identidade a ser consultado na Internet.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE bi SYSTEM "http://alfarrabio.di.uminho.pt/mp/dtd/bi.dtd">
3  <bi>
4   <projecto>Memórias do Trabalho</projecto>
5   <depoente>José Vieira</depoente>
6   <biografia>
7     José Vieira nasceu a 16 de Outubro de 1920 em Resende. Filho de
8     lavradores passou uma infância dificil. Tinha 7 anos quando o pai
9     ficou paralisado. Dois anos depois foi morar com os tios que eram
10    lavradores.  Mais tarde, foi com a mãe e os irmãos para o Porto
11    trabalhar na lavoura. Foi feitor na Quinta do Ramalho e trabalhou
12    ainda como serralheiro na F. Brindley. Esteve sempre envolvido nas
13    reivindicações dos trabalhadores.
14  </biografia>
15  <profissao>lavrador; feitor; serralheiro; jardineiro</profissao>
16  <nascimento onde="Resende" mes="10" dia="16" ano="1920"/>
17 </bi>
```

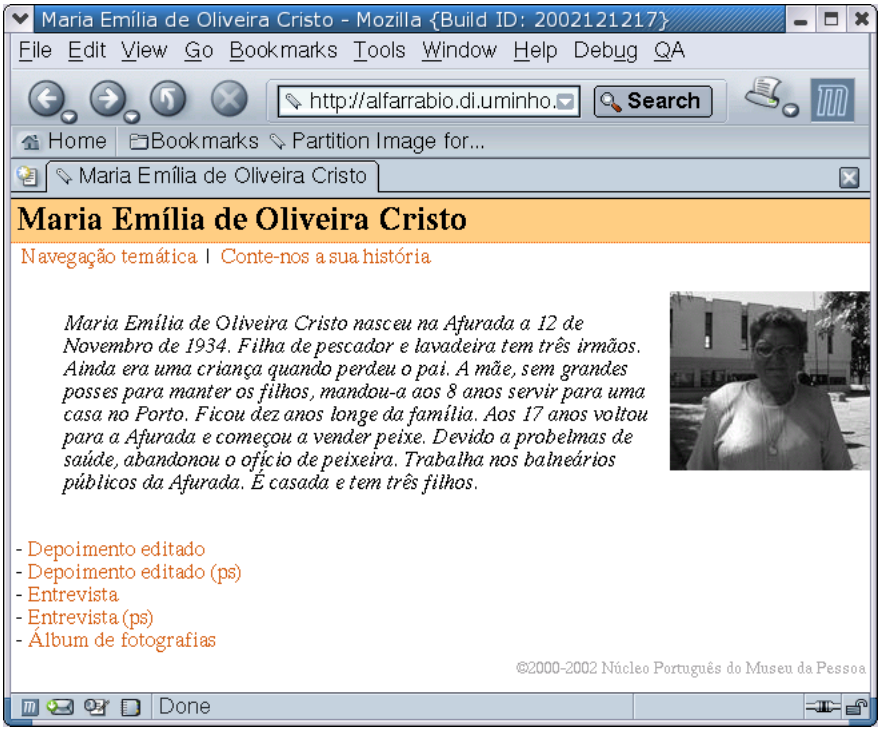


Figura 3. Consulta de um Bilhete de Identidade

História de Vida

As etiquetas usadas nos documentos de histórias de vida dividem-se em três grupos fundamentais:

marcação de estrutura do mesmo género das etiquetas estruturais do HTML, permitem que se definam parágrafos bem como estrutura em poemas (poema, estrofe e verso);

```
<poema>
  <estrofe>
    <verso>Hoje que Deus me levou</verso>
    <verso>Não chorem que é o Destino. </verso>
    <verso>Ele já ficou traçado </verso>
    <verso>Dos meus tempos de menino. </verso>
  </estrofe>
  <estrofe>
    <verso>Fui um poeta romântico </verso>
    <verso>E cumpri a minha sina. </verso>
    <verso>Fui um jovem tão feliz </verso>
    <verso>E formei uma família. </verso>
  </estrofe>
</poema>
```

marcação de meta-informação delimitam zonas do depoimento que devem de algum mode ser processadas de forma especial. Exemplo desta etiquetação são as datas do documento, bem como nomes de instituições ou expressões regionais (e respectiva explicação).

```
(...)
Não quer ir para a pesca do bacalhau, não trabalha mais, vá trabalhar
para onde quiser, mas aqui não trabalha mais" - e ele passou muita
fome até resolver esse problema. Isto era no tempo da
<ref tipo="Instituição">PIDE</ref>,
no tempo da
<expressao
  significado='Quando as pessoas se referiam à PIDE, chamavam sempre
     "outra senhora" e não PIDE, com medo de represálias.'>
  outra senhora</expressao>,
no tempo do
<ref tipo="personalidade">Cardeal Cerejeira</ref>
que dizia que - "Para o povo ser humilde tem que
passar fome" - é verdade, é verdade.
(...)
```

marcação de histórias embora também possam ser consideradas etiquetas de meta-informação, estas delimitam pequenas porções da história de vida que podem ser lidas independentemente do depoimento completo. Desta forma, permitimos que uma história de vida possa dar origem a histórias temáticas. Como exemplo temos mais de uma dúzia de etiquetas como: "*namoro*", "*ofício*", "*casamento*", "*infância*". Além destas, existe uma genérica denominada "*episódio*" para ser usada caso não exista uma etiqueta para o tipo de história a anotar. Poderiamos ter usado sempre esta, para todas as histórias a anotar. No entanto, ao definir um conjunto de etiquetas específicas para este propósito, lembramos os transcritores da sua existência e marcamos *slots* a preencher.

```
<ascendencia>
 <p>
 O meu pai era José Teixeira Bonifácio, era tintureiro na <ref
 tipo="empresa">Fábrica dos Carrinhos</ref>, onde ganhava 12$50 por
 dia. O meu pai não sabia ler nem escrever, fui eu que lhe ensinei a
 escrever o nome. A minha mãe, Emília Tomásia Pereira da Silva, era
 doméstica mas tinha ocasiões em que vendia peixe. No Inverno havia o
 período das traineiras encostarem, durante 3 meses. Nessa altura não
 havia peixe. Os meus pais parece que tiveram 12 filhos, mas alguns
 morreram com a meningite. Eu era o mais velho.
 </p>
</ascendencia>
```

A imagem 4 apresenta o aspecto de uma história de vida a ser consultada na Internet.

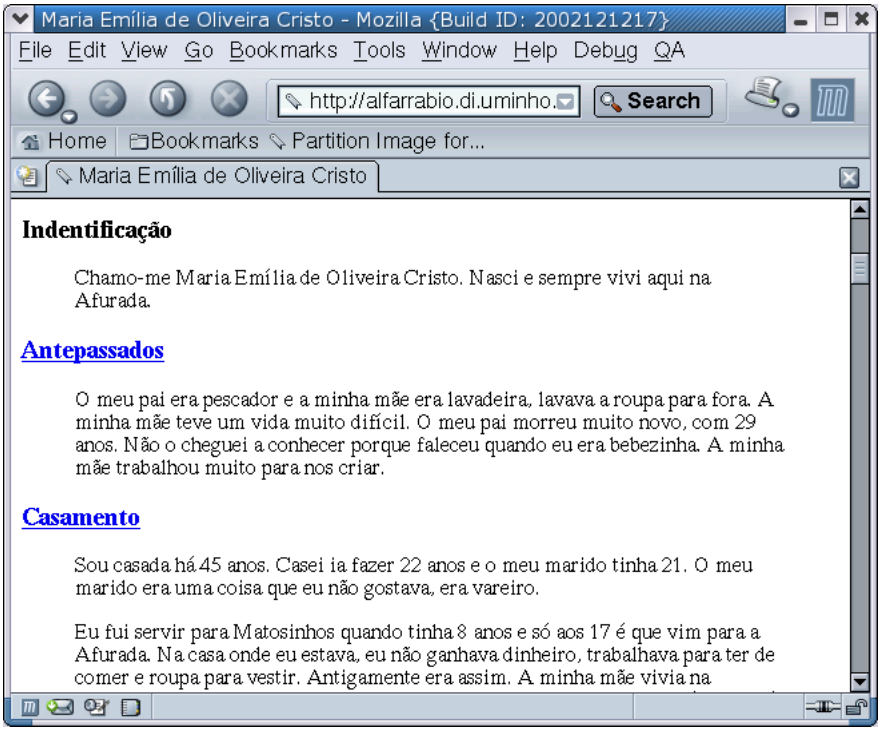


Figura 4. Consulta de História de vida

Legendas

A legenda inclui uma secção para cada fotografia ou documento digitalizado com o nome do ficheiro e uma legenda composta não só pela descrição da imagem mas também uma data e, sempre que possivel, nomes dos intervenientes. A figura 5 mostra um álbum fotográfico.

```
<fotos>
 <foto ficheiro="004-F-07.jpg">
  <onde>Ribeira, Porto</onde>
  <quando>1950</quando>
  <quem>
   Do lado esquerdo, Francisco da Cruz Lopes, o marido de Maria de
   Lurdes Pereira Vásquez, e do lado direito, Francisco Moreira dos
   Santos, o sogro.</quem>
  <facto>
   As cheias de 1950 que atingiram a Ribeira. O rio Douro invadiu o
   mercado.</facto>
 </foto>
 <foto ficheiro="004-F-06.jpg">
  <quando>1925</quando>
```

```
15      <quem>Rosalina Pereira, sentada com a filha Maria de Lurdes Pereira
16       Vásquez ao colo, acompanhada pelo marido Isidro Jorge Vásquez.</quem>
17      <facto>Baptizado de Maria de Lurdes Pereira Vásquez. </facto>
18     </foto>
19    </fotos>
```

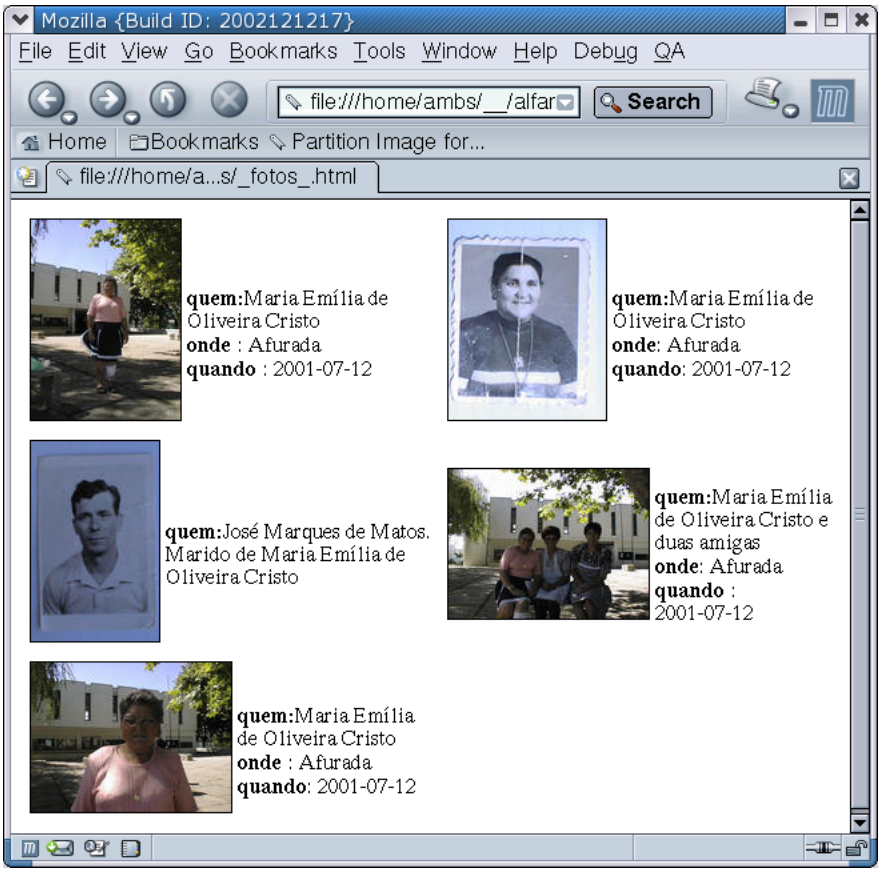


Figura 5. Álbum fotográfico

4 Geração Web

Como o museu da pessoa é virtual, a sua presença é feita especialmente na Internet. Para isso é necessário criar algum tipo de navegação sobre as entrevistas dando ao utilizador vários métodos de encontrar a informação que lhe interessa.

4.1 Navegação por projecto

Dado que os projectos estão organizados hierarquicamente (como descrito na secção 3.1) no sistema de ficheiros torna-se simples a construção de uma árvore e a sua navegação.

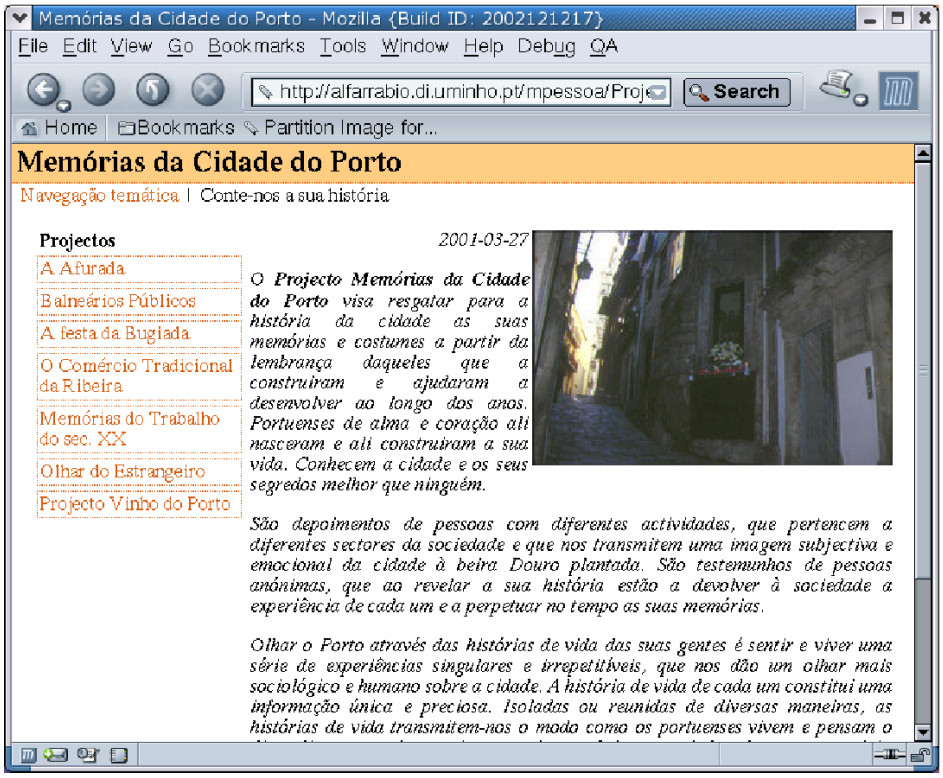


Figura 6. Navegação por projecto

A figura 6 mostra uma página típica de um projecto onde, do lado esquerdo, se pode ver um conjunto de ligações para sub-projectos.

4.2 Navegação temática

Para navegação temática foi utilizado um módulo Perl que utiliza um formato semelhante ao ISO para Thesaurus Multilingue. Este módulo Perl denominado `Biblio::Thesaurus` e alguns outros de suporte a bibliotecas digitais[4] estão a suportar a ontologia de navegação sobre o Museu.

No entanto, a definição de uma ontologia nem sempre é simples. Ou se tem uma equipa a trabalhar na construção de uma que contemple toda a realidade, ou torna-se difícil a sua manutenção.

Uma das vantagens de se ter utilizado XML e portanto, uma etiquetação definida de acordo com as necessidades do projecto, é o de se ter convencionado um conjunto de etiquetas para indicar termos de catalogação (ver a secção 3.2). Deste modo, pode ser feita a extracção automática destas entidades. No caso da navegação conceptual, são extraídos regularmente os termos usados nas histórias para ser criada uma lista de termos não contemplados na ontologia do projecto.

Além da ontologia é usado um catálogo de histórias (construído a partir da etiquetação das histórias) e que faz a ponte entre os termos apresentados na ontologia e os respectivos documentos. Podemos ver o par ontologia/catálogo como uma forma mais flexível para o que poderia ser implementado com Topic Maps.

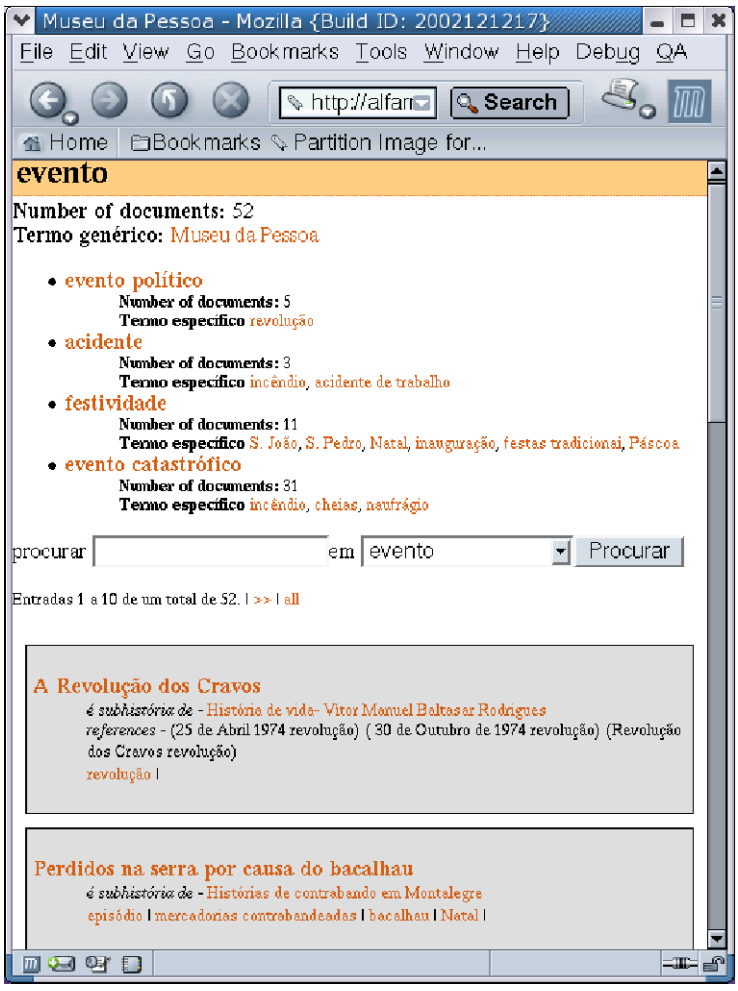


Figura 7. Navegação temática

A figura 7 mostra um resultado da navegação temática do Museu que incluí uma vista da ontologia mas também a possibilidade de uso directo de termos de pesquisa sobre o catálogo.

4.3 Outros recursos on-line

Além da navegação sobre as histórias e a sua consulta, o Museu pretende disponibilizar um conjunto de recursos relacionados com os depoimentos. Para seguir as directivas do Museu é crucial que estes recursos existam mas que sejam automatizados seus processos de construção.

Calendário

Qualquer história que seja recolhida pelo Museu acaba por conter um conjunto razoável de datas: nascimento, baptizado, casamento, volta da guerra, catástrofe e outras. Dados que todas estas datas devem ser correctamente etiquetadas na história, é possível construir ferramentas de extracção automática das datas e construção de um calendário mensal com as efemérides registadas no Museu. A figura 8 é o calendário deste mês, no dia 6 de Fevereiro de 2003.

Figura 8. Calendário a 6 de Fevereiro de 2003

Eixos Cronográficos

Utilizando as datas extraídas para o calendário e de eventos sociais, políticos e religiosos torna-se possível construir um eixo ou friso cronográfico. Neste eixo

marcam-se as datas relacionadas com as histórias para as relacionar com outros eventos.

Enciclopédia

Determinadas figuras ou personalidades, eventos sociais, políticos e religiosos, ferramentas ou instrumentos típicos, localidades, instituições e outros objectos são anotados para que se possam extrair e, desta forma, construir uma enciclopédia que os explique.

Glossário

Dada a diversidade de expressões usadas nas várias regiões do país é provável que nem todas as pessoas as conheçam. Com a marcação destas expressões e seu significado nas várias histórias recolhidas, estas são extraídas automaticamente para a construção de glossários.

5 Publicação em papel

Embora o Museu da Pessoa seja virtual um dos objectivos é poder publicar histórias noutros suportes como sejam CD-ROMs ou livros. Estes livros são especialmente de dois tipos: livros de depoimentos e colectâneas.

Para a publicação em papel optou-se por gerar LaTeX a partir dos documentos XML. Esta escolha permite que, por um lado, a geração dos livros possa ser automática (sem necessidade de utilizar ferramentas interactivas), e por outro, que se beneficie de toda a panóplia de módulos e ferramentas que o LaTeX disponibiliza.

Para as colectâneas tornou-se necessário definir que depoimentos devem ser introduzidos, e por que ordem, no livro resultante. Para isso, definiu-se um novo formato XML que especifica não só que história deve ser incluída e em que sítio, mas também qual o título, autores, e outras partes fixas como introdução e comentários.

6 Processamento estrutural

Para o processamento de toda estes documentos armazenados no sistema de ficheiros foi desenvolvido um sistema denominado DAG[5] (Directory Attribute Grammars).

Esta ferramenta utiliza um conjunto de regras semelhantes às usadas na escrita de gramáticas de atributos, onde é especificada a estrutura de todo o acervo e, sobre ela, um conjunto de funções de processamento. Na figura 2, apresentada na secção 3.1, podemos ver a semelhança existente entre uma especificação da estrutura arbórea e uma gramática tradicional.

Todos os ficheiros das páginas de Internet, e publicações em papel (documentos PostScript) são construídos por este sistema. Executando a ferramenta, esta irá detectar quais os ficheiros desactualizados e recalculá-los.

A conversão de XML para HTML é feita utilizando um módulo escrito em Perl[6] denominado XML::DT[2].

Uma das principais razões da utilização de métodos de programação convencional para o processamento dos documentos XML deve-se à flexibilidade demonstrada em comparação com ferramentas específicas para o manuseio destes documentos como seja o XSL. Em particular, algumas das funções definidas utilizam ferramentas existentes no sistema operativo, o que nos permite não reinventar a roda.

Por exemplo, para que as imagens possam ser mostradas num álbum fotográfico, as *scripts* de processamento das legendas vai, para cada ficheiro, criar um *thumbnail*. Da mesma forma, está a ser desenvolvido um método que permita aos membros do Museu da Pessoa verificar a ortografia dos documentos, utilizando para esse efeito o corrector ortográfico e analisador morfo-sintáctico Jspell[3].

Tamanho da árvore	283 Mbytes
Tamanho da árvore decorada	534 Mbytes
ficheiros XML	267 ficheiros
ficheiros JPG	524 ficheiros
ficheiros HTML	14 ficheiros
atributos HTML	1 745 ficheiros
atributos JPG	1 056 ficheiros
atributos EPS	533 ficheiros
atributos PS	223 ficheiros
Total de atributos	4 926 ficheiros
Atravessar a árvore	2 minutos
Recalcular a árvore	2:30 horas

A tabela mostra o número de ficheiros, atributos e tempos de geração do *site*. No entanto, é de frisar que estes valores não são da actual versão do museu. Além de outras razões, o tempo que demoraria a recalcular todo o *site* desde o início seria bastante elevado. No entanto, em relação aos tamanhos e números de ficheiros, estes duplicaram estando a árvore decorada a ocupar, actualmente, cerca de 1 GByte.

7 Conclusões e trabalho futuro

Sem dúvida que o projecto Museu da Pessoa é importante para o conhecimento social da população e do país. No entanto, do lado técnico podemos concluir:

– é possível que pessoas de áreas menos ligadas à informática tenham a capacidade de utilizar editores estruturados de forma eficiente;

– embora uma solução temporária, a utilização do sistema de ficheiros para armazenagem dos documentos tem vindo a mostrar-se mais fiável e manuseável do que o esperado;
– o método de cálculo de todo o *site* é bastante eficiente (visto não recalcular ficheiros desnecessariamente) e, ao gerar páginas estáticas, permite que a navegação sobre o Museu não obrigue à geração dinâmica de páginas;
– a publicação de pequenos livros por depoente ajuda não só aos editores de histórias para a sua correcção em qualquer sítio, como veio causar grande entusiasmo aos entrevistados;
– os métodos de navegação e pesquisa, embora em fase de desenvolvimento, permitem a consulta temática do acervo, muito apreciado pelos utilizadores;

Actualmente, centra-se o desenvolvimento em:

– recolha de histórias;
– preparar o acervo audio-visual do Museu para publicação, com a sua digitalização (de filmes e de som) e respectivo tratamento;
– criação de um suporte de gestão integrada do Museu utilizando a Web como Interface;
– análise e desenho de novas estruturas de armazenagem que venham a possibilitar a escalabilidade de todo o *site*.

Referências

1. J. João Almeida, J. Gustavo Rocha, P. Rangel Henriques, Sónia Moreira, and Alberto Simóes. Museu da pessoa — arquitectura. In *ABAD*, 2000.
2. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, May 1999.
3. Alberto Manuel Simões and José João Almeida. `jspell.pm` — um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas da Associaçáo Portuguêsa de Linguística*, 2001.
4. Alberto Manuel Simões and José João Almeida. Library::⋆ — a toolkit for digital libraries. In *ElPub 2002 - Technology Interactions*, 2002.
5. Alberto Manuel Simões, José João Almeida, and Pedro Rangel Henriques. Directory attribute grammars. In *VI Simpósio Brasileiro de Linguagens de Programação*, pages 297–308, 2002.
6. Wall, Larry & Christiansen, Tom & Schuartz, Randal. *Programming Perl*. O'Reilly & Associates, Inc.

O XML no software de gestão de bibliotecas: GiB

Manuel Alves, Adalberto Ferreira
SDUM,
Universidade do Minho
Braga, Portugal, 4710-057
{Manuel.Alves@bpb.uminho.pt, adaf@libware.pt}

Resumo

Este documento pretende ilustrar a mais valia que a adopção do XML trouxe à informatizaçãos das bibliotecas. Como suporte desta afirmação apresentaremos o software GiB e o uso que faz do XML. Porquê o XML? Por ser moda, uma "keyword" de marketing, ou porque realmente se revela uma ferramenta útil na execução da tarefa em questão? Julgamos verdadeira a última hipótese e é para ilustrar essa convicção que iremos apresentar o caso GiB.

- O problema -

O GiB nasceu de um protótico criado durante o projecto Geira, vertente bibliotecas. A informatização dos processos biblioteconómicos revelou-se uma tarefa extremamente "interessante" isto é, aquilo que a um informático poderia parecer inicialmente mais um caso de bases de dados e fluxos de informação revelou que às vezes o todo é mais do que a soma das partes. Ora então vejamos um exemplo simples: o caso do registo do ou dos autores de uma obra: pode ter apenas um, pode ter muitos, pode não ter nenhum, pode ter vários autores mas com papéis diferentes (história, ilustrações,...) podem ser pessoas físicas ou instituições, pode ser uma mistura de todos. Em resumo, para além da questão da quantidade existe também uma questão de qualidade. Existe um contexto. A coisa não fica bem resolvida com uma tabela de autores... Nota: o caso do autor não é único, antes pelo contrário.

As bibliotecas usam há muitos anos formas de descrever essa e outra informação. Usam standards denominados comunemente por normas MARC, que depois tem variantes regionais: USMARC que evoluiu para MARC21; UKMARC; etc ; e UNIMARC que foi uma tentativa de alguns países pararem com as mutações. Em Portugal o MARC usado segue a norma UNIMARC.

O que o UNIMARC faz é arranjar caixas "tags" onde se podem arrumar cada um dos elementos de catalogação. Como se pode adivinhar pelo caso do "autor" existem muitas "tags", cerca de 966 no caso do UNIMARC. Dentro do "bloco" de autoridade existem vários quarteirões para acomodar todas as variantes possíveis e imaginárias de autores.

Bem lá no fundo ainda existem subdivisões, por exemplo o apelido e o nome são guardados em tags distintas... Além do ano de nascimento do autor... bem como o ano da morte se fôr caso disso... ou ainda de uma codificação desses periodos se as datas forem incertas.

Quando juntamos mais de um milhão destes átomos - monografias, periódicos, panfletos publicitários, mapas, partituras musicais, CDs, DVDs e incunábulos temos algo como a Biblioteca Pública de Braga. Apenas a flexibilidade do UNIMARC permite meter todo este zoo na mesma base de dados e conseguir um sistema coerente e altamente eficaz de gestão da biblioteca.

No caso do GiB o XML é usado para resolver várias questões:

a) Transferir um registo de um módulo da aplicação a outro - Toda a informação necessária para perceber o que são os dados bem como o contexto dos mesmos vão numa cadeia de caracteres. Não existe o perigo de uma nova versão do software ser incompatível com a "API" do módulo da versão anterior

b) Extrair de um registo a informação que nos interessa, por exemplo a lista de autores seja o registo de um artigo, de um livro ou de um CD. Nas primeiras iterações do GiB foi construído um parser e uma linguagem própria que consumiam mais recursos com questões de processamento de linguagens do que com os problemas das bibliotecas.

c) Trocar registos com outros sistemas. Actualmente existe um protocolo, o Z39.50, que almejava esse objectivo. Quem já trabalhou com alguma implementação do mesmo sabe que embora seja um passo na direção certa não chega lá.

d) Aumentar a eficiência real da biblioteca, eliminando a catalogação. A catalogação é um processo moroso e muito subjectivo. O GiB no módulo de Aquisições e conhecendo alguma chave primária da obra, lança um pedido usando SOAP a um serviço web que permite que quando o livro chega à mão do catalogador o sistema já lhe apresente várias sugestões de catalogação feitas noutras bibliotecas. Por exemplo, num mundo perfeito, a Biblioteca Nacional deverá ter a catalogação bem feita de todos os livros publicados em Portugal...

e) O uso do XML permite estender o exposto na alínea anterior a livros publicados noutros países. Por exemplo podemos "capturar" um registo de um manual americano na Biblioteca do Congresso (em MARC 21) e usando uma simples transformação convertê-lo para UNIMARC sem qualquer intervenção do operador, já que para além dos dados, cada registo carrega consigo o seu bilhete de identidade e manual de instruções.

f) Aumentar a qualidade da base de dados. O GiB utiliza diversos tesuarus, em particular o Eurovoc, para permitir (finalmente) que a pesquisa por assunto

obedeça à lógica e não dependa da subjectividade do bibliotecário... Usando esta ferramenta, cada registo é indexado por uma estrutura lógica. Por exemplo, se consultarmos o tesaurus Eurovoc do GiB em http://brg.libware.net/eurovoc/ e digitarmos como ponto de partida a palavra GUERRA iremos obter uma extensa lista de termos e relações que permitirão de uma forma efectiva aumentar a incisividade da nossa pesquisa. Depois de navegarmos no tesauros de assuntos até ao ponto de acesso que realmente descreve o que procuramos é que saltamos para o catálogo da biblioteca. Estas ligações são todas feitas incluindo tags XML no registo de catalogação. Dado que o tesaurus cobre de momento nove línguas, poderíamos ter iniciado a pesquisa com a palavra WAR (mudando o índice para termos-geral) que obteríamos o mesmo resultado.

g) A descrição da própria base de dados é feita usando uma estrutura XML: a definição dos campos e subcampos MARC; o MARC utilizado; os índices disponíveis; as scripts que geram esses índices; os formatos de visualização de registos; as estruturas de validação e controle de qualidade; tesaurus disponíveis, etc.

Paramos o resumo neste ponto levantando no entanto o véu para os desafios que implicam a gestão de um catálogo colectivo onde cada biblioteca pode ter as suas políticas de gestão de utentes e de documentos totalmente distintas, independentes e autónomas sem deixar de surgir ao utente como uma única entidade: a biblioteca ou devemos dizer a bibliociberteca?</resumo>

No passado…será XML

Luís G. M. Ferreira [‡]

Docente Convidado no IPCA-ESG [†]

lufer@ipca.pt

[†] *IPCA-ESG – Instituto Politécnico do Cavado e Ave - Escola Superior de Gestão: Grupo de Informática; http://www.ipca.pt*

[‡] *CTO da Sidereus, - Sistemas de Informação e Consultoria Informática, http://www.sidereus.pt, no período de 1996-2002*

RESUMO

Não vão longe os tempos nos quais a programação de aplicações, tal qual nos haviam ensinado, era suportada ou por grandes equipas de pessoas, ditas os informáticos - ou por pedras chave assentes na polivalência e capacidade de alguns. A análise daquilo que se passa nos dias de hoje, impele-nos necessariamente a uma reflexão...

Se por um lado as empresas, em progressivas sinergias com as instituições de ensino, tendem a condicionar a forma como "fazem as coisas" para o atingir dos objectivos propostos, por outro, o técnico em quem a empresa acredita e delegou responsabilidades, joga constantemente o puzzle da tecnologia, tendo de intervir numa primeira fase, como um verdadeiro arquitecto de sistemas.

Para além da complexidade em montar plataformas de suporte, fundamentadas em análises prévias devidamente elaboradas, a grande disponibilidade de tecnologias, e subjacentes nomenclaturas e termos, tendem a "baralhar" e por vezes condicionar a aposta. Processos como a Engenharia Reversa, a Integração de sistemas (em EAI e IAI), a Revitalização de Sistemas Legados, etc., estarão fortemente condicionados, a curto ou longo prazo, por estas mesmas apostas.

O objectivo desta comunicação prende-se então com a análise de alguns casos de estudo, que nos permitirão constatar, por um lado, o desafio colocado ao programador ao ter de optar e gerir a forma de desenvolvimento, e por outro, reforçar a importância das tecnologias de Markup (hoje etiquetada com XML) que hoje apregoamos, e constatar que, embora ainda não se falasse nestes termos, parte daquilo que hoje é aceite como uma evidência, já outrora o poderá ter sido para alguns...

Considerando alguns projectos empresariais mais específicos que se desenrolaram no âmbito da actividade profissional na Sidereus, procuraremos reforçar a virtuosidade do mundo do desenvolvimento de soluções informáticas, apoiado numa multiplicidade de tecnologias e arquitecturas, perturbadas agora, pelo ressurgimento em força das tecnologias de Markup.

Iremos analisar um caso de estudo concreto, não muito recente, de interacção com uma base de dados em flat-files, com recurso a scripts e uma linguagem de markup adequada, contrapondo com a análise de outros dois casos actuais onde se cruzaram as recentes "boas novas" do XML, nomeadamente as XML StyleSheets- XSLT, com as necessárias interacções com Bases de Dados Relacionais e Bibliotecas de Templates.

Consequentemente ou não, com a inclusão destas temáticas nos programas de algumas disciplinas do curso SIG-Sistemas de Informação para a Gestão, da ESG, procuramos transmitir a importância de estarmos permanentemente atentos ao que se está a passar "lá fora" no mundo das tecnologias, e fazer sentir a necessidade de abdicar de fundamentalismos na adopção de estratégias de desenvolvimento. O XML não deve navegar sozinho...

Na essência, a tecnologia com que hoje trabalhamos não diverge muito daquela que outrora nos ensinaram; a tendência para simplificar é natural, mas nem sempre assim acontece... e como "as coisas" estão em contínua e rápida transformação, de certo no final desta nossa apresentação as coisas poderão já não ser bem assim...

Por tudo isto, julgamos ser bem mais cómodo falar do passado!

Palavras-chave

XML, Metadados; Integração de sistemas; Templates; Arquitecto de Sistemas; Gestão de Conteúdos.

INTRODUÇÃO

Ao fazermos uma retrospectiva sobre alguns anos a esta parte, especificamente no panorama do desenvolvimento de soluções informáticas, assistimos a uma contínua migração de aplicações para ambientes Web. No início deste período (e não necessitamos de retroceder no tempo mais do que 5 ou 6 anos) as aplicações começavam a aproximar-se do espectro cliente/servidor como norma a atingir…vingou a Internet e, desde logo, as inerentes consequências não pararam de ditar as regras de como "fazer as coisas"…dia-a-dia são poucas as instituições que conseguem trabalhar, de dentro-para-fora com resultados sólidos e convincentes, i.e., com iniciativas assentes na investigação que conseguem fazer…o mercado continuará a ditar as estratégias?

Fundamentalmente, pretendemos nesta comunicação mostrar a forma como foram abordados três casos de estudo reais, enquadrados, temporalmente, neste período de contínuas novas (ou diferentes) exigências de serviços e, tecnologicamente, na vigência de múltiplas tecnologias de desenvolvimento, das quais emerge em destaque, a tecnologia de Markup, hoje o XML.

Ao longo de toda esta nossa análise, reforçamos a necessidade real de nunca nos abstrairmos da dualidade de critérios ou necessidades, entre os intervenientes nestes processos, que são o cliente e o fornecedor de soluções.

No primeiro caso de estudo, abordaremos o desenvolvimento de uma plataforma de suporte de publicação periódica de conteúdos na Web associados a uma revista comum, na qual as condições a preservar ditaram fortemente o rumo a seguir;

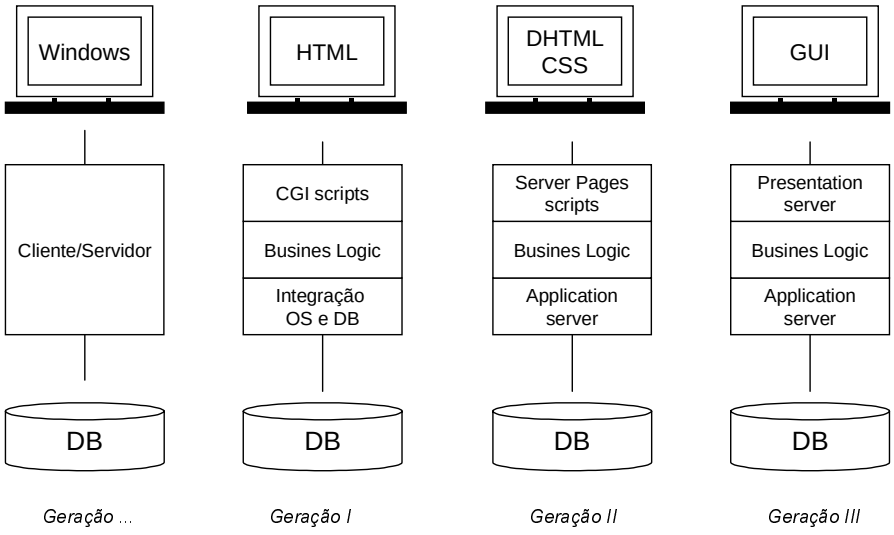


Figura 1 - Arquitecturas de desenvolvimento

Na essência do segundo caso de estudo, procuraremos mostrar como a integração com um Sistema de Gestão da Biblioteca existente, conseguida numa arquitectura de múltiplos níveis e vários componentes de software, cuja interoperabilidade está garantida pelo XML.

No terceiro caso, abordaremos uma iniciativa de reorganização interna de processos e conteúdos, inerentes a um Departamento de uma Universidade, no qual, ao ocorrerem diariamente vários processos, necessitam de permitir a interacção eficiente entre as diferentes entidades participantes (docentes, alunos, administrativos, etc.).

Na Figura 1arriscamo-nos a sintetizar as principais linhas orientadoras sobre a estruturação das soluções, inerentes à evolução seguida ao longo destes últimos anos e na qual procuraremos situar as nossas opções. Uma Geração IV estará agora a constituir-se, fortemente impulsionada pela forte (mas não exclusiva) utilização de Serviços Web – vulgo webservices (via XML_RPC ou SOAP), como canal de interoperabilidade entre os componentes de cada sistema, cada vez mais distribuídos.

ESTUDO DE CASOS

CASO I

Desafio: Publicação na Web da Revista Boletim do Contribuinte[1].

Enquadramento: O Grupo Editorial Vida Económica é detentor de um vasto e reconhecido património documental, publicamente manifestado pelas várias publicações de especialidade que criou ao longo de mais de 70 anos. Decidiram então transformar uma das publicações de forma a poder ser disponibilizada na Web. Trata-se do Boletim do Contribuinte, doravante abreviado por BC, com uma publicação bimensal e com uma tiragem de alguns milhares de exemplares.

Principais Requisitos:

- O processo de criação de cada exemplar da revista (em papel, várias folhas, sistema de livro, organizada com recurso a sistemas Macintosh e software Pagemaker) não poderia de forma alguma ser perturbado;
- Não pretendiam complicar o processo, i.e., não gerar mais trabalho para as pessoas. Era importante tentar "pegar" nos documentos que já se preparavam.
- Queriam atingir respostas rápidas a quem solicitasse informação (várias revistas cruzam informação entre si)
- Queriam consultas eficientes, geração de índices globais e cruzamento de informação;
- Cada edição na Web deveria estar plenamente sincronizada com a sua versão em papel, incluindo a disposição da informação;

Análise

- Cada edição continha cerca de 50 documentos;
- Havia uma única pessoa que "arranjava" a versão final da edição, também em Pagemaker;
- Uma edição demorava cerca de 15 dias a ser preparada;
- Os documentos estavam organizados segundo regras pouco consistentes;
- Constatou-se que cada documento poderia facilmente ser convertido em RTF.

Solução técnica

A Figura 2 pretende dar a conhecer a solução tecnológica adoptada (arquitectura típica da Geração I apresentada na Figura 1).

Num primeiro passo (❶) os documentos são escritos e convertidos em RTF. Nesta fase é necessário seguir determinadas regras para que, mesmo no RTF, possa ser introduzida meta-informação para as fases posteriores do processo [1][2].

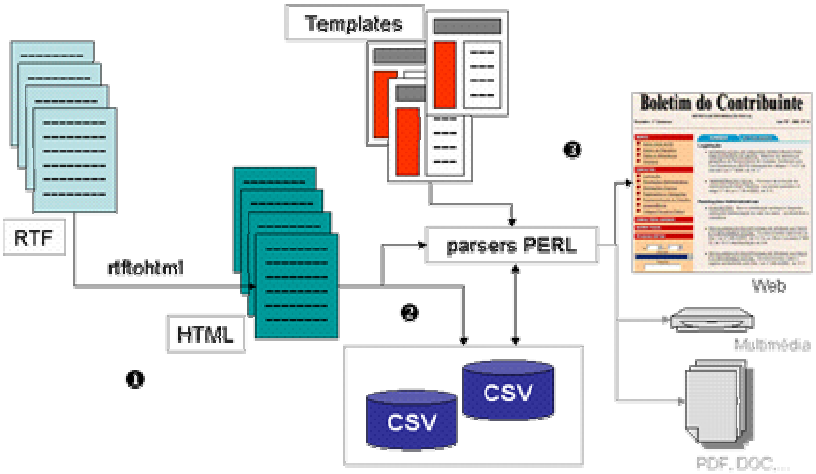


Figura 2 - Arquitectura tecnológica da solução para o BC

Algumas dessas regras ditam o próprio nome do ficheiro e a identificação de partes do documento, como o Título, Assunto, Tipo de Diploma e Autor. Por exemplo, o Assunto de um documento deveria ser escrito em fonte Arial, All Caps, tamanho 12. Esta era a forma possível de indicar ao rtftohtml[2] (software usado para a conversão dos ficheiros RTF em HTML) que identificasse correctamente esse texto. As frases a seguir representam as regras para o BC definidas no ficheiro *html-trn*, responsável por parte da configuração do rtftohtml.

```
.TTag
"Cab","<!--Cab>","<!--EndCab>"
"Ass","<!--Ass>","<!--EndAss>"
"Dip","<!--Dip>","<!--EndDip>"
.TMatch
"Arial",0,00000000000001,00000000000001,"Cab"
```

[1] http://www.boletimdocontribuinte.pt
[2] *rtftohtml - A Filter to Translate RTF to HTML - Chris Hector*

"Arial",0,00000001000000,00000001000000,"Ass"
"Arial",0,00000000000010,00000000000010,"Dip"

Do nome do ficheiro conseguia-se outra informação, também importante para a catalogação de cada documento. As regras definidas para os nomes dos ficheiros eram:

Tipo de Documento	*Temática*	*Nº de ordem*	*Nível*

em que:

Tipo de Documento:	*Um dos códigos de Tipos de Documentos;*
Temática:	*Um dos códigos de Temáticas;*
Nº de Ordem:	*Número de documentos do mesmo tipo e temática (01, 02, 03,…);*
Nível:	*O nível de utilização (a utilizar na definição de perfis).*

Exemplos:

OFRRAD0103	*Ofício Circular de Resoluções Administrativas (1º)*
OFRRAD0201	*Ofício Circular de Resoluções Administrativas (2º)*
OFILEI0103	*Ofício sobre Legislação (1º)*

Com o rtftohtml, o documento original RTF era então convertido num documento HTML muito particular. Para além das marcas (vulgo tags) normais do HTML, passava a conter também o conjunto de marcas especificamente definidas para o BC. A título de exemplo, o texto a seguir mostra o excerto de um documento HTML gerado, com um cabeçalho identificando um Assunto (IRS) e um Diploma (Circular).

<!--Cab><!--Ass>IRS<!--EndAss><!--Dip>
Circular 3, de 15.2.2002, da Direcção de Serviços
do IRS, da DGCI<!--EndDip><!—EndCab>

Esta constituía a nossa linguagem de Markup. As marcas *<!--Cab>, <!--Ass>* e *<!--Dip>*, e várias outras, constituíam os elementos do nosso DTD.

Para completar o processo, estes documentos eram processados através de parsers desenvolvidos com scripts PERL (❷).

Deste processamento resultava, por um lado, meta-informação para arquivar na base de dados, e por outro, com a aplicação de templates específicos (❸), resultavam documentos HTML prontos a ser publicados na Web. Em cada template existem marcas que ditam a localização do conteúdo dinâmico. Todo este processo assenta num mecanismo de CGI.

A base de dados era constituída por dois *CSV Flat-Files*, um mantinha todos os registos dos documentos normais inseridos, enquanto que o outro, por razões de eficiência, mantinha todos os documentos do tipo Diploma. Os documentos eram registados com duas datas, uma data correspondente à de publicação oficial do documento e outra correspondente à data da edição do BC.

Tal como se de XML StyleSheets se tratasse, estando a base de dados devidamente catalogada e disponível uma biblioteca de templates devidamente constituída, era possível uma multiplicidade de resultados. Consequentes alterações e ou criação de novos serviços, exigiam somente a definição de um novo ou alteração do template pretendido e respectiva script de manipulação dos dados.

Reflexões

Se pararmos para reflectir um pouco, apraz-nos anotar e reforçar algumas considerações:

Em termos de resultados,

- Trata-se de uma ferramenta muito importante para uso da própria instituição…
- O controlo de logs da sua utilização ao longo deste período, prova um acréscimo significativo na atenção dada a esta revista
- A publicação on-line é uma referência na área da especialidade. Seguiu os passos da sua congénere em papel.

- Anualmente conseguem-se classificar aproximadamente 5000 documentos.

Tecnologicamente,

- A necessidade de uma análise e respectiva especificação devidamente certificada (pelo cliente) mostrou-se crucial. Nem mesmo a força de uma especificação formal (adoptada) foram capazes de, por si só, garantir tudo o resto...
- Foi sugerida a actualização para uma plataforma suportada por um SGBD mais adequado;
- O "XML" utilizado provou a sua importância na sistematização de processos e produtividade atingida.
- Os parsers em PERL (o nosso XSLT) provaram que a transformação do "XML" obtido da base de dados proporcionava uma mais valia até então não experimentada. Exceptuando os elementos fixos dos templates, todos os conteúdos do site eram dinamicamente gerados.

Actualmente o projecto encontra-se em estudo para a sua migração para uma Base de Dados Relacional e para o tratamento de informação baseado mesmo em XML e seus afins. O próprio rtftohtml já exporta para formato XML.

CASO II

Desafio: Integração de Serviços na Biblioteca do IPCA - SIB.

Enquadramento: O Instituto Politécnico do Cavado e Ave, sedeado em Barcelos, representando uma instituição de ensino público, possui a Escola Superior de Gestão, doravante abreviada por ESG, como única unidade de ensino. No corrente ano já possui cerca de 1300 alunos.

Ao encontrar-se fisicamente "espalhada" por várias instalações (prédios urbanos), a instituição depara-se com alguns problemas que dificultam o seu normal funcionamento.

Uma das infra-estruturas "isoladas" é a Biblioteca. A utilização desta unidade ainda requer a deslocação física ao espaço constituído para o efeito.

Embora já existisse a infra-estrutura de rede montada, abrangendo praticamente todas as unidades funcionais, faltava ainda a disponibilização remota dos serviços essenciais, tanto a docentes como a alunos[8].

Principais Requisitos:

- A Biblioteca possui desde há algum tempo o Software de Gestão de Bibliotecas da PORBASE;
- Os módulos do software de gestão que a Biblioteca possui, não respondem a todas as solicitações;
- Pretendia-se disponibilizar remotamente a consulta e requisições de livros, para além de outros serviços essenciais neste contexto;
- O utilizador final acederá à aplicação através de terminais a instalar no espaço físico da instituição.
- O sistema operativo de suporte é o Windows.

Análise

- Existe uma Base de Dados CDS/ISIS[3] que suporta a informação sobre os livros existentes;
- Existe a versão Porbase 5.0 que gere os actuais processos da biblioteca;
- Existe também uma base de dados relacional utilizada pelo Porbase para suportar outras informações para além dos livros: contas de utilizadores, gestão, etc.

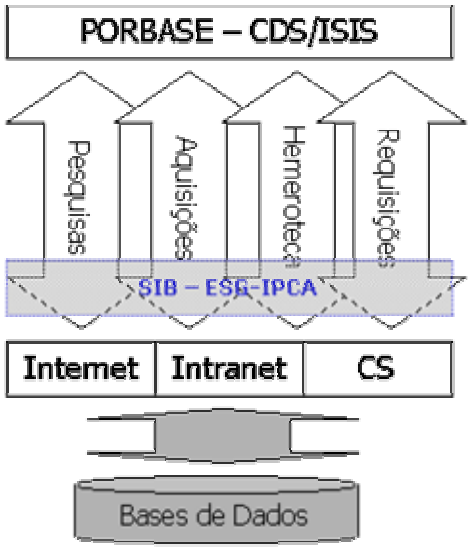


Figura 3 - Arquitectura da solução SIB

Solução técnica

A Figura 3 pretende mostrar a solução tecnológica adoptada (arquitectura típica da Geração III apresentada na Figura 1), na qual se identifica o posicionamento da solução a implementar (SIB).

De uma forma geral, podemos pensar numa base de dados CDS/ISIS como um ficheiro de dados que armazenamos para conter informação associada a uma temática em particular. Podemos considerar como

[3] Desenvolvido pela UNESCO

exemplos um ficheiro de endereços ou um catálogo bibliográfico. Cada unidade de informação guardada consiste num dado elementar, com uma característica particular duma entidade descrita. Por exemplo, uma base de dados bibliográfica poderá conter informação de livros, relatórios, artigos de jornal, etc. Cada unidade terá, neste caso, um conjunto de elementos, como autor, título, data de publicação, etc.

Os dados são guardados em **campos**, cada um deles identificado por etiquetas numéricas identificativas do seu conteúdo[4]. Pode pensar-se numa etiqueta como um nome de um campo tal como é conhecido pelo CDS/ISIS.

A colecção de campos que armazenam todos os elementos da base de dados de uma unidade de informação é designada **registo.**

Sendo esta especificação de âmbito não comercial, existe disponível bibliografia [3][4] considerável, que expõe devidamente a abordagem de uma base de dados deste tipo, tanto de uma forma directa como através de mecanismo CGI, por nós adoptado.

Toda a análise de requisitos e especificação associada foram feitas através da utilização de *Uses Cases* do UML, tal e qual, como se evidencia na Figura 4, na qual se descrevem os limites de actuação de um dos intervenientes no processo. Neste caso, o Administrador da aplicação através de um Backoffice. O mesmo foi feito para os restantes utilizadores (Utilizador Registado, Utilizador normal e Bibliotecário).

Este projecto foi entendido como uma instância de um *DCM - Dynamic Content Management*, área à qual estrategicamente nos decidimos dedicar.

É claro que se trata de um processo típico de integração entre diferentes sistemas, que permitiu ensaiar a criação de um framework assente em componentes plenamente configuráveis e integráveis.

A tecnologia de suporte adoptada, .NET da Microsoft, resultou pela disponibilidade de vários componentes e pela capacidade de se constituírem outros. O desenvolvimento assentou em C#.

A ideia de componentes assenta fundamentalmente no enriquecimento do middleware que fazem a ponte entre os níveis I e II da arquitectura multinível seguida. Todo este processo de criação de novos módulos está completamente documentada em [10]. Neste contexto, concentramo-nos na parte do projecto que identifica a interacção com a BD CDS/ISIS.

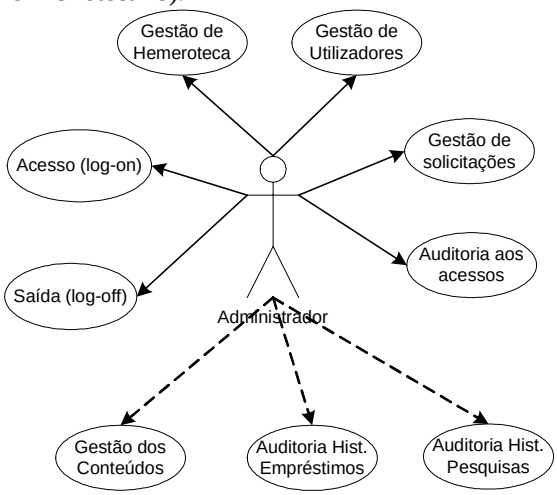


Figura 4 - Use Case para o Administrador (Módulo BackOffice)

Com a utilização de DLLs disponíveis, que garantiam a interacção com a BD ISIS, um parser em C# gerava um documento XML com um schema específico a partir de código UNIMARC. O esquema seguinte mostra um pequeno excerto XML gerado.

```
…
<RECORD Mfn="1184">
        <Tag_200><Tag_200_a>Como se faz uma tese…</Tag_200_a></Tag_200>
        <Tag_100><Tag_100_a>20010921d1998</Tag_100_a></Tag_100>
        ....
</RECORD>
…
```

Nesta fase, era necessário a visualização de resultados, conseguidos com a aplicação de uma stylesheet (*xslrecord.xsl*). Um dos resultados possíveis pode ser observado no excerto seguinte.

```
<xsl:template match="RECORD">
    <xsl:value-of select="Tag_200/Tag_200_a"/>
    MFN: <xsl:value-of select="@Mfn"/></b>
    Data Aq.: <xsl:value-of select="Tag_100/Tag_100_a"/>
    Titulo:<xsl:value-of select="Tag_200/Tag_200_a"/>
    Autor: <xsl:value-of select="Tag_200/Tag_200_f"/>
    ISBN: <xsl:value-of select="Tag_10/Tag_10_a"/>
```

[4] Neste caso seguindo a especificação UNIMARC para identificação dos campos

```
        Valor (Aquisicao): <xsl:value-of select="Tag_10/Tag_10_d"/>
        Local: <xsl:value-of select="Tag_210/Tag_210_a"/>
        Editora: <xsl:value-of select="Tag_210/Tag_210_c"/>
        Data Lancamento: <xsl:value-of select="Tag_210/Tag_210_d"/>
        Info. publicacao: <xsl:value-of select="Tag_215/Tag_215_a"/>
        Tamanho publicacao: <xsl:value-of sect="Tag_215/Tag_215_d"/>
        CDU: <xsl:value-of select="Tag_675/Tag_675_a"/>
        Nr. Registo: <xsl:value-of select="Tag_966/Tag_966_a"/>
        Nr. Exemplares: <xsl:value-of select="Tag_966/Tag_966_c"/>
        Localizacao: <xsl:value-of select="Tag_966/Tag_966_l"/>
        Notas: <xsl:value-of select="Tag_966/Tag_966_n"/>
        Cota: <xsl:value-of select="Tag_966/Tag_966_s"/>
</xsl:template>
```

Para melhor percepção do resultado obtido com a querie à BD, a Figura 5 mostra o UNIMARC na base de dados ISIS que descreve o livro pretendido.

```
MFN: 1184
Estado: Tipo: Nível bibl.:
Nível hierárquico: Nível de cod.: Forma de desc.:

010: ^a972-23-1351-17^bbrochado^dcompra
100: ^a20010921d1998 k y0pory0103 ba
101:1 ^apor^cita
102: ^aPT
200:1 ^aComo se faz uma tese em Ciências Humanas^fUmberto Eco^gpref. de Hamilton
Costa^gtrad. Ana Falcão Bastos, Luís leitão
205: ^a7ª ed.
210: ^aLisboa^cEditorial Presença^djan. 1998
215: ^a235 p.^cil.^d21 cm
225:2 ^aUniversidade Hoje^v4
300: ^aTit. orig.: Como si fa una tesi di laurea.- cop. 1977
675: ^a001.8^vmed^zpor
700: 1^aEco^bUmberto
702: 1^aCosta^bHamilton^4080
702: 1^aBastos^bAna Falcão^4730
702: 1^aLeitão^bLuís^4730
930: ^mMaria José
966: ^a1124^c1^lIPCAB^n19991026^sBESG 001.8
```

Figura 5 - UNIMARC associado a um livro

O resultado da aplicação da referida stylesheet mostra-se na Figura 6, que representa parte de uma página Web.

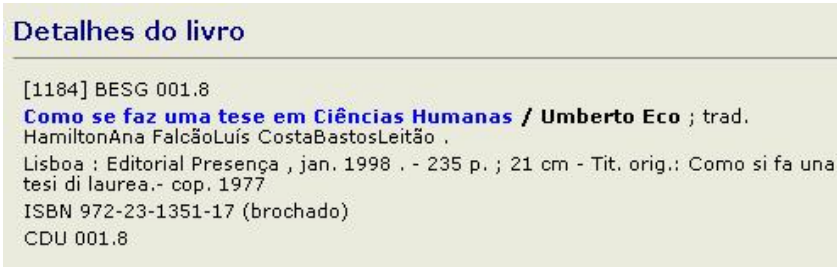


Figura 6 - Exemplo de aplicação de stylesheets na solução SIB

Reflexões

A tecnologia .NET da Microsoft evidenciou claramente aquilo que já é conhecido…De facto apresenta serviços e capacidades muito interessantes mas aparece desprotegida para enfrentar uma programação exigente. A documentação é escassa e muitas vezes inconsistente….

Aquilo que de facto se conseguia manter "afinado" terá sido tudo que nós programava-mos…o abdicar de demasiada dependência do CVL do Visual Studio .NET (só em alguns serviços se utilizaram os serviços das *Datagrids*) foi uma opção necessária. Ou seja, é importante trabalhar com componentes que já existam mas devemos ser cautelosos…

Resultou deste trabalho um framework – hyPortal – que está a ser utilizado pela própria intranet da Sidereus. Uma solução agradável como suporte a um DCM [10].

Importante foi a "migração" para uma programação sob o paradigma *CBD – Components Based Development* [9]. Foi necessário um setup considerável…

O XML voltou a marcar pontos…significou método e ordem na programação. Possibilitou uma vez mais a estruturação da equipa de desenvolvimento com responsabilidades diferentes…O Visual Studio .NET apresentou aqui uma mais valia importantíssima…a organização da documentação do código fonte!

CASO III

Desafio: Integração de serviços do Departamento de Matemática da Universidade do Minho

Enquadramento: Na averiguação das actuais necessidades e solicitações dos membros do departamento, foi decidido a criação de um Sistema de Informação Interno via Web – SIIW, vulgo *Intranet* com serviços integrados, que possibilitasse a gestão da informação associada a processos inerentes à actividade lectiva do departamento. Basicamente, pretendia-se reorganizar toda a informação associada aos cursos e respectivas disciplinas, aos docentes e restantes colaboradores do departamento.

Principais Requisitos:

- O sistema a desenvolver teria de ser suportado pelo sistema operativo Linux e teria como clientes qualquer browser em qualquer tipo de sistema operativo;
- O novo sistema teria de se integrar com as soluções informáticas existentes, nomeadamente permitir a utilização de templates já utilizados na elaboração de documentos internos;
- O sistema teria de ser configurável e autónomo;
- O sistema teria duas instâncias diferentes e autónomas, embora com informação cruzada, uma para o Departamento de Matemática e outra para o Centro de Matemática.
- Teria de se manter o histórico e registar todas as operações efectuadas no sistema;

Análise

- O processo de análise assentou num guião fornecido pelo Departamento;
- A informação e processos associados encontravam-se devidamente estruturados;
- Pretendiam-se três perfis de utilizador: Convidado, Utilizador registado e Administrador;
- A imagem do sistema (aspecto gráfico) não seria contemplado neste desenvolvimento, daí que a infra-estrutura a montar, deveria permitir a adaptação posterior de tratamentos gráficos.
- Existia muita informação espalhada, muita da qual se encontrava em folhas de cálculo;

Solução técnica

A Figura 7 pretende dar a conhecer a solução tecnológica adoptada para o SIIW (arquitectura típica da Geração III apresentada na Figura 1).

Procurou-se neste projecto continuar a avançar na área de DCM, tal como se havía feito com o projecto analisado no caso de estudo anterior desta comunicação. Pretendia-se criar um framework [6], capaz de gerar sistematicamente, entre outras, as componentes associadas aos primeiros níveis da arquitectura, i.e., classes que permitissem a interacção com a Base de Dados (BD)[5].

O framework foi desenvolvido em JAVA e apoiava-se no XML como "língua franca" na interacção entre os vários componentes. As servlets JAVA criadas era processadas pelo servidor Web Tomcat.

De entre os vários pacotes que constituem o framework,, o *extractor* contém a classe DBE que é responsável pela extracção do esquema da base de dados para um ficheiro XML que o descreve (*schemaDB.xml*). O pacote *enterpriseengine, ao ser* responsável pela

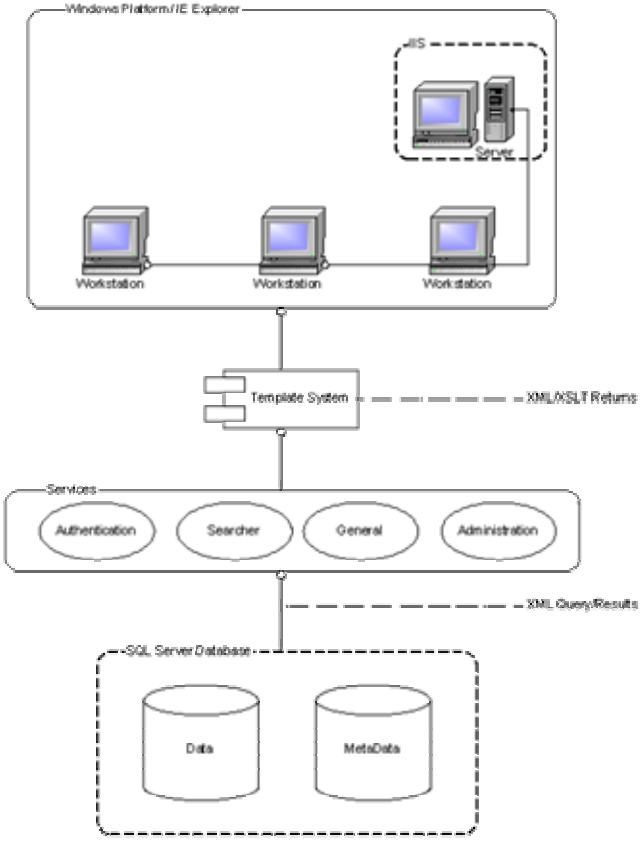


Figura 7 - Arquitectura da solução SIIW

administração do esquema da BD, garante as operações de geração de formulários, manutenção da meta-informação referente às tabelas e seus campos e geração do front-end do backoffice.

```
<DATABASE>

<TABLE catalog="null" schema="null" name="pga_scripts" type="null" remarks="no remarks">
<FIELD name="scriptname" type="12" typename="varchar" size="64" nullable="YES"/>
<FIELD name="scriptsource" type="12" typename="text" size="-1" nullable="YES"/>
</TABLE>

<TABLE catalog="null" schema="null" name="professores" type="null" remarks="no remarks">
<FIELD name="id" type="12" typename="varchar" size="50" nullable="NO"/>
<FIELD name="nome" type="12" typename="varchar" size="100" nullable="YES"/>
<FIELD name="morada" type="12" typename="varchar" size="100" nullable="YES"/>
</TABLE>

<TABLE catalog="null" schema="null" name="horarios" type="null" remarks="no remarks">
<FIELD name="id" type="12" typename="text" size="-1" nullable="YES"/>
<FIELD name="designacao" type="12" typename="varchar" size="122" nullable="YES"/>
<FIELD name="hora" type="12" typename="varchar" size="10" nullable="YES"/>
<FIELD name="data" type="12" typename="varchar" size="10" nullable="YES"/>
</TABLE>

</DATABASE>
```

Exemplo de um schemaDB.xml

Com mais meta-informação (*MetaDataTables.xml*) entretanto conseguida com a ajuda de um *wizard*, consegue-se a geração de stylesheets para cada entidade tabela, com as operações pretendidas entre *Inserir*, *Alterar*, *Remover*, *Listar*, *Imprimir* ou *Exportar*. Este processo é garantido pela *stylesheet MetaDataTables.xsl***.** Com isto, fica preparada a criação dos respectivos formulários.

Pormenores como o tratamento de campos tipo *Date*, cruzamento entre tabelas com chaves estrangeiras (que poderão resultar *Comboboxes* no front-end), regras de Validação de dados (parte a aplicar via *Javascript*), campos calculados, etc.., estão em fase de desenvolvimento.

```
<METADATATABLES>
<TABLE formName="tiposcurso" show="yes" name="tiposcurso">
<FIELD nullable="NO" size="4" formName="tiposcursoid" name="tiposcursoid"></FIELD>
<FIELD nullable="NO" size="60" formName="ticudesc" name="ticudesc"></FIELD>
<FIELD nullable="YES" size="60" formName="ticudescen" name="ticudescen"></FIELD>
<OPERATIONSshow="yes" name="Inserir"></OPERATIONS>
<OPERATIONS show="yes" name="Alterar"></OPERATIONS>
<OPERATIONS show="yes" name="Remover"></OPERATIONS>
<OPERATIONS show="yes" name="Listar"></OPERATIONS>
<OPERATIONS show="yes" name="Imprimir"></OPERATIONS>
<OPERATIONS show="yes" name="Exportar"></OPERATIONS>
</TABLE>
</METADATATABLES>
```

MetaDataTables.xml resultante da aplicação da stylesheet MetaDataTables.xsl ao ficheiro schemaDB.xml

Paralelamente foram criados os processos que garantem a definição das "regras de negócio" (nível II da arquitectura da Geração III, na Figura 1), que permitiriam montar a solução pretendida [7].

Com os *parsers* desenvolvidos em PHP, processa-se o XML resultante das *queries* feitas à BD. Com *stylesheets* adequadas e templates devidamente estruturados, todo o *front-end* é dinamicamente gerado, desde menus contextualizados até resultados em HTML ou PDF.

Na próxima sequência de excertos de código pretende-se mostrar todo o processo de geração de um resultado numa das páginas Web que constituem o sistema, partindo do XML devolvido pelas classes de manipulação da BD.

```
<table class="Table">
  <tr><td class="Header" colspan="2" ><!--{header}-->
  <tr><td class="Menu" colspan="2" ><!--{buttons}-->
  <tr><td class="Label">Nome: </td><td class="Content"><!--{nome}-->
  <tr><td class="Label">Descrição: </td><td class="Content">{desc}-->
  <tr><td class="Label">Url: </td><td class="Content"><!--{url}-->
  <tr><td class="Label">Código: </td><td class="Content"><!--{codigo}-->
  <tr><td class="DisciplinasTitulo" colspan="2" ><!--{titulo}-->
  <!--{disciplinas}-->
</table>
```

Template cursos.tpl

Cada template é previamente especificado e definido com recurso a marcas para posteriores substituição por conteúdos. Neste caso trata-se de um template que prepara uma listagem em forma tabelar, com integração de alguns serviços de manipulação da tabela.

```
....
<xsl:template match="row[position() mod 2 = 1]">
    <tr class="Impar">
        <xsl:call-template name="doinfo"/>
    </tr>
</xsl:template>
<xsl:template name="doinfo">
    <xsl:for-each select="field">
        <xsl:if test="position()=3">
            <td class="ListContent">
                <a>
                    <xsl:call-template name="url"/>
                    <xsl:value-of select="following-sibling::*"/>
                </a>
            </td>
        </xsl:if>
            <xsl:if test="position()=1">
                <td class="ListContent"><xsl:value-of select="."/></td>
            </xsl:if>
    </xsl:for-each>
</xsl:template>

<!-- template para criar as checkboxes -->
<xsl:template name="checkbox">
    <td>
        <input type="checkbox"
            name="chk{@id}"
                value="chk{@id}"/>
    </td>
</xsl:template>
....
```

StyleSheet cursos.xsl

Uma biblioteca de stylesheets permite a transformação dos resultados conseguidos e apresentá-los segundo o template a utilizar. Um API disponibilizado pelo Tomcat em PHP possibilita esta transformação (neste caso, o interface *ArrayToHtml* responsabiliza-se por essa transformação. As próprias classes CSS podem assim ser atribuídas isoladamente. Repare-se na distinção de classes CSS a atribuir a uma linha (row) par ou impar (row[*position() mod 2 =1]*).

```
...
function show_servico_normal($conn, $curso, $id){
    $template = readtemplate("templates/cursos.tpl");
    $template = ereg_replace("{nome}",$curso[2], $template);
    $template = ereg_replace("{descricao}", $curso[4], $template);
    $url = criaLink($curso[6]);
    $template = ereg_replace("{url}", $url, $template);
    $template = ereg_replace("{codigo}", $curso[7], $template);
    $query = "select D.disnome, D.disciplinasid, DC.cursosid,...";

    $xml2 = GetQuery($conn,$query);
    $disc  = getsqlinfo($xml2);
    $html = ArrayToHtml($disc);
    $size = count($disc);
```

```
    if ($size > 1) {
            $template = ereg_replace("{titulo}", "Disciplinas", $template);
    }
    else {
            $template = ereg_replace("{titulo}", "", $template);
    }
    $template = ereg_replace("{disciplinas}", $html, $template);
    return $template;
}
…
```

Módulo show_serviços.php

O resultado obtido neste exemplo pode ser analisado na imagem seguinte. As checkboxes representam métodos que ficam disponíveis para a manipulação de cada registo.

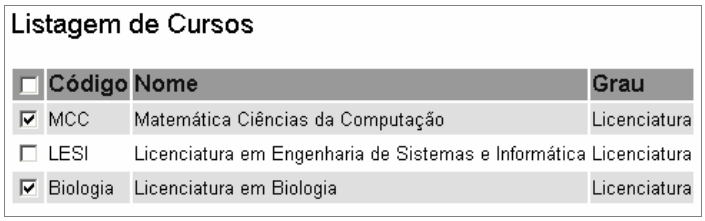


Figura 8 - Resultado de uma querie transformado em HTML

A opção pela utilização de templates garantiu a facilidade de utilização deste sistema, possibilitando que, mesmo os relatórios pudessem ser facilmente configuráveis. Todo o código HTML gerado, estava devidamente estruturado com classes CSS ou mesmo secções DIV. Por conseguinte, até mesmo o cliente poderia ensaiar mudanças a nível de pormenores gráficos

Reflexões

Demonstrou-se a vitalidade da utilização de uma linguagem de Markup como o XML para servir de "idioma" comum entre múltiplas peças que necessitam de interagir.

Provou-se também que o XML não consegue navegar sozinho, comprovado pela necessidade de utilização do próprio *Latex* para a criação de relatórios em PDF (de forma a garantir similaridade com os timbrados existentes). A nossa veia académica…

Provou-se também que, a aposta em tecnologias é uma tarefa séria e delicada. Neste pequeno projecto a variante tecnológica posta em jogo: *Servelets*, JAVA, XML e PHP, evidencia isso mesmo… não menos sério, é também o garantir da interoperabilidade numa arquitectura de desenvolvimento apoiado em componentes. O XML mostrou-se uma excelente opção…contudo, a necessidade constante de processar XML pode conduzir a pequenos atrasos nas respostas….

A especialização e a reutilização de código provaram ser uma mais valia necessária e imprescindível para garantir qualidade e produtividade.

CONCLUSÕES

Em primeiro lugar apraz-nos anotar que todo este percurso vem uma vez mais provar a rotatividade inerente à investigação. Novas necessidades, novas soluções, novas tecnologias,…

Estaremos perante o conceito da original função ou subrotina presente nos vários módulos, carregados num trama de processo de RAM que já não é "só nossa" e cujo o espaço de endereçamento é global a todo um universo magnífico de "memória" distribuído por esse mundo informático.

Também a ideia de estarmos na fase obrigatória de "pensarmos" em componentes não é nova, tal como não o era o "pensar" em objectos. Tal como esta, também o CBD está na fase de "ter que provar" na prática que merece a credibilidade que a teoria que lhe é adjacente tenta apregoar.

Num mundo actual de *Solutions and Components Providers*, o campo de batalha assenta na razão entre produtividade e qualidade oferecida, e na forma eficiente de garantir a interoperabilidade, mesmo com Sistemas

Legados. O XML mostra-se cada vez mais uma ferramenta adequada, fortemente vinculada com a adesão ao SOAP pelos mais importantes intervenientes. Contudo, se a reacção ao SGML não terá sido a melhor, o XML poderá correr o mesmo risco…ainda não é tudo simples…sê-lo-á alguma vez?

Fica também anotado que, o navegar sozinho só com o "*astrolábio"* XML, tal como diria o nosso amigo navegador, pode levar-nos de facto a uma ilha…mas não ainda à terra firme tão almejada.

Continua a parecer ser mais fácil falar do passado…

AGRADECIMENTOS

Gostaria de deixar aqui os meus agradecimentos aos técnicos António Soares, Jorge Ribeiro, António Gonçalves, Paulo Correia e Filipe Campos que, com os seus trabalhos de investigação persistente, conseguiram abordar estes projectos, num leque de muitos outros, com eficiência e dedicação. Também graças a eles esta comunicação foi possível.

REFERÊNCIAS

[1] - **Classificação de documentos no BC** - documento técnico de desenvolvimento da Sidereus , LF;

[2] - **Especificação de Documentos para conversão de RTF para HTML** - documento técnico de desenvolvimento da Sidereus, LF.

[3] - **WWWISIS: Metodología para la instalación de bases de datos CDS-ISIS en el World Wide Web** – CONICYT.

[4] - **The CDS/ISIS for Windows Handbook** - Andrew Buxton, Alan Hopkinson – UNESCO.

[5] - **SIIW - Sistema de Informação Interno via Web para o Departamento de Matemática da UM** - documento técnico de desenvolvimento da Sidereus, LF.

[6] - **Arquitectura para prototipagem rápida de aplicações baseadas em HTML replicando a estrutura de uma base de dados** – documento técnico de desenvolvimento da Sidereus, JR.

[7] – **Documentação Técnica do projecto DMUM** - documento técnico de desenvolvimento da Sidereus, APS.

[8] – **Documentação Técnica do projecto ISB-IPCA** - documento técnico de desenvolvimento da Sidereus, AJG.

[9] – **"Interact – Component-Based Development & Integration",** na Web em www.cbdiforum.com.

[10] - **"hyPortal" Framework -** documento técnico de desenvolvimento da Sidereus, AJG.

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

13 Fev.	S4(16.30h) Tecnologia e Web Services	[*Utilização do UDDI no suporte à descoberta de serviços baseados na localização*] - **Noé Vilas Boas**, DSI-UM; **Helder Pinto**, DSI-UM; **Rui José**, DSI-UM; [*Actualização de Dados de GIS usando Assistentes Digitais Pessoais*] - **Henrique Silva**, ISMAI; **Alexandre Valente de Sousa**, ISMAI; **João Correia Lopes**, FEUP-UP; [*Engenharia reversa de HTML usando tecnologia XML*] - **José João Almeida**, DI-UM; **Alberto Simões**, DI-UM;

Utilização do UDDI no suporte à descoberta de serviços baseados na localização

Helder Pinto, Noé Vilas Boas e Rui José

Departmento de Sistemas de Informação
Universidade do Minho
Azurém, 4800-058 Guimarães, Portugal
{helder,noe,rui}@dsi.uminho.pt

Resumo Este artigo explora a utilização do registo UDDI como plataforma de suporte à descoberta de serviços baseados na localização. O registo UDDI é essencialmente vocacionado para o registo e pesquisa de serviços eminentemente globais e apresenta diversas limitações quanto ao suporte à descoberta baseada na localização, nomeadamente a falta de normas para a associação dos serviços a localizações de dimensão reduzida e a inadequação do processamento das pesquisas aos requisitos da descoberta assente em modelos de proximidade baseados na distância ou no âmbito dos serviços. Este artigo apresenta uma solução proposta com o objectivo de aproveitar as vantagens dos Web Services e do registo UDDI em termos de interoperabilidade. Com a solução proposta, o uso destas tecnologias é beneficiado por uma arquitectura mais adequada à descoberta de serviços baseados na localização.

1 Introdução

A proliferação de dispositivos móveis e redes sem fios despertou o interesse pelo acesso à informação em qualquer lugar e em qualquer momento. Os utilizadores móveis têm, em resultado da sua mobilidade, uma relação mais próxima com o ambiente físico e, embora continuem a necessitar de aceder a informação global, têm uma forte necessidade de informação que esteja associada ao seu ambiente. Esta necessidade levou ao desenvolvimento de modelos que suportem o fornecimento de informação baseada na localização, nos quais os serviços de informação estão associados a determinadas localizações ou posições geográficas. Um desafio fundamental no desenvolvimento de sistemas baseados na localização é a forma como se lida com as questões da abertura e da heterogeneidade no fornecimento de informação por parte de entidades terceiras. Qualquer entidade fornecedora de informação de âmbito local deveria ter a possibilidade de publicar e tornar acessível o seu serviço de forma aberta. Dada a heterogeneidade que caracteriza as plataformas tecnológicas dos diversos actores de um sistema baseado na localização, este deve assentar sobre um conjunto de normas que permitam a interoperabilidade na publicação, no fornecimento e no acesso à informação. Outro desafio colocado por este tipo de sistemas tem a ver com a capacidade das aplicações baseadas na localização descobrirem os serviços de que fazem uso

em função da localização do utilizador. A descoberta de serviços é um tema que foi alvo de bastante interesse em anos recentes, tendo sido desenvolvidas diversas normas e tecnologias com abordagens muito diversas. No que se refere à descoberta de serviços baseados na localização, foram também já propostas diversas formas de a suportar nos sistemas existentes ou mesmo propostos sistemas especificamente desenhados para esse fim [3].

No projecto VADE [2], tem vindo a ser desenvolvido um conjunto de sistemas com o objectivo de fornecer o acesso a informação baseada na localização a utilizadores móveis. O conceito de um Value ADded Environment (VADE) é o de um espaço físico como, por exemplo, um centro comercial, no qual os dispositivos móveis beneficiam de um conjunto alargado de serviços, explorando a complementaridade entre portais móveis globais e serviços locais. Para suportar essa funcionalidade, é importante poder associar serviços aos diversos espaços existentes num VADE e fazer com que a aplicação vá seleccionando os serviços adequados. Tendo por objectivo o desenvolvimento de uma infraestrutura aberta e baseada em normas que permitam a interoperabilidade, optou-se por fornecer informação baseada na localização através de Web Services. Os Web Services fornecem um mecanismo programático de acesso à informação e são uma abordagem particularmente interessante para lidar com estas questões, dada a sua elevada interoperabilidade e a sua natureza fracamente acoplada. A abordagem adoptada no projecto VADE para a publicação e descoberta dos serviços passou pela utilização da norma Universal Description, Discovery and Integration (UDDI) [6]. Assim, uma aplicação baseada na localização, ao detectar a sua entrada num VADE, iria utilizar um registo UDDI local para realizar a pesquisa de serviços associados à localização do utilizador. Embora o conjunto de normas UDDI seja essencialmente vocacionado para o registo e pesquisa de serviços eminentemente globais, i.e. independentes da localização, a utilização de Web Services e do UDDI como forma de modelação e descoberta de serviços baseados na localização é extrememante atraente pelo seu enorme potencial em termos de interoperabilidade.

O objectivo deste artigo é estudar a adequabilidade do UDDI para o suporte à descoberta de serviços baseados na localização, identificando limitações e apresentando a abordagem proposta para suplantar tais limitações. O artigo está organizado da seguinte forma: a secção seguinte aborda as questões associadas à descoberta de serviços baseados na localização; a secção 3 descreve o UDDI e a forma como este é utilizado para suportar o registo e a descoberta de serviços, sendo identificadas as limitações desta tecnologia quanto ao suporte às especificidades da descoberta de serviços baseados na localização; a secção 4 apresenta a solução proposta para ultrapassar as limitações do UDDI; e a secção 5 encerra o artigo com os resultados e as conclusões do trabalho.

2 Descoberta de serviços

Antes de considerar os aspectos directamente relacionados com a forma como o UDDI é utilizado para suportar a descoberta de serviços baseados na localização,

convém compreender a forma como se associam os serviços ao espaço físico e que critérios utilizar para determinar os serviços que se encontram associados a localizações na proximidade do utilizador. Este trabalho considera dois modelos de proximidade distintos: baseado na distância e baseado no âmbito. No modelo baseado na distância, o cliente selecciona os serviços associados a localizações dentro dos limites de uma dada distância a partir da posição do utilizador. Essa distância pode ser estática ou variar em função das necessidades do cliente. Este modelo é o mais adequado para informação fortemente associada a um ponto específico no espaço: tipicamente serviços que agem como representações de entidades reais, e.g. um terminal ATM ou uma paragem de autocarro. A principal limitação deste modelo é o facto de, à medida que se alarga a distância, haver uma proporção menor de serviços relevantes para o contexto do utilizador, i.e. surge na proximidade do utilizador uma maior quantidade de serviços que não o interessam. Outro aspecto limitativo deste modelo tem a ver com a menor adequação da distância para servir de base à definição do contexto do utilizador. Por exemplo, um utilizador numa auto-estrada não estará necessariamente interessado nas estações de serviço na vizinhança da sua localização.

O segundo modelo de proximidade considerado é o modelo baseado no âmbito. Esta abordagem considera que cada serviço está disponível para uma determinada área, não dependendo da distância entre o cliente e o serviço. O âmbito do serviço corresponde portanto à representação de uma área no espaço físico para a qual o serviço está disponível. Neste modelo, os clientes descobrem os serviços cujo âmbito inclua a localização do utilizador. Isto implica que a localização do serviço seja irrelevante para a sua descoberta, passando a ser importante a área para o qual o serviço é passível de ser descoberto. Este modelo garante que qualquer serviço descoberto, independentemente da distância, tem relevância para a localização do utilizador. Este modelo é o mais adequado para a disponibilização de informação associada a uma área geográfica, mas não a um ponto específico no espaço, como, por exemplo, mapas, informação meteorológica, ou páginas amarelas. O principal inconveniente deste modelo é o facto do cliente perder o controlo sobre a definição do intervalo de proximidade desejado para a descoberta de serviços, passando a ser uma entidade passiva que depende dos serviços que cubram a localização do utilizador. De modo a tornar o cliente mais activo e flexibilizar a descoberta de serviços, pode ser introduzida neste modelo a possibilidade do cliente definir a escala do âmbito de cobertura dos serviços que mais lhe convém. Por exemplo, duas pessoas na mesma localização podem estar interessadas em mapas com um âmbito diferente, em que um representa a planta de um edifício, e o outro representa o mapa do quarteirão onde o edifício se encontra.

No desenvolvimento de um sistema que suporte a descoberta de serviços baseados na localização, a consideração destes dois modelos é importante pelo facto de serem abordagens complementares na satisfação de necessidades específicas de informação. No entanto, o desenvolvimento destes sistemas levanta diversas questões, sobretudo quando se tratam de sistemas baseados em tecnologias especificadas para o suporte à descoberta de serviços independentes da

localização, como é o caso do UDDI. A próxima secção descreve o UDDI e analisa as questões específicas desta tecnologia e as respectivas limitações quanto ao suporte à descoberta de serviços baseados na localização.

3 Uso da localização no UDDI

O UDDI consiste numa plataforma de suporte à publicação e descoberta de recursos, com o objectivo de fomentar a interoperabilidade e a adopção dos Web Services. A abordagem do UDDI baseia-se num registo distribuído de organizações e descrições dos respectivos serviços, implementado num formato XML comum. A componente principal do UDDI é o registo, que corresponde a um documento XML utilizado para descrever uma organização e os seus Web Services. Conceptualmente, a informação fornecida por um registo de uma organização no UDDI consiste em três componentes: "páginas brancas"que incluem endereço, contactos, e identificadores; "páginas amarelas"que incluem categorizações baseadas em taxonomias; e "páginas verdes"que incluem referências para especificações de Web Services. Estas três componentes conceptuais estão, na prática, implementadas no formato XML através de quatro elementos básicos (ver figura 1) que contêm informação sobre a organização (*businessEntity*), informação sobre os serviços (*businessService*), informação sobre o acesso aos serviços (*bindingTemplate*), e informação sobre a especificação dos serviços (*tModel*).

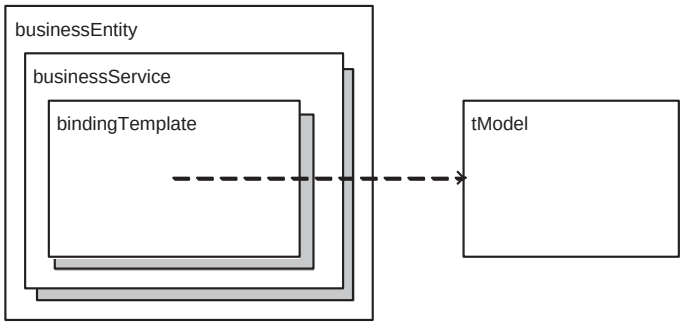


Figura 1. Modelo de informação do registo UDDI

A informação presente em cada um dos elementos da estrutura UDDI inclui o suporte para taxonomias de categorização, de forma a suportar a classificação e a pesquisa de organizações e serviços baseadas em determinada categoria (e.g. de negócio, de produto, ou de localização geográfica). Essas taxonomias podem ser normalizadas ou privadas, permitindo que sejam desenvolvidas taxonomias para quaisquer tipos de categorização. Adicionalmente, os serviços podem referenciar documentos que especifiquem um determinado tipo de serviço, através do uso dos *tModels*. Isto permite que as aplicações possam pesquisar serviços compatíveis com uma especificação conhecida *a priori*.

O UDDI possui, para além de um formato comum para o registo de organizações e Web Services, uma API que permite a publicação e consulta no registo. Os utilizadores de determinado UDDI podem, através da API, publicar informação sobre organizações, Web Services, e especificações técnicas de Web Services. Podem igualmente efectuar pesquisas de organizações no registo, seja por nome, seja por categorização, seja por identificador, seja por especificação técnica de serviços.

A associação entre os Web Services e as localizações é suportada no UDDI através do uso de categorizações geográficas. A especificação do UDDI referencia a taxonomia normalizada ISO 3166 [4] para a categorização geográfica de elementos UDDI. Esta taxonomia permite associar serviços a localizações cuja granularidade varia entre países, e que, por exemplo, em Portugal, permite apenas associar um serviço à totalidade do território nacional. O registo UDDI da Microsoft [5] utiliza uma outra taxonomia, onde a categorização espacial pode atingir a dimensão da cidade, não sendo, no entanto, exaustiva na cobertura do território nacional. Surge então a questão de como associar serviços cujo âmbito seja o de um espaço inferior ao de uma cidade ou, baseando-nos no cenário de um VADE, um *campus* universitário, um edifício ou uma sala. Além disso, também se levanta o problema de como associar um serviço a um determinado ponto no espaço, de modo a suportar um modelo de proximidade baseado na distância.

A descoberta de serviços baseados na localização usando o UDDI e pondo em prática os dois modelos de proximidade anteriormente identificados coloca algumas questões adicionais. Por exemplo, considerando o modelo baseado no âmbito, quando um utilizador está localizado numa sala, procura um serviço, e não encontra nenhum associado àquela sala, o sistema deveria possibilitar a pesquisa de um serviço associado ao piso ou ao edifício onde a sala se inclui. O mesmo deveria acontecer na situação na qual um utilizador foi localizado num dado edifício e desejaria encontrar serviços associados a salas. No caso do modelo baseado na distância, o sistema deveria ser capaz de determinar a distância entre a localização do utilizador e a localização do serviço. Ora, o registo UDDI não possui qualquer funcionalidade que suporte o modelo baseado na distância e, embora a definição das taxonomias espaciais possa representar implicitamente relações entre as localizações, não há qualquer utilização da semântica das relações entre as localizações (e.g. uma localização estar contida noutra) para o suporte ao modelo baseado no âmbito. Deste modo, se um cliente não descobrir qualquer serviço para uma dada localização, o UDDI não será capaz de suportar a descoberta de serviços associados a localizações relacionadas espacialmente com a primeira. Quando o registo UDDI recebe uma pesquisa de serviço para uma determinada localização, retorna a referência apenas aos serviços cuja localização condiga exactamente com a indicada na pesquisa.

Observa-se desta análise que as principais limitações do UDDI no suporte à descoberta de serviços baseados na localização se prendem essencialmente com a falta de uma taxonomia especificamente dirigida à categorização de serviços associados a áreas de dimensão inferior à de uma cidade e com a inexistência da modelação do espaço geográfico e da inferência de relações entre localizações

no processamento das pesquisas baseadas na localização. O desafio de investigação que aqui se coloca é o de conseguir suportar os modelos estudados de descoberta de serviços baseados na localização sem comprometer a interoperabilidade fornecida pela norma UDDI. O primeiro passo na resposta a esse desafio consiste na definição de uma meta-taxonomia que permita associar os serviços a espaços de reduzida dimensão. O segundo passo traduz-se na implementação de um modelo de espaço associado ao registo UDDI. No entanto, deve o modelo de espaço estar integrado numa re-implementação do registo UDDI ou ser complementar a este? Na primeira abordagem, os clientes beneficiariam de continuar a usar apenas um só ponto e uma só *interface* para a descoberta de serviços. No entanto, surgiria a questão de como manter intacta a API do UDDI e, simultaneamente, permitir aos clientes a produção de pesquisas baseadas nos modelos de proximidade estudados. Na segunda abordagem, o desenvolvimento de um modelo de espaço que informaria os clientes sobre a melhor forma de produzir as pesquisas ao registo ao UDDI implicaria que a descoberta baseada na localização passasse pela utilização de dois serviços, tornando-se menos eficiente. Além disso, os clientes teriam de ter conhecimento da *interface* e da semântica de utilização do serviço de modelação do espaço, assim como dos procedimentos a tomar na posterior formulação das pesquisas ao registo UDDI. A próxima secção descreve a solução adoptada no âmbito do projecto VADE para o suporte à descoberta baseada na localização utilizando o registo UDDI.

4 Modelo de suporte à descoberta de serviços baseados na localização usando o UDDI

A solução proposta neste trabalho para o suporte à descoberta de serviços baseados na localização utilizando o UDDI consiste então na definição de uma meta-taxonomia para a categorização de serviços associados a espaços de pequena dimensão e na reformulação do modelo de pesquisa ao registo UDDI. A razão da proposta de uma meta-taxonomia resulta do facto desta poder posteriormente ser instanciada em taxonomias específicas de determinados espaços (e.g. taxonomia espacial do *campus* de Azurém). A meta-taxonomia servirá portanto de referencial para o desenvolvimento de taxonomias para espaços de reduzida dimensão. No que diz respeito à reformulação do modelo de pesquisa ao registo UDDI para o tornar adequado à descoberta baseada na localização, a solução proposta passou pela consideração das duas opções identificadas na secção anterior. Como um dos requisitos do sistema a implementar no projecto VADE é a interoperabilidade entre os fornecedores de serviços, a plataforma de suporte à descoberta de serviços e os clientes, é importante manter a utilização da API da norma UDDI, e apenas esta, na solução que aqui é proposta. A decisão por uma das duas opções apresentadas na secção anterior (re-implementação do registo UDDI ou desenvolvimento de modelo de espaço complementar) foi dirigida por este requisito. Apesar da re-implementação do registo UDDI parecer a solução mais adequada, dado que introduz menos alterações no modelo de descoberta, o esforço que estaria associado ao desenvolvimento integral dum registo UDDI

não se coaduna com os objectivos de investigação do projecto VADE. Portanto, a solução passou pelo desenvolvimento de um serviço complementar de modelação do espaço e de suporte à pesquisa ao registo UDDI - o *proxy* UDDI.

4.1 Meta-taxonomia para espaços pequenos

O modelo proposto neste trabalho para associar os serviços à localização é suportado por uma meta-taxonomia para a categorização no registo UDDI de serviços associados a espaços pequenos. A meta-taxonomia é uma proposta de referencial para o desenvolvimento de taxonomias específicas aplicadas a situações reais. Portanto, o conjunto de termos definido nesta meta-taxonomia é uma abstracção, sem aplicação concreta, que orienta a definição de novas taxonomias quanto à estrutura e ao conteúdo. Esta meta-taxonomia parte do ponto onde o âmbito das taxonomias normalizadas de categorização geográfica termina, ou seja áreas inferiores à de uma cidade. O nível mais elevado da taxonomia (ver figura 2) corresponde ao de uma zona de interesse. Este conceito abrange áreas que podem estar incluídas numa cidade (e.g. complexos desportivos, zonas comerciais, *campii* universitários, etc.), mas pode igualmente cobrir áreas rurais onde poderiam igualmente estar disponíveis serviços baseados na localização (e.g. aldeia histórica, complexo megalítico, parque florestal, etc.). As zonas de interesse subdividem-se em espaços abertos e edifícios (espaços fechados). Um espaço aberto consiste em qualquer área ao ar livre (e.g. parques, praças, áreas desportivas ao ar livre, etc.). Os espaços abertos podem incluir equipamentos como, por exemplo, estátuas, WCs, fontanários, cabines telefónicas, terminais ATM, etc.)

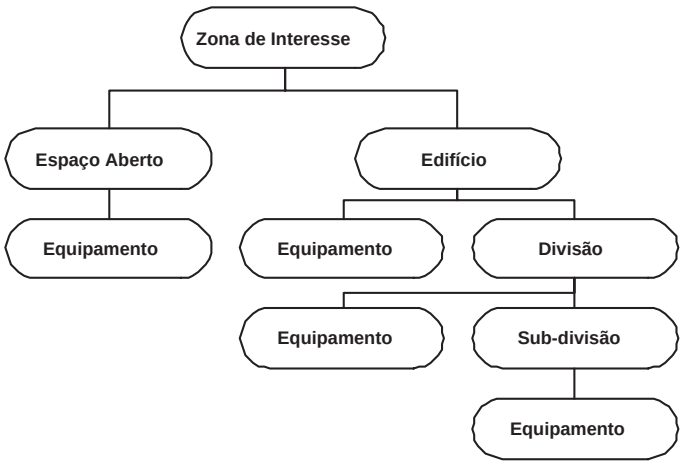


Figura 2. Meta-taxonomia para espaços pequenos

Os edifícios podem ser compostos por divisões (e.g. pisos, blocos) e subdivisões (e.g. salas, garagens, lojas, WCs, etc.). No caso de edifícios com, por

exemplo, um só piso ou um só bloco, não fará sentido estar a criar um elemento artificial na taxonomia, podendo considerar-se como divisões os exemplos dados para sub-divisões. Ou ainda, no caso de edifícios com uma só célula, como uma garagem ou um pavilhão, considerar-se-á apenas o edifício em si. Os edifícios podem conter, tal como os espaços abertos, certos equipamentos, como cabines telefónicas, obras expostas num museu, ou aquários num oceanário, para os quais se pudesse disponibilizar um serviço.

A instanciação desta meta-taxonomia em taxonomias aplicadas a situações concretas resulta numa série de pressupostos:

- as taxonomias são conhecidas tanto pelos fornecedores de serviços (para a categorização do serviço) como pelos clientes (para a pesquisa baseada na localização);
- o registo dos serviços pelas entidades fornecedoras é feito através de uma *interface web*, onde a taxonomia pode estar representada de forma a ser facilmente utilizada;
- no caso dos clientes, pressupõe-se que o mecanismo de localização dos utilizadores se baseie em dados de localização coincidentes com o espaço de nomes da taxonomia;
- o modelo de espaço utilizado para suportar o registo UDDI na descoberta baseada na localização baseia-se na taxonomia definida para aquele espaço.

4.2 *Proxy* UDDI

Tal como foi referido na secção 3, o registo UDDI não fornece por si uma solução adequada para a descoberta de serviços associados a espaços de reduzida dimensão. A definição de uma meta-taxonomia privada que possa ser instanciada para a representação de espaços pequenos contribui para o suporte ao registo de serviços baseados na localização. No entanto, como referido anteriormente, a descoberta de serviços exige mais do que uma simples taxonomia. Existe portanto a necessidade de desenvolver mecanismos que forneçam às aplicações soluções mais ricas para a descoberta de serviços baseados na localização, mantendo a concordância com a norma UDDI. A solução que aqui se propõe consiste num *proxy* do registo UDDI. O *proxy* UDDI é um Web Service que implementa a API UDDI (apenas a componente de pesquisa), fornecendo funcionalidades adicionais às aplicações, e usando na rectaguarda o registo UDDI como fonte de informação. A arquitectura de um sistema de descoberta de serviços baseados na localização em que fosse utilizado o *proxy* UDDI está representada na figura 3. A arquitectura é composta por aplicações (ou clientes), Web Services associados a determinada localização, um registo UDDI e um *proxy* UDDI. Como se pode verificar pela figura, as aplicações podem continuar a usar directamente o registo UDDI, se assim for necessário (e.g. aplicações que não conheçam a semântica do *proxy* ou simplesmente não queiram utilizá-lo).

O *proxy* UDDI baseia a sua funcionalidade num modelo de espaço da realidade servida pelo UDDI. Esse modelo de espaço é compatível com a taxonomia que for definida para o mesmo espaço, e contém a informação sobre todas

as localizações da taxonomia e os relacionamentos existentes entre aquelas. As localizações estão organizadas hierarquicamente, possibilitando a extracção de relações de conteúdo (e.g. a sala A está contida no piso B, ou a zona de interesse C contém o edifício D). Cada localização do modelo de espaço tem um tipo associado (edifício, piso, sala, espaço aberto, equipamento, etc.).

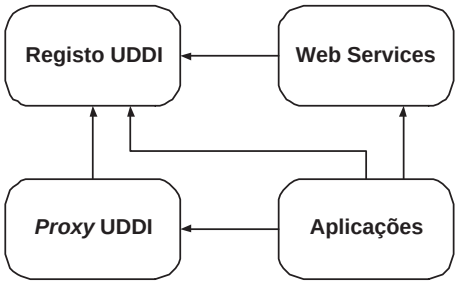


Figura 3. Arquitectura de um sistema de descoberta de serviços baseado num *proxy* UDDI

A funcionalidade básica oferecida por defeito pelo *proxy* UDDI corresponde à interpretação de *queries* UDDI contendo um critério geográfico correspondente à localização do utilizador e, baseado no modelo de espaço, produzir as *queries* necessárias ao registo UDDI, de modo a obter a referência a serviços associados à localização do utilizador ou localizações hierarquicamente superiores. Por exemplo, no caso de uma aplicação que pesquise o *proxy* UDDI para a obtenção de serviços associados a "sala A", o *proxy* vai:

- examinar o modelo de espaço e obter informação sobre as localizações que contenham a localização "sala A"(por exemplo, "piso B", "edifício C"e "Centro Comercial D");
- pesquisar o registo UDDI para serviços associados a "sala A";
- caso não sejam encontrados serviços para aquela localização, pesquisar sucessivamente o registo UDDI para serviços associados às localizações hierarquicamente superiores, até encontrar um serviço;
- responder à aplicação com a informação encontrada no registo UDDI.

Para além da funcionalidade oferecida por defeito, o *proxy* UDDI fornece às aplicações a possibilidade de submeterem *queries* específicas, utilizando valores de parâmetros de pesquisa adicionais. Considerem-se os seguintes exemplos:

- uma *query* para serviços associados a "edifício B"ou a localizações até dois níveis abaixo na hierarquia (pisos e salas contidos em "edifício B");
- uma *query* para serviços associados a "sala A"ou associados a localizações acima na hierarquia, até uma localização do tipo "edifício"(piso e edifício onde se inclui "sala A");

– uma *query* para serviços associados a "sala A"ou a localizações até a uma distância de 100 metros.

De modo a suportar estes parâmetros adicionais mantendo intacta a API de pesquisa do UDDI, a forma como as *queries* são feitas deve ser modificada. A API UDDI fornece mecanismos para a parameterização de *queries* - as já referidas categorizações - através dos quais se informam as condições que os serviços requeridos devem preencher (e.g. estar associados a determinada zona geográfica, ser de determinado tipo de negócio, etc.). Um tipo de categorização já considerado neste trabalho pode ser baseado numa determinada instanciação da meta-taxonomia definida em 4.1. No entanto, para poderem ser especificados parâmetros como os exemplificados acima, são necessários novos tipos de categorizações. Com base no trabalho desenvolvido no projecto AROUND [1], foi definido um conjunto de taxonomias de suporte a pesquisas avançadas no *proxy* UDDI: tipo de localização, distância, tipo de *query*, número de saltos na hierarquia espacial, e regra de seguimento de ligações entre localizações. Algumas destas taxonomias são validadas enquanto que outras não o são. Uma taxonomia validada permite que o *proxy* UDDI verifique a validade dos valores utilizados para aquela taxonomia, evitando a categorização errada de uma *query*. Uma taxonomia não validada permite a utilização de qualquer valor.

A taxonomia de tipo de localização (validada), especificamente orientada a espaços pequenos, é utilizada para informar o *proxy* que uma *query* deve terminar quando for encontrado um serviço associado a uma localização de determinado tipo (considere-se o segundo exemplo de *query* acima enunciado). A taxonomia de tipo de localização é constituída pelos seguintes valores:

Tipo de localização	**Descrição**
`espaco-aberto`	alguma área de um espaço aberto
`espaco-fechado`	tipo genérico para espaços fechados
`edificio`	qualquer tipo de edifício
`divisao-edificio`	tipo genérico de divisão de edifícios
`piso`	piso de edifício
`sala`	sala de edifício
`equipamento`	qualquer tipo de equipamento

Tabela 1 - taxonomia de tipo de localização

A taxonomia de distância é utilizada para categorizar *queries* em que se especifique uma distância máxima à qual um serviço se deve encontrar da localização do utilizador (modelo de proximidade baseado na distância). A funcionalidade oferecida pelo *proxy* neste caso corresponderia a obter do modelo de espaço as localizações que estivessem dentro do limite de distância indicado e pesquisar o registo UDDI para aquelas mesmas localizações. Embora esta taxonomia não seja validada, supõe-se que o *proxy* rejeite valores não numéricos.

A taxonomia de tipo de *query* (validada) especifica apenas três valores: `distancia`, `contido`, e `contendo`. O tipo `distancia` é utilizado para realizar pesquisas de serviços baseada na distância e pressupõe a utilização conjunta da

taxonomia de distância. O tipo de *query* `contido` informa o *proxy* para pesquisar serviços cuja localização esteja hierarquicamente contida na localização do utilizador. O tipo `contendo` é reservado para a descoberta de serviços associados a localizações que contenham a localização do utilizador.

A taxonomia de número de saltos é utilizada para indicar o número máximo de saltos que podem ser efectuados, na busca de serviços, aquando da travessia da hierarquia do modelo espacial. Por exemplo, se uma *query* do tipo `contendo` for categorizada com um máximo de dois saltos, e o utilizador estiver localizado em "sala A", o *proxy* pesquisará apenas serviços associados a localizações até dois níveis acima na hierarquia. Tal como a taxonomia de distância, a taxonomia de número de saltos não é validada, permitindo, contudo, que o *proxy* rejeite valores não numéricos.

Finalmente, a taxonomia de regra de seguimento de ligação é utilizada para informar o *proxy* sobre a forma como as ligações (ou relações) entre as localizações devem ser seguidas, aquando da travessia do modelo de espaço a partir da localização do utilizador. Esta taxonomia validada aceita três valores:

- `soLocal` - as ligações para outras localizações não são seguidas, mantendo a *query* apenas para a localização do utilizador;
- `seNaoLocal` - a regra de seguimento por defeito do *proxy*, na qual as ligações para outras localizações são seguidas apenas no caso de não serem encontrados serviços associados à localização do utilizador;
- `sempre` - não são aplicadas quaisquer restrições ao seguimento das ligações entre localizações).

Quando nenhuma destas taxonomias é utilizada para categorizar uma *query*, o *proxy* assume o comportamento por defeito, que corresponde a uma *query* do tipo `contendo`, com um número de saltos ilimitado, e uma regra de seguimento `seNaoLocal`. Se uma *query* for categorizada com um valor em alguma das taxonomias definidas, o valor utilizado sobrepõe-se ao comportamento por defeito do *proxy*.

A utilização do *proxy* UDDI tem várias vantagens:

- transparência - as aplicações podem usar o *proxy* tal como se estivessem a usar o registo UDDI, beneficiando principalmente as aplicações que não sabem utilizar as taxonomias para a categorização das *queries* baseadas na localização, podendo usufruir da funcionalidade por defeito do *proxy*;
- flexibilidade - as aplicações que sabem como utilizar o *proxy* podem decidir se usam a funcionalidade por defeito, se categorizam a *query* de modo a obter uma funcionalidade específica, ou se nem sequer utilizam o *proxy*.

5 Conclusão

Como se pôde verificar, a utilização do registo UDDI por si só não é o meio mais adequado para o suporte à descoberta de serviços baseados na localização. São necessárias alterações ao modelo de modo a ultrapassar as limitações de que

o registo UDDI carece. As normas taxonómicas utilizadas para a categorização espacial no UDDI não permitem associar serviços a localizações cuja área seja inferior à de uma cidade. Por outro lado, o mecanismo de processamento das pesquisas do UDDI não suporta modelos de proximidade baseados na distância ou no âmbito, como, por exemplo, a pesquisa de serviços que estejam localizados até uma dada distância do utilizador, ou de serviços cujo âmbito de acção inclua a localização do utilizador.

Este artigo apresentou uma proposta que procura resolver os problemas provocados pelas limitações do UDDI quanto à descoberta baseada na localização, mantendo a concordância com a norma UDDI. A primeira componente da solução proposta constitui uma meta-taxonomia para a categorização de serviços associados a espaços de pequena dimensão (inferior a uma cidade). Esta meta-taxonomia pode ser instanciada em taxonomias específicas de ambientes de dimensão reduzida, como, por exemplo, um centro comercial ou um *campus* universitário. A segunda componente da solução corresponde a um *proxy* do registo UDDI cuja funcionalidade é mapear as pesquisas de serviços para determinada localização em uma ou mais pesquisas ao registo UDDI, tendo em conta as relações espaciais existentes a partir da localização do utilizador. O *proxy* UDDI implementa a componente de pesquisa da API UDDI, permitindo manter a interoperabilidade com as aplicações.

Esta abordagem encontra-se actualmente em desenvolvimento no projecto VADE, nomedamente num protótipo onde é feito o uso de um registo UDDI privado, de um determinado número de Web Services e aplicações, e para o qual estão a ser desenvolvidos uma taxonomia de espaço específica do *campus* universitário de Azurém e um *proxy* UDDI (incluindo as respectivas taxonomias de categorização de *queries*).

Referências

1. Grupo de Engenharia da Telecomputação (DSI/UM). Around - supporting location-based internet services. `http://get.dsi.uminho.pt/around`, 2003.
2. Grupo de Engenharia da Telecomputação (DSI/UM). Vade - value added environments for dynamic support to location-based services in umts networks. `http://get.dsi.uminho.pt/vade`, 2003.
3. R. José e N. Davies. Scalable and flexible location-based services for ubiquitous access. In *First Conference on Handheld and Ubiquitous Computing (HUC '99)*, Karlsruhe, Alemanha, Setembro 1999.
4. International Organization for Standardization. Maintenance agency for iso 3166 country codes. `http://www.iso.ch/iso/en/prods-services/iso3166ma/index.html`, 2003.
5. Microsoft. Microsoft uddi business registry node. `http://uddi.microsoft.com`, 2003.
6. OASIS. Universal description, discovery and integration of business for the web. `http://www.uddi.org`, 2003.

Actualização de Dados de GIS usando Assistentes Digitais Pessoais

Henrique Silva[1], João Correia Lopes[2], Alexandre Sousa[1]

[1] Instituto Superior da Maia, Av. Carlos Oliveira Campos, 4475-690 Castelo da Maia.
http://www.ismai.pt/
{hmgs,avs}@ismai.pt

[2] Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto.
http://www.fe.up.pt/
jlopes@fe.up.pt

Resumo Neste trabalho é estendida uma plataforma de visualização de informação geo-referenciada em Assistentes Digitais Pessoais (PDA), Mordomo da empresa ParadigmaXis, por forma a permitir que informação geo-referenciada seja actualizada no local e posteriormente reflectida num Sistema de Informação Geográfica (GIS). Por forma a tratar facilmente diversos formatos de dados de diferentes aplicações, a informação do GIS é transformada para GML (*Geographic Markup Language*), uma instância de XML proposta pela *OpenGIS Consortium* (OGC). A arquitectura proposta foi testada numa aplicação simples, construída dentro do projecto "Percursos Augustinianos", que decorreu no âmbito do "Porto 2001 Cidade Europeia da Cultura".

1 Introdução

Dados geo-referenciados são adquiridos no local e posteriormente são guardados num Sistema de Informação Geográfica (GIS). Mais tarde, é frequente ser necessário alterá-los no GIS, no caso de ser detectada alguma alteração no local. Temos, assim, o registo das alterações no local e posteriormente a alteração no GIS.

Com o advento de Assistentes Digitais Pessoais (PDA), a tarefa de integração de dados no GIS pode ser evitada, uma vez que pode ser efectuada directamente no local, numa representação existente no PDA e posteriormente, através de sincronização, reflectida no GIS.

Neste trabalho é estendida uma plataforma de visualização de informação geo-referenciada num PDA (Mordomo da empresa ParadigmaXis) por forma a permitir a modificação de meta-dados. Estes meta-dados podem representar, por exemplo, informação sobre as características de edifícios, tais como: estado de conservação, cor, etc. Por forma a tratar facilmente diversos formatos de dados de diferentes aplicações, a informação do GIS é transformada para GML (*Geographic Markup Language*) [3], uma instância de XML proposta pela *OpenGIS Consortium* (OGC) [8][9].

Foi projectado e construído um conjunto de módulos aplicacionais, seguindo uma metodologia de programação literária, que permitem passar um mapa representado em GML, mais um conjunto de meta-dados relacionados, para um PDA com o sistema Operativo PalmOS, seguida da manipulação da informação no dispositivo móvel, e trazer essa informação para o ficheiro GML original. Para além do Mordomo, foram utilizadas sobretudo tecnologias ligadas ao XML (*eXtensible Markup Language*) e a linguagem de programação Java.

Na construção e manutenção dos programas desenvolvidos, foi utilizada a ferramenta *Open Source* de suporte à aplicação da metodologia de programação literária, *dotNoweb* [10].

Na Secção 2 é apresentado o standard GML (*Geographic Markup Language*), na Secção 3 — Sincronização de Dados — referem-se os assuntos relevantes sobre a reconciliação de cópias alteradas a partir de uma fonte comum, na Secção 4 — Arquitectura da aplicação — mostram-se os passos que foram dados no sentido de usar o Mordomo para actualizar meta-dados no local e para sincronizar de volta à representação GML a informação alterada no dispositivo móvel, na Secção 5 descrevem-se as opções de implementação e as decisões tomadas com vista à realização dos módulos da arquitectura proposta, na Secção 6 — Exemplo de aplicação — descreve-se a aplicação construída para avaliar a arquitectura proposta e, finalmente, na Secção 7 são apresentadas as conclusões e trabalhos futuros.

2 O *OpenGIS Consortium* e a iniciativa GML

Mapas são representações espaciais de uma dada realidade contendo as suas características relevantes, de acordo com a decisão de um Geógrafo. Os dados geo-referenciados, representando estas características, são adquiridos, analisados e integrados num GIS por forma a poderem ser usados mais tarde. A aquisição destes dados, no local, pode ser conseguida com um PDA equipado com *Global Positioning System* (GPS).

Nos GIS, dados geo-referenciados são representados, principalmente, segundo dois modelos: Matricial e Vectorial [3].

No Modelo Matricial (*raster*) o espaço é quantificado (dividido) num conjunto de pacotes, cada um representando uma quantidade limitada da superfície terrestre; estes pacotes podem ser definidos sobre variadas formas geométricas (quadrados, triângulos, hexágonos) desde que as mesmas possam ser ligadas, isto é, desde que tenham uma aresta comum. Os dados obtidos de satélite, por exemplo, são *rasterizados*, sendo as imagens representadas neste formato, o que leva a que os GIS, vulgarmente, aceitem dados neste modelo, representando-os em pixeis.

O Modelo Vectorial permite indicar explicitamente localizações espaciais, assumindo um espaço contínuo onde os pontos e as posições dos vértices de linhas e polígonos são armazenados sobre um eixo de referência, pela distância à origem (ponto (0,0)) dos eixos. Visto que o Modelo Vectorial permite uma grande compactação dos dados representados, é também suportado pelos GIS.

Os Sistemas de Informação Geográficos permitem ainda manipulação de meta-dados, isto é, permitem associar a localizações geográficas dados relativos a determinados fenómenos e, assim, poderemos ter a manipulação de dados ligados a mapas.

Por forma a permitir a troca de dados entre diferentes GIS, cada um com o seu formato nativo, o *OpenGIS Consortium* desenvolveu uma representação standard, tentando manter as vantagens dos vários modelos de representação. Este standard, de aceitação generalizada, é conhecido por *Geographic Markup Language* (GML) [9] e, por ser baseado em *XML Schema Definition* (XSD) [5], possui tipos complexos e herança de tipos, integração de esquemas XSD distribuídos e *namespaces*. Na Figura 1 é apresentado um exemplo simples de GML.

```
<os:Road>
   <gml:description>Georgia Street</gml:description>
   <os:numberLanes>4</os:numberLanes>
   <gml:centerLineOf>
      <gml:LineString srsName="EPSG:4326">
         <gml:coordinates>0.0,100.0 100.0,0.0</gml:coordinates>
      </gml:LineString>
   </gml:centerLineOf>
</os:Road>
```

Figura 1. Fragmento de GML Representando uma Característica

O OGC propõe o Modelo Abstracto de Características (*OGC Abstract Model*), constituído por um modelo essencial, onde é descrito como o mundo funciona, e por um modelo de especificação, que descreve como o software deve funcionar (através de uma API). O conceito mais importante deste modelo é a Característica (*Feature*), o átomo da informação geográfica, representando uma abstracção de um fenómeno do mundo real (por exemplo, um segmento de estrada entre intersecções consecutivas, uma auto-estrada consistindo em diversos segmentos, uma imagem de satélite geo-referenciada, uma camada de temperaturas num mapa meteorológico).

GML utiliza XLink e Xpointer para exprimir relacionamentos entre entidades geo-referenciadas, as chamadas Características GML. Desta forma, estes relacionamentos podem ser estabelecidos entre *features* da mesma base de dados geográfica ou entre *features* em pontos diferentes da Internet. Para além disso, GML2.0 permite o estabelecimento de relacionamentos entre *features* GML em diferentes bases de dados, sem necessidade de efectuar alterações nas bases de dados participantes. Apenas é requerido acesso de leitura para estabelecer o relacionamento [9].

3 Sincronização de Dados

Por definição, utilizadores de dispositivos móveis não estão permanentemente com ligação à rede e aos seus dados aí guardados, num qualquer sistema anfitrião. Os utilizadores de PDA, retiram do repositório num GIS, uma cópia dos dados do seu interesse para o seu dispositivo móvel, através de uma *conduta* existente na máquina anfitriã com acesso ao GIS. Essa cópia de informação pode ser vista e manipulada no local, através da imediata alteração no PDA. Periodicamente, o utilizador do PDA liga-o à conduta para enviar a informação, entretanto alterada, de volta para o repositório.

Os utilizadores podem ainda ser avisados de alterações efectuadas ao repositório de dados enquanto estiveram desligados e decidir integrar essas alterações na sua cópia local residente no seu dispositivo móvel. Ocasionalmente, terão de resolver conflitos entre alterações efectuadas nos dados, por exemplo no dispositivo móvel e nos dados do repositório, ou em mais do que um dispositivo móvel.

Esta operação de reconciliação, envolvendo trocas de alterações e resolução de conflitos, por forma a tornar equivalentes dois conjuntos de dados, é conhecida como *sincronização de dados*.

Aplicações para PDAs frequentemente geram ou modificam dados partilhados com um servidor ou com outros dispositivos móveis. Já que o dispositivo está desligado da rede a maior parte do tempo, ou então está ligado através de uma ligação demasiado cara ou lenta para ser usada todas as vezes que os dados são modificados, aplicações móveis tipicamente mantêm cópias separadas dos dados partilhados em diferentes dispositivos. Estas cópias são sincronizadas, de tempos a tempos, de tal forma que todas elas contenham os mesmos dados.

O problema, complexo, que aqui tem de ser tratado, é o de reconciliar dados provenientes de várias cópias, com diversas modificações. Uma vez que a representação está em XML, aquilo que é necessário é encontrar diferenças entre árvores XML.

Na especificação *Mobile Network Computing Reference Specification* (MNCRS) [2] é apresentada uma API (*Application Programming Interface*) que fornece serviços de sincronização de dados. Estes serviços simplificam muito a escrita de aplicações móveis.

Para encontrar as diferenças entre documentos XML têm sido propostas várias técnicas de comparação: X-Diff [11], HtmlDiff e TopBlend (baseado em HtmlDiff) [1]. Estas ferramentas efectuam uma comparação de XML *cega* não fazendo qualquer análise semântica. Desta forma, estas aproximações detectam pequenas diferenças entre dados XML que não serão reflectidas no mapa final e poderiam, por isso, ser ignoradas.

4 Arquitectura da aplicação

Para atingir os objectivos enunciados, foi definida e prototipada uma arquitectura que permite colocar num PDA um mapa e meta-informação sobre esse

mapa, alterar essa informação e integrar as alterações, de volta, no repositório em GML.

Primeiramente é necessário seleccionar um dispositivo móvel, do tipo PDA, onde serão colocados os dados geo-referenciados, que poderão ser alterados no local. As três principais plataformas, dispositivos com PalmOS, dispositivos PocketPC e com sistema operativo Psion, diferem em custo, popularidade, funcionalidades, ergonomia mas, no que diz respeito à arquitectura proposta, não introduzem diferenças significativas. Assim, por razões práticas, dada existência de uma aplicação que poderia ser usada para mostrar o mapa de base, foi escolhida a plataforma PalmOS.

Para efeitos de desenvolvimento e testes será usado um *Palm OS Emulator* (POSE), que permite igualmente testar a viabilidade da conduta.

Por forma a permitir a passar documentos GML para um dispositivo móvel, a arquitectura ilustrada na Figura 2, inclui um conjunto de módulos designados por "Aplicações MetaGeo", que passaremos a descrever.

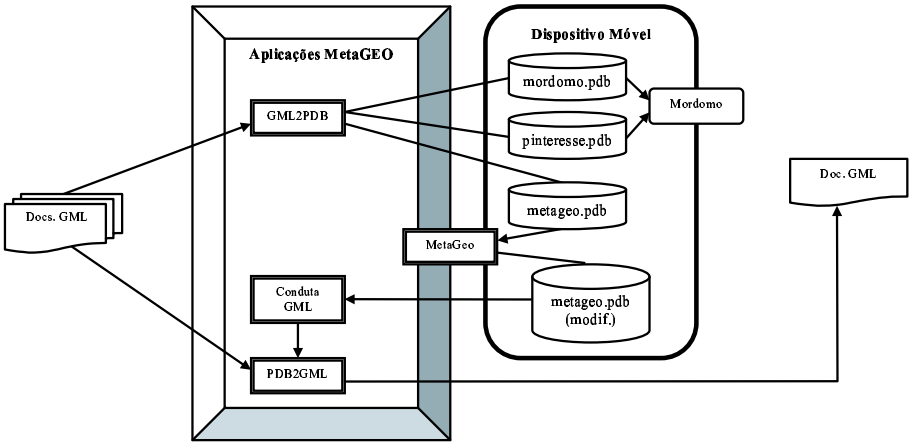


Figura 2. Arquitectura para a Aplicação

A arquitectura proposta inclui um módulo (GML2PDB) que separa a informação a enviar para o PDA em três categorias, cada uma colocada numa base de dados do PDA (do tipo PDB, com extensão `.pdb` no PalmOS):

`mordomo.pdb` só de leitura, contendo a informação cartográfica para o Mordomo, descrevendo um mapa;

`interestPoints.pdb` só de leitura, contendo a informação sobre a localização dos pontos de interesse para a aplicação;

`metageo.pdb` de leitura e escrita no PDA, contendo meta-informação associada aos pontos de interesse, isto é, os atributos desses pontos de interesse (por

exemplo, informação sobre as características de edifícios tais como: estado de conservação, cor, etc).

O módulo MetaGeo da arquitectura, desenvolvido neste trabalho, apresenta a informação no PDA e permite efectuar alterações na base de dados `metageo.pdb` do PalmOS.

Em cada pedido de sincronização, a conduta, GMLConduit, chama o componente PDB2GML que converte a base de dados `metageo.pdb` para uma árvore DOM [4]. Esta árvore é depois comparada com a árvore DOM que resulta do GML original.

Para passar os dados da sua representação em GML para uma representação no formato PDB do PalmOS, foi desenvolvido um modelo de dados, baseado no Modelo Relacional, que pode ser manipulado através de uma interface semelhante ao *DAO Object Model* (DAO) [6]. Este modelo representa em *RecordSets* colecções em memória, contendo dados de uma tabela ou vista.

5 Implementação da aplicação

O módulo GML2PDB percorre um ou mais ficheiros GML por forma a colocar no dispositivo móvel, o mapa e a meta-informação que deve será manipulada no local, por forma a introduzir eventuais alterações.

No princípio tentou guardar-se o GML no PalmOS, usando um algoritmo de compressão por forma a lidar com os recursos exíguos existentes, relativos à ligação ao dispositivo e à memória que, tipicamente, estes dispositivos apresentam.

O algoritmo XMill [7] é reconhecido por apresentar bons resultados de compressão de dados XML, no entanto não se mostrou nada conveniente para este cenário de utilização.

O primeiro problema reside no facto de este algoritmo tratar todo o GML de uma só vez e por isso, quando usado na fase de descompressão no dispositivo, para um conjunto de dados representativo, não existir memória suficiente no dispositivo. Não é possível trabalhar apenas com os blocos visíveis pois, apesar de o algoritmo usar blocos de tamanho fixo (parametrizável) na fase de descompressão eles ficam com tamanhos variáveis, dependendo da taxa de compressão conseguida para cada um. Outro problema inerente à utilização deste algoritmo para esta aplicação, prende-se com o facto de ele não entender a semântica dos dados e, por isso, não colocar dados próximos geograficamente no mesmo bloco, na altura da descompressão. Não fossem estes dois problemas e seria possível trabalhar apenas com o quadrado visível e com os oito quadrados adjacentes descomprimidos [12].

Tal como é ilustrado na Figura 3, o quadrado mais escuro representa o fragmento do mapa em visualização no ecrã do dispositivo móvel e os quadrados cinzentos representam dados já descomprimidos, por forma a minimizar o deslocamento do quadrado visível do mapa. Os quadrados seriam trazidos e retirados de memória à medida das necessidades e de acordo com as movimentações requeridas pelo utilizador.

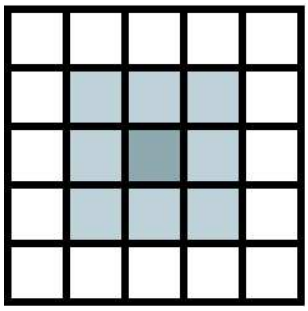


Figura 3. Compressão de um Mapa

Para tentar resolver os problemas de falta de memória no dispositivo móvel, experimentou-se com fragmentos de compressão mais pequenos mas isso tornou o algoritmo menos eficiente. Também foi tentada uma verificação (*parsing*) prévia do GML por forma a colocar próximos dados geograficamente próximos, mas os resultados também não foram satisfatórios. Para além destes problemas, de cada vez que fosse efectuada uma alteração, o bloco respectivo teria de ser comprimido novamente assim que saísse de memória, o que baixaria a eficiência.

Por todas estas razões, decidiu-se usar o GML apenas como meio de obter e fornecer dados de formatos nativos garantindo assim a interoperabilidade entre a aplicação no dispositivo móvel e os dados residentes em repositórios GIS.

Para obter uma árvore DOM correspondente a um documento GML, foi utilizado o pacote GML4Java desenvolvido por *Galdos Inc.* As classes fornecidas suportam bem o Modelo Abstracto OGC, especialmente as Características GML. Usando esta ferramenta criou-se um *parser* capaz de extrair as bases de dados PDB para o PalmOS a partir de ficheiros GML, atravessando os nós da árvore DOM e, de acordo com o tipo de característica, transformar os dados para o formato esperado pelo PalmOS.

Por forma a permitir a manipulação dos dados geo-referenciados no PDA, construiu-se a aplicação MetaGeo, que apresenta a interface com o utilizador ilustrada na Figura 4.

Para colocar os meta-dados alterados no dispositivo móvel de volta ao repositório no ficheiro GML original (correspondente ao repositório no GIS), podem ser consideradas duas possibilidades:

modificações apenas no Palm caso mais simples em que é apenas necessário comparar as árvores DOM correspondentes à cópia de partida no ficheiro GML e à cópia modificada no dispositivo móvel e reflectir, no ficheiro GML, as modificações encontradas;

modificações nas duas cópias neste caso é necessário comparar a árvore DOM correspondente ao ficheiro GML original, com a árvore DOM correspondente ao ficheiro GML modificado e ainda com a árvore DOM correspondente ao

Figura 4. MetaGeo em uso num PDA

ficheiro PDB com as alterações efectuadas no dispositivo móvel e realizar uma sincronização de dados.

Para detectar as diferenças entre as árvores DOM pode ser usado um dos métodos referidos na secção 3 [11] [1].

Para o protótipo desenvolvido, considerou-se apenas alterações nos metadados residentes no dispositivo móvel, situação que se espera seja a mais frequente, e, por isso, basta eliminar, modificar ou inserir nós na árvore DOM representando o ficheiro GML pretendido, usando o algoritmo apresentado em pseudo-código na Figura 5.

No caso de existirem cópias em alteração simultânea em diferentes dispositivos móveis, é necessário usar a API (MNCRS) [2] e no caso de existirem alterações conflituosas deve ser envolvido o utilizador, que será responsabilizado por escolher qual a cópia a manter.

6 Exploração e avaliação

Para avaliar a arquitectura proposta, foi construída uma aplicação simples. Foram marcados num mapa da cidade do Porto existente no Mordomo, os pontos de interesse do projecto "Percursos Augustinianos", que decorreu no âmbito do "Porto 2001", relacionado com os locais da cidade referidos nos livros da escritora Augustina Bessa Luís.

Nesta aplicação prática, não se partiu de um mapa da cidade do Porto representado em GML, que não foi possível encontrar, usou-se um mapa já em

```
ModifiedTree <- NEW DOMTree
DeletedTree <- NEW DOMTree

FOR EACH Object IN DataBase
   ObjectNode <- NOTHING
   IF (Object.New OR Object.Modified) THEN
      ObjectNode <- ADD( ModifiedTree, Object, NOTHING, True )
   ELSE IF (Object.Deleted) THEN
      ObjectNode <- ADD( DeletedTree, Object, NOTHING, True )
   END IF
   FOR EACH Property IN Object
      IF (Property.New OR Property.Modified)) THEN
         IF (ObjectNode IS NOTHING) THEN
            ObjectNode <- ADD( ModifiedTree, Object, NOTHING, True )
         END IF
         ADD( ModifiedTree, ObjectNode,
              Property, True )
      ELSE IF (Property.Deleted) THEN
         IF (ObjectNode IS NOTHING) THEN
            ObjectNode <- ADD( DeletedTree, Object, NOTHING, True )
         END IF
         ADD( DeletedTree, ObjectNode, Property, True )
      END IF
   END FOR
END FOR

GMLTree <- NEW GML4Java.Document

# First DELETE Nodes

FOR EACH Node IN DeletedTree

   deleteNode <- FIND( GMLTree, Node )

   IF NOT (deletedNode IS NOTHING) THEN
      GMLTree.Delete( Node )
   END IF
END FOR

# Next ADD / UPDATE Nodes
FOR EACH Node IN ModifiedTree
   changedNode <- FIND( GMLTree, Node )
   IF changedNode IS NOTHING THEN
      GMLTree.Add( Node )
   ELSE
      changedNode <- Node
   END IF
END FOR

Serialize( GMLTree )
```

Figura 5. Algoritmo de Sincronização

PDB do PalmOS, que não era suposto ser alterado no dispositivo móvel. Esse mapa em PDB foi convertido, usando o PDB2GML, para uma representação em GML. Os pontos de interesse, seleccionados sobre o mapa da cidade, foram adicionados ao ficheiro GML, que seria transformado no formato PDB pelo módulo GML2PDB.

Usando a aplicação MetaGeo executando num dispositivo móvel (Palm) equipado com GPS foram feitas alterações na meta-informação relacionada com os pontos de interesse e posteriormente, após sincronização, a informação foi colocada de volta no ficheiro GML.

Foi verificado, através da leitura de jornais (*logs*), que todas as modificações nos meta-dados foram devidamente tratadas. Em termos de desempenho, o processo decorreu com excessiva lentidão, que foi atribuída ao algoritmo de comparação entre a árvore DOM gerada a partir do ficheiros PDB vindo do dispositivo móvel e à árvore correspondente ao ficheiro GML original.

Para validar o ficheiro GML gerado pela sincronização, incorporando já as alterações, foi utilizado o *parser* Xerces de XML, que não detectou qualquer problema. O *parser* GML4Java, específico de GML, continuou a considerar o documento GML válido.

Finalmente foi verificado o conteúdo do ficheiro GML num navegador com suporte para XML, o *Internet Explorer*, que também não indicou problemas.

Ficou, assim, provado que a arquitectura resolve o problema de alterar dados apenas no local, evitando a posterior alteração dos dados no GIS e os eventuais erros que podem ocorrer nessa altura.

7 Conclusões e trabalho futuro

Provada a viabilidade da arquitectura, é altura de facilitar o seu uso através da construção de uma biblioteca que pode ser chamada por outras aplicações, para além do Mordomo. Para isso vai ser identificada uma API, que através de parametrização, possa ser configurada para variadas plataformas existentes em PDAs, nomeadamente no PocketPC e no Psion, para além de dispositivos com PalmOS.

Deverá ainda ser tratado o problema de sincronizar diversas cópias da mesma informação, por forma a tornar a arquitectura útil, na prática.

Referências

1. Yih-Farn Chen, Fred Douglis, Huale Huang, and Kiem-Phong Vo. TopBlend: An efficient implementation of HtmlDiff in Java. Technical Report 00.5.1, AT&T Labs, January 2000.
2. Norman H. Cohen. Design and Implementation of the MNCRS Java Framework for Mobile Data Synchronization. Technical Report RC 21774, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, November 2000.
3. Simon Cox, Adrian Cuthbert, Ron Lake, and Richard Martell. Geography Markup Language (GML) 2.0 — OGC Recommendation, February 2001. OGC Project Document 01-029.

4. W3C Architecture Domain. Document Object Model (DOM). W3C Technical Reports, June 2002. http://www.w3.org/DOM/.
5. David C. Fallside (Editor). XML Schema Part 0: Primer. W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-0/.
6. Helen Feddema. *DAO Object Model: The Definitive Reference*. O'Reilly & Associates, Inc., January 2000.
7. Hartmut Liefke and Dan Suciu. XMill: an efficient compressor for XML data. In Weidong Chen, Jeff Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD on Management of Data, Dallas, Texas, 16–18 May, 2000*, pages 153–164. ACM Press, June 2000.
8. OGC. OpenGIS Geography Markup Language Specification. OGC Request 11: A request for Comments, December 1999.
9. OGC. Geography Markup Language (GML) 2.0. OGC Recommendation Paper, February 2001. http://www.opengis.org/.
10. Alexandre Valente Sousa. Literate programming applied to software maintenance and teaching computer science. *Perspectivas XXI*, 4(8):101–120, November 2001.
11. Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: An effective change detection algorithm for XML documents. In *ICDE 2003 International Conference on Data Engineering, India*, March 2003.
12. JieBing Yu and David J. DeWitt. Processing satellite images on tertiary storage: A study of the impact of tile size on performance. In *Proceedings of the 1996 NASA Conference on Mass Storage Systems, College Park, Maryland*, September 1996.

Engenharia reversa de HTML usando tecnologia XML

José João Almeida and Alberto M. Simões

Projecto Natura
Departamento de Informática
Universidade do Minho
{jj@|albie@alfarrabio.}di.uminho.pt
http://natura.di.uminho.pt

Resumo O proliferar de ferramentas criadores de HTML e o uso de HTML guiado pelo aspecto, tem vindo a arruinar o seu lado conceptual. Este problema foi reconhecido e deu origem a vários formatos ou tecnologias com o objectivo de separar o aspecto do conceito.
No entanto a realidade actual mostra uma enorme quantidade de páginas HTML com péssima leitura conceptual e estrutural, invalidando uma série de usos possíveis da informação nelas contida.
Nesta comunicação apresenta-se um trabalho (em fase inicial) que pretende fazer engenharia reversa de HTML para permitir aumentar a sua acessibilidade, a fim de ser usada num *browser* para invisuais.

1 Introdução

O uso abusivo de HTML para estruturar graficamente a informação tem tornado grande parte das páginas ilegível mesmo para pessoas sem deficiência visual. Embora o *World Wide Web Consortium* e outras entidades tenham apresentado várias tecnologias para separar o lado conceptual do lado gráfico das páginas de Internet (HTML[4] com CSS[3] ou XML[1] com XSL[2]), o seu uso é ainda bastante reduzido.

Em particular, os portais, os jornais e páginas de informação são das que mais saturaram o aspecto visual descuidando completamente o lado conceptual. No entanto, estas páginas são as que mais interessam ao público em geral (e em particular aos invisuais).

O nosso objectivo principal é construir um conjunto de ferramentas para simplificar o código HTML.

Em vez de construir um ferramenta simplificadora de HTML, optou-se por alargar o domínio de uso de uma ferramenta de processamento de XML (o XML::DT), levando a que o processamento estrutural de XML que ela permite possa ser aplicável ao HTML. Deste modo a criação de ferramentas de simplificação e processamento de HTML, passa a ser programar XML::DT, permitindo uma escrita compacta e usufruindo de uma experiência e de uma cultura já existente.

Neste documento, para além de exemplos ligados a acessibilidade, serão ainda apresentados alguns exemplos de manipulação geral de HTML, sendo nossa intenção demonstrar que é simples a escrita destes processadores.

As ferramentas aqui apresentadas foram construídas usando um módulo Perl de processamento de documentos XML denominado de XML::DT[5]. Este módulo funciona sobre a biblioteca `libxml2`, equipada com fuzzy-parsing, pelo que é capaz de processar HTML com erros.

2 Introdução ao `XML::DT`

Este documento vai introduzir o uso do módulo apresentando exemplos progressivamente mais complicados. Todos estes exemplos serão de processamento e limpeza de documentos HTML. Embora a compreensão total de determinados exemplos obrigue a conhecimentos da linguagem Perl, é perfeitamente possível entender a mensagem deste artigo através duma compreensão parcial.

Por uma questão de simplificação apenas usaremos as seguintes funções do módulo `XML::DT`:

- `dt` – (down-translate) dado um ficheiro (XML ou HTML) e um processador, faz a travessia e processamento estrutural.
- `dturl` – idem a partir do url dum documento
- `toxml` – gera XML.

e das variáveis globais:

- `$q` – nome da etiqueta do elemento actualmente em processamento
- `$c` – conteúdo do elemento actualmente em processamento
- `%v` – array associativo dos atributos do elemento actualmente em processamento

2.1 Remoção de etiquetas

No primeiro exemplo faz-se a remoção das etiquetas `script` e a substituição das etiquetas de imagem pelo seu atributo `alt`. Este atributo, frequentemente ignorado, é fulcral para permitir a legibilidade a invisuais.

```
1 #!/usr/bin/perl
2  use XML::DT;

3  %h=( -html      => 1,
4       -outputenc => "ISO-8859-1",
5       img        => sub{ $v{alt} || "" },
6       script     => sub{ "" } );

7  print dt(shift,%h);
```

A primeira linha, comum a todas as *scripts* (de qualquer linguagem), indica qual o interpretador da linguagem a ser usado. No nosso caso, o Perl. A linha seguinte importa o módulo que vamos usar[1].

As quatro linhas seguintes configuram o processador do documento XML. A primeira opção, denominada por `-html` instrui o módulo de que deve utilizar o fuzzy-parsing para ler documentos HTML mal formados. De seguida indicamos qual a codificação a utilizar na apresentação dos resultados. Segue-se a definição de duas funções para processar as etiquetas em questão (`img` e `script`. No caso desta última, associamos-lhe uma função que retorna a *string* vazia (pelo que esta etiqueta e seu conteúdo é removido). No caso da etiqueta `img`, retornamos o seu atributo `alt`, acessível a partir do *array* associativo `v` ou, caso não exista, a *string* vazia.

A última linha imprime o resultado de processar o ficheiro passado como argumento.

Como se observa, o facto de se poder definir transformações locais (associadas a um contexto expecífico) simplifica bastante a acção de transformar HTML. Às etiquetas que não associamos uma função de processamento é associada automaticamente a função identidade.

2.2 Remoção de atributos e etiquetas

Uma das coisas que tornam o HTML complexo é a utilização de uma série infindável de atributos visuais.

No exemplo seguinte vão ser removidos atributos que constem de uma lista de atributos *ignoráveis*.

```
1  my @ignore = qw/color width height/;

2  %h = ( -html => 1,
3         -outputenc => 'ISO-8859-1',
4         -default => sub{
5            for (@ignore) { delete($v{$_}) }
6            toxml
7         });

8  print dt(shift,%h);
```

Este exemplo mostra o uso da regra `-default` que é executada para todas as etiquetas que não têm função de processamento especificada. Mostra também a função `toxml` que de acordo com o nome da etiqueta em causa `$q`, do *array* associativo de atributos `%v` e do conteúdo da etiqueta `$c`, reconstrói o texto XML. Note-se que depois de apagar determinados atributos estes deixam de existir e portanto não aparecem na reconstrução do texto.

Embora funcione, este exemplo é bastante simplicista. Na verdade, o que vamos querer é remover determinadas etiquetas (como as `script`), determinados atributos (como o `color`) e alguns atributos para determinados elementos.

[1] nos exemplos seguintes omitiremos essas linhas para simplificar.

```
1  my %ignore_tag = (script => 1);
2  my %ignore_atts => ( -default => ['color','width','height'],
3                       a => ['onmouseover', 'onmouseclick'] );

4  %h = ( -html => 1,
5         -outputenc => 'ISO-8859-1',
6         -default => sub{
7            my @atts_to_remove = ();

8            return "" if $ignore_tag{$q};

9            if (exists($ignore_atts{$q}))
10            { @atts_to_remove = @{$ignore_atts{$q} }
11           else
12            { @atts_to_remove = @{$ignore_atts{-default}} }

13           for (@atts_to_remove) { delete($v{$_}) }
14           toxml
15        });

16 print dt(shift,%h);
```

Descrição do programa:

linha 1: definimos um *array* associativo em que as chaves são as etiquetas a remover — usar um *array* associativo em vez de um simples *array* torna a consulta mais rápida;

linha 2-3: definimos também um *array* que associa a cada etiqueta uma lista de atributos a remover. A chave `-default` corresponde aos atributos a remover por omissão;

linha 7: na primeira linha da função de processamento de todas as etiquetas definimos um *array* local dos atributos a remover;

linha 8: se a etiqueta em causa (acessível na variável `$q`) é para remover, retornar a *string* vazia;

linha 9-12: verificar se existe uma regra específica de atributos a remover para a etiqueta em causa. Se não existir, usar a regra por omissão;

linha 13-14: semelhante ao exemplo anterior, apaga os atributos em causa e retorna o texto XML correspondente;

2.3 Tratamento de `frames` e `iframes`

A tecnologia de *frames* é das que levantam complicações em relação à forma como devem ser tratadas para que um invisual consiga perceber minimamente o conteúdo do *site* em questão.

Nesta subsecção vamos apresentar um exemplo (simplicista) que se baseia em:

– todas as *frames* vão ser concatenadas da esquerda para a direita e de cima para baixo;

– as *iframes* vão ser incluídas directamente no local onde aparecem no documento;
– estas inclusões devem ser processadas recursivamente.

```
1  use LWP::Simple;

2  %h = ( -html => 1,
3         -outputenc => 'ISO-8859-1',
4         frameset => sub { "$c" },
5         frame  => sub { dturl($v{src}, %h) },
6         iframe => sub { dturl($v{src}, %h) },
7         noframes => sub { "" } );

8  print dt(shift,%h);
```

Descrição do programa:

linha 1: precisamos de um novo módulo Perl para fazer download de páginas;
linha 4: as etiquetas `frameset` desaparecem, mantendo o seu conteúdo;
linha 5,6: para cada um dos tipos de `frames`, chamar uma função que processe o URL do atributo `src`;
linha 7: por fim, remover a etiqueta `noframes`, desnecessária para o exemplo em questão, visto que iria introduzir redundância ao documento (e também porque nunca é usada da forma correcta);

2.4 Extracção de sub-tabelas

Em muitas páginas que são constituídas por tabelas, há utilidade em extrair partes, por exemplo em extrair o corpo central, ou uma sub-tabela que seja o índice.

No próximo exemplo apresenta-se um programa que quando usado sem argumentos constrói uma página HTML com a lista de todas as tabelas existentes (e respectivos identificadores); quando invocado com um um identificador de tabela, constrói um página HTML contendo apenas essa tabela.

Os identificadores de tabela usados estão a ser constituídos por um numeração composta de acordo com a hierarquia das tabelas. Exemplo: 1.1.3 – 3ª tabela contida na 1ª tabela da 1ª tabela. Este tipo de identificador é mais estável às mudanças locais.

```
1  my $file= shift or die("usage tabela file.html [tableid]\n");
2  my $tn=shift;

3  %h1=(-html => 1,
4       -outputenc => 'ISO-8859-1',
5       -begin => sub{ print "<html><body>" },
6       -end   => sub{ print "</body></html>" },
7       table  => sub{ print "\n<h1>Tab. ",ntb()."</h1>\n",toxml();
8                      toxml() });
```

```
9  %h2=(-html => 1,
10      -outputenc => 'ISO-8859-1',
11      table => sub{
12        if($tn eq ntb()){
13            print "<html><boby>",toxml(),"</body></html>"; exit; }
14        else {toxml()} });

15 if($tn){ dt($file,%h2);}  ## extrai a tabela $tn
16 else   { dt($file,%h1);}  ## constrói uma pág. com a lista das tabelas

17 sub ntb{
18    my $l=$dtcontextcount{table};
19    for (0..$l-1){$na[$_] ||= 1 ;  }
20    @na = (@na[0..$l-2], $na[$l-1]+1);
21    join(".", (@na[(0..$l-2)],$na[$l-1]-1));
22 }
```

Descrição do programa:

linha 2-7: Definição do processador para a situação em que não é dado o identificador de tabela pretendido.
linha 4: Código executado no ínicio do processamento.
linha 5: Código executado no fim do processamento.
linha 6-7: função de processamento de tabela: imprimir um cabeçalho com o identificador da tabela seguido da tabela (efeito lateral) e devolve o texto da tabela.
linha 8-13: Processador para a situação em que é dado o identificador de tabela a extrair.
linha 10-13: função de processamento de tabela: se o identificador de tabela é o pretendido, imprime a tabela (efeito lateral) e sai.
linha 14-15: Conforme há ou não identificador de tabela, invoca o processador pretendido.
linha 16-23: Ignorar! (Define a função que calcula o identificar da tabela corrente.

Esta script pode ser usada do seguinte modo

```
1  tabelas publico.html > listaDeTab.html
```

para a construção da lista das tabelas encontradas, permitindo observar qual ou quais as que mais interessem ao fim em vista. Para extrair uma tabela estecífica, usaremos:

```
1  tabelas publico.html 1.3 > tabela.html
```

3 `XML::DT` com XPath

Depois de muito se ter usado o módulo `XML::DT` verificou-se que algum do código escrito em muitos dos processadores construídos podiam ser simplificados com a utilização do XPath. Com esse objectivo foi desenvolvida uma função de processamento específica para aceitar (um subconjunto de) predicados XPath.

Esta nova função (`pathdt`) faz uma análise dos predicados em questão e gera o código necessário para que a função genérica do `XML::DT` realize as transformações especificadas.

3.1 Criação de índices

À semelhança do que acontece com quem vê, a leitura de uma página por parte de um invisual não é normalmente uma leitura de sequencial. A página deve ser trabalhada e seccionada, associando um título a cada uma destas secções. Estes títulos é que serão lidos sequencialmente, permitindo ao ouvinte escolher que secções quer ler.

O exemplo sequinte tem como objectivo processar um documento HTML e colocar o seu índice no seu início;

```
1  my @index = ();
2  my $i=0;
3  %h = ( -html => 1,
4         -outputenc => 'ISO-8859-1',
5         'h1|h2|h3' => sub{
6             $index[$i] = toxml;
7             $c = toxml("a",{name=>"a$i"},$c);
8             $i++;
9             toxml; },
10        body => sub{
11            my $index = "";
12            my $j = 0;
13            for (@index) {
14                $index.="<a href='a$j'>$_</a>"; $j++
15            }
16            $c = "$index $c";
17            toxml;
18        }
19 );

20 print pathdt(shift, %h);
```

Embora este exemplo seja um pouco *naïve*, demonstra a utilidade do XPath para determinadas situações, bem como a importância de ferramentas deste género.

Descrição do programa:

linha 1,2: inicializar uma lista com as entradas do índice, e inicializar o contador de entradas;
linha 5: utilizar um predicado XPath para associar uma função às etiquetas `h1`, `h2` e `h3`;
linha 6: guardar no índice o título (e respectiva etiqueta de *heading*) na lista de entradas do índice;
linha 7-9 : colocar uma âncora de destino no título;
linha 11-16: construir o índice e colocá-lo no início do documento;

3.2 Secções por `font`

É habitual nos tempos que correm, determinadas ferramentas geradoras de HTML (como o *Microsoft Word*) produzirem HTML em que os títulos não são nada mais do que uma etiqueta `font` a definir determinado tamanho.

Podemos tentar fazer engenharia reversa com o seguinte código:

```
%h = ( -html => 1,
       -outputenc => 'ISO-8859-1',
       "//font[@size&gt;'4']" => sub { $q = 'h1'; toxml },
       "//font[@size='4']"    => sub { $q = 'h2'; toxml },
       "//font[@size='3']"    => sub { $q = 'h3'; toxml },
       "//font[@size='2']"    => sub { $c               },
       "//font[@size='1']"    => sub { $q = 'small'; toxml },
);

print dt(shift, %h);
```

Este código, fortemente baseado em predicados XPath, associa funções diferentes às etiquetas `font` dependendo do valor do seu atributo `size` (que segundo a especificação, varia entra 1 e 7). Embora em Perl não fosse necessário utilizar a entidade ">" para representar o sinal, optou-se por manter coerência com a especificação do XPath.

4 `XML::DT` Tipado

O `XML::DT` pode associar tipos elementos XML, permitindo que seja automaticamente construída uma representação perl de ficheiro XML (ou de partes dele. Considere-se o seguinte documento XML:

```
<a>
  <b>1</b>
  <b>2</b>
  <b>3</b>
  <b>4</b>
</a>
```

Por omissão a função de processamento de cada elemento recebe uma string com o resultado de processar os filhos. Se associarmos o tipo `SEQ` ao elemento "a", a função que o processa passa a receber a lista `[1,2,3,4]`[2] como argumento (em `$c`).

Os tipos suportados de base pelo `XML::DT` são:

STR é o tipo por omissão e concatena todos os valores retornados pelos seu sub-elementos, o que implica que os seus sub-elementos devem retornar uma string;

SEQ constrói uma lista (array) com o conteúdo dos sub-elementos. Isto significa que os atributos são ignorados (ou devem ser processados no sub-elemento). Retorna a referência para a lista.

SEQH constrói uma lista de arrays associativos com todos os seus sub-elementos. Cada elemento da lista é um array associativo que associa:

[2] Em perl esta lista é uma referência.

-q ao nome do elemento;
-c ao seu conteúdo;
at1 ao valor do atributo `at1` (para cada atributo existente);

MAP cria um array associativo com os sub-elementos; as chaves são os nomes dos sub-elementos e os valores os seus conteúdos. Os atributos são ignorados (devem ser processados nos sub-elementos);

MULTIMAP cria um array associativo de listas; as chaves são os nomes dos sub-elementos e os valores são listas com os conteúdos dos vários elementos com o mesmo nome. Os atributos são ignorados.

MMAPON é escrito como `MMAPON(lista-de-elementos)`, e cria um array associativo com os sub-elementos. As chaves são os nomes dos sub-elementos e os valores os seus conteúdos. Os atributos são ignorados. Para todos os elementos contidos na lista de elementos, é criado uma lista com os seus conteúdos;

XML retorna uma referência para um array associativo com:
-q ao nome do elemento;
-c ao seu conteúdo;
at1 ao valor do atributo `at1` (para cada atributo existente);

ZERO não processa os sub-elementos. Retorna ;

4.1 HTML com design por Tabelas

Nas páginas HTML aparece frequentemento o uso de tabelas para compor gráficamente pedaços HTML. Esse sistema complica o seu uso em ambientes não gráficos.

No próximo exemplo cada tabela está a ser analisada e transformado de acordo como as suas dimensões, seguindo a eurística que a seguir se descreve:

- tabelas 1×1 — dão origem a um título (`h1`);
- tabelas $1 \times n$ — dão origem a uma lista (`ul`);
- tabelas $2 \times n$ — dão origem a uma lista descritiva (`dl`) se a opção `mapping_as_dl` tiver sido selecionada;
- tabelas $n \times n$ — são mantidas como tabelas;

Para que o cálculo das dimensões da tabela fosse fácil, estamos a associar tipos a algumas etiquetas:

- cada `table` vê os seus filhos como uma sequência (de `tr`)
- cada `tr` vê os seus filhos como uma sequência (de `td`)

Deste modo o tipo associado é estruturado e pode ser facilmente usado para determinar o número de linhas e colunas.

```
1 use CGI qw(:all) ;

2 our ($mapping_as_dl);

3 my $filename = shift;
```

```
4  %h=( -html => 1,
5       -outputenc => 'ISO-8859-1',
6       -type  => { table => "SEQH",
7                   tr    => "SEQH",
8                 },
9       tr     => sub{ $c},
10      td     => sub{ $c},
11      table  => sub{
12         my ($li,$co)=dimTabela($c);
13         print STDERR "Debug Tabela($li,$co)";
14         if($li == 1 && $co == 1){ h1($c->[0]{-c}[0]{-c}) }
15         elsif($co == 1)         { ul(li([map {$_->{-c}[0]{-c}} @$c])) }
16         elsif($li == 1)         { ol(li([map {$_->{-c}} @{$c->[0]{-c}}])) }
17         else                    {
18           if($co==2 && $mapping_as_dl){
19               dl(map {CGI::dt($_->{-c}[0]{-c}).dd($_->{-c}[1]{-c}) } @$c) }
20           else { toxml()}}
21      },
22    );

23 print dt($filename,%h);

24 sub dimTabela{
25   my $t=shift;
26   my $nrows = @$t;
27   my $ncolumns = (sort map { scalar(@{$_->{-c}}) } @$t)[-1];
28   ($nrows,$ncolumns);
29 }
```

Descrição do programa:

linha 1: Está a ser usado o módulo perl para gerar HTML (permitindo o uso de funções com nome igual às várias etiquetas HTML como h1, ul, li, dl, etc)
linha 2: Declara-se uma opção de linha de comando.
linha 6 a 7: definição de tipos e modo de processamento de table e tr. Esta definição leva a que o `$c` correspondente seja uma lista de listas que é processada na entrada `table` e na função `dimTabela`.
linha 11 a 21: Processamento de tabelas...
linha 12: determinação das dimensões da tabela
linha 14: ... tabela 1×1 – faz um cabeçalho
linha 15: ... tabela $1 \times n$ – faz uma unorded list
linha 16: ... tabela $n \times 1$ – faz uma orded list
linha 18: ... tabela $2 \times n$ – faz uma description list
linha 19: ... tabela $n \times n$
linha 20: Apesar de `$c` ser uma lista de listas, a função `toxml` sabe reconstituir o XML (HTML) original.
linha 24 a 29: Função determinadora das dimensões da tabela. Ignorar os detalhes (está a ser usado perl avançado...)

Considere-se o seguinte documento HTML:

```
<body>
  <table width="100%" bgcolor="red" border="1">
    <tr> <td> Um estranho título (table 1 x 1) </td> </tr>
  </table>
  <p>Seguidamente uma tabela normal (2 x 3)</p>
  <table border="1">
    <tr> <td> Tomates </td> <td> Vermelhos </td> </tr>
    <tr> <td> Couves  </td> <td> Verdes    </td> </tr>
    <tr> <td> Pencas  </td> <td> Amarelas  </td> </tr>
  </table>
  <p>Seguidamente uma tabela com uma coluna</p>
  <table border="1">
    <tr> <td> Sopa            </td> </tr>
    <tr> <td> ovos estrelados </td> </tr>
    <tr> <td> Pudim           </td> </tr>
  </table>
</body>
```

Na figura 1 mostra-se o resultado de visualizar a página depois de processada (com a opção `-mapping_as_dl`) e apresenta-se também o aspecto inicial.

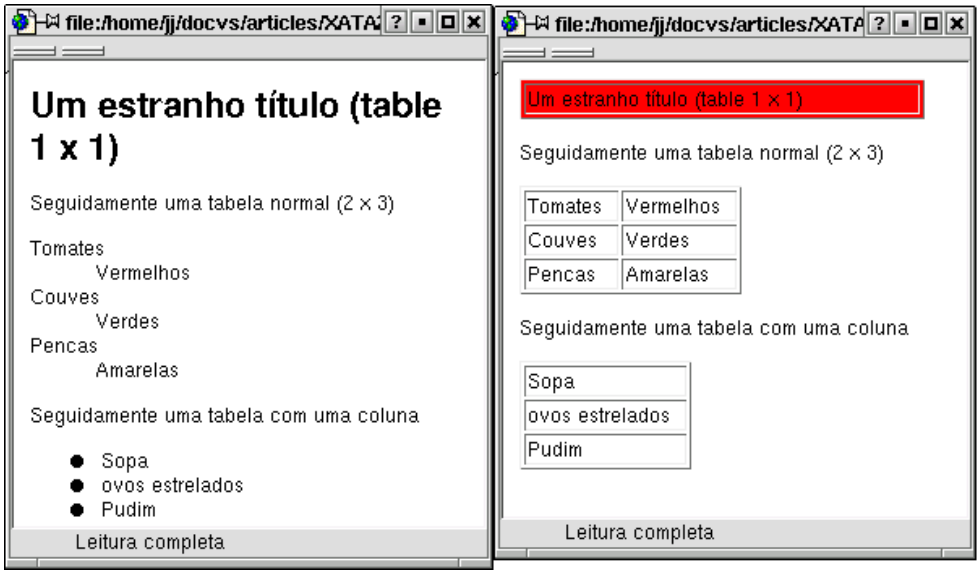


Figura 1. Exemplo de tabelas

5 Conclusões e trabalho futuro

Os exemplos aqui apresentados são propositadamente pequenos e simplicistas mas acreditamos que demonstram as potencialidades da abordagem. Todos estes exemplos terão de cooperar numa mesma ferramenta para que seja possível processar páginas obtendo resultados realmente interessantes.

O objectivo desta ferramenta é ser incluída num serviço de *Proxy* para *browsers* de invisuais que torne uma página HTML complicada numa mais simples de ler por um sintetizador de voz, e para situações em que a acessibilidade seja crítica.

Está a decorrer também um conjunto de testes que usam esta abordagem num leque mais vasto de problemas ligados a processamento de HTML, incluindo web-mining.

Referências

1. *eXtended Markup Language (XML) version 1.0 recommendation.* World Wide Web Consortium, 10 February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.html/.
2. *Extensible Stylesheet Language (XSL), Version 1.0.* World Wide Web Consortium, 15 October 2001. http://www.w3.org/TR/xsl/.
3. *Cascading Style Sheets, level 1.* World Wide Web Consortium, 17 January 1999. http://www.w3.org/TR/REC-CSS1.
4. *HyperText Markup Language Version 4.0 (HTML) recommendation.* World Wide Web Consortium, 24 April 1998. http://www.w3.org/TR/1998/REC-html40-19980424.
5. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, May 1999.

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

14 Fev.	S5(9.00h) Dialectos XML	[*Especificação XML de Aplicações para WWW*] - **Alexandre Martins**, DI-UM, FEUP-UP; **Pedro Rangel Henriques**, DI-UM; **Gabriel David**, FEUP - UP; [*Xexam - uma linguagem de suporte para exames online (e-learning)*] - **Daniel Edgar Pinto Soares** , DI-UM; **Maria da Conceição Vieira Mota**, DI-UM; **José Carlos Ramalho**, DI-UM; [*XML Templates for Constraints (XTC), Um Nível de Abstracção para Linguagens de Especificação de Restrições*] - **Marta Henriques Jacinto**, DI-UM, ITIJ-MJ; **Pedro Rangel Henriques**, DI-UM; **José Carlos Ramalho**, DI-UM;

Especificação XML de Aplicações para WWW

Alexandre Martins[1], Pedro Rangel Henriques[2], and Gabriel David[3]

[1] Aluno de Mestrado em Tecnologias Multimédia
Faculdade de Engenharia
Universidade do Porto
[2] Departamento de Informática
Universidade do Minho
[3] Faculdade de Engenharia
Universidade do Porto

Resumo A necessidade de disponibilizar e manter a informação constantemente actualizada tem conduzido a que, cada vez mais, as aplicações para WWW recorram a bases de dados para suportar o seu funcionamento. Apesar de existir um conjunto de tecnologias que visam facilitar o processo de integração das páginas Web com as bases de dados, o desenvolvimento deste tipo de aplicações ainda é um processo repetitivo e moroso.
Na tentativa de minimizar esse esforço de desenvolvimento, criou-se um dialecto XML, designado WASL (*Web Application Specification Language*), que permite a especificação dos vários componentes responsáveis pela interacção da aplicação com uma base de dados relacional.
Uma especificação WASL consiste em dois documentos XML: o primeiro descreve o modelo de dados relacional existente na base de dados; o segundo contém a descrição da aplicação propriamente dita, em particular o conteúdo dinâmico e o mecanismo de navegação das diversas páginas que a compõem. Com base nesta especificação, um sistema de processamento gera o código necessário à implementação da aplicação.
Neste artigo descreve-se a linguagem WASL bem como o seu sistema de processamento. Para exemplificar a sua utilização, apresenta-se um *case study* de especificação em WASL de uma aplicação para WWW existente e, após a geração do respectivo código, comparam-se os resultados obtidos com a aplicação original.

1 Introdução

O constante crescimento do número de utilizadores da World Wide Web (WWW) bem como das suas expectativas, tem obrigado as organizações a disponibilizarem cada vez mais informação. A grande concorrência que tem imperado nesta área também pressiona constantemente estas organizações a tornarem os seus sites mais completos, mais complexos e mais actualizados.

Na tentativa de minimizar o esforço de desenvolvimento e manutenção deste tipo de aplicações, idealizou-se um sistema que partindo de uma especificação dos diversos componentes responsáveis pela interacção da aplicação com uma base de dados, gerasse o código necessário à sua implementação. A linguagem teria de ser suficientemente genérica de forma a permitir a especificação de um vasto conjunto de aplicações [Mar02].

Para o desenvolvimento da especificação optou-se pela utilização do *eXtensible Markup Language*(XML). Esta meta-linguagem permite ao utilizador a criação da sua própria linguagem de marcas. Assim, a especificação de uma aplicação será um documento anotado com um conjunto de marcas apropriadas para definir o conteúdo das suas páginas, navegação e operações sobre a base de dados.

O objectivo deste projecto foi construir um sistema que partindo de uma especificação XML dos diversos componentes que constituirão a aplicação (modelo relacional da base de dados, formulários de inserção e actualização, consultas, remoções, etc), fosse capaz de gerar os diversos elementos de *software* que permitam o funcionamento da aplicação, minimizando, tanto quanto possível, o esforço do desenvolvimento de aplicações com *interface* em WWW.

Neste artigo descreve-se, o sistema criado, em particular o dialecto XML criado, designado WASL (*Web Application Specification Language*). Posteriormente, apresenta-se um exemplo de especificação de uma aplicação para WWW existente e comparam-se os resultados obtidos depois da geração do código com a aplicação original.

Relativamente à sua estrutura, na secção 2 descreve-se um conjunto de sistemas de modelação e desenvolvimento de aplicações Web que partilham alguns objectivos e estratégias com o sistema desenvolvido neste projecto.

Seguidamente, na secção 5, apresenta-se e descreve-se a sintaxe da linguagem WASL.

O exemplo de especificação e uma aplicação para WWW é apresentado na secção 3.

Finalmente, na secção 5, mostram-se as conclusões retiradas da elaboração deste projecto e apresentam-se algumas orientações para trabalho futuro.

2 Metodologias e ferramentas de desenvolvimento de aplicações para WWW

Na tentativa de minorar o esforço empregue no desenvolvimento de aplicações para WWW, diversas ferramentas e metodologias têm sido criadas. Nesta secção faz-se um levantamento e descrição de alguns desses projectos no domínio da investigação académica.

Devido à enorme quantidade de ferramentas e metodologias existentes, a pesquisa, que foi efectuada, centrou-se naquelas que se inserem no contexto deste projecto: especificação (ou modelação) e geração automática de aplicações para WWW que interagem com repositórios de informação, em particular, bases de dados relacionais. Ferramentas mais centradas no design ou criação de páginas HTML, por exemplo, não serão aqui abordadas.

Um estudo mais abrangente das diferentes ferramentas e metodologias existentes (comerciais e académicas), considerando as diferentes categorias de soluções, pode ser encontrado em [Fra99]. Um outro estudo, mais orientado para projectos académicos, de aplicação de conceitos de bases de dados ao desenvolvimento de aplicações Web é apresentado em [FLM98].

2.1 Araneus

O projecto Araneus introduz uma metodologia para a construção e manutenção de Web sites baseado no princípio que os dados a serem disponibilizados no site são geridos através de um SGBD [AMM97a].

O processo de desenvolvimento é o resultado da inter-ligação de duas actividades: o *desenho da base de dados* e o *desenho do hipertexto*. Cada uma destas actividades é ainda dividida em *desenho conceptual* e *desenho lógico* baseados em modelos de dados específicos.

O ponto de partida do processo é o desenho conceptual da bases de dados, gerando um modelo Entidades-Relações. Seguidamente, o modelo lógico (e possivelmente físico) da base de dados é gerado utilizando técnicas habituais de desenho e implementação de bases de dados.

O desenho conceptual do hipertexto (modelo NCM) descreve como as principais entidades da aplicação irão estar organizadas no site. Esta actividade é independente da implementação física (em termos de páginas e *links*) e concentra-se em dois aspectos essenciais:

- decidir quais as entidades que farão parte do hipertexto, i.e., que vão corresponder aos "nodos" do hipertexto;
- escolher os caminhos para navegar entre as entidades, i.e., como vai ser possível explorar a rede de nodos.

A etapa seguinte consiste no desenho lógico do hipertexto recorrendo ao modelo de dados ADM (*Araneus Data Model*) [AMM97b]. Neste modelo faz-se uma representação das páginas Web, recorrendo a esquemas de páginas (*page schemas*). Nestes esquemas existe uma representação do conteúdo de cada página ao nível das entidades e atributos que a constituem, assim como as hiperligações.

Neste esquema especificam-se já certos aspectos muito próximos das páginas HTML, por exemplo: se cada instância da entidade aparece numa página individual, ou se aparecem todas numa lista; especificam-se as *forms*; indica-se se o valor do atributo é texto, imagem ou outros tipos MIME.

A metodologia propõem um conjunto de passos para fazer a transformação do modelo NCM para o modelo ADM. O modelo ADM é um modelo visual que permite uma visualização bastante próxima de como será constituído o site ao nível das suas páginas, da estrutura navegacional de hiperligações e da informação contida em cada uma (listas de atributos, imagens, forms, etc).

O passo seguinte, consiste em fazer o mapeamento desta estrutura com a respectiva base de dados.

Ao nível da apresentação, o ARANEUS propõe uma abordagem baseada em *templates* e *page-styles* (baseados em HTML [MMAC99]) para cada *page schema*. Uma *page-style* especifica como os valores de cada atributo do *page-schema* deverão ser formatados na página, para além de uma secção de cabeçalho e rodapé.

Com base neste nestas *page-styles* e na especificação PENELOPE anterior, são geradas as páginas HTML (ou XML).

Um aspecto que não é abrangido por este projecto é a questão da personalização de conteúdos. Na bibliografia pesquisada não foi encontrada nenhuma referência a estas questões.

2.2 Strudel

O sistema Strudel baseia-se em conceitos de sistemas de gestão de bases de dados aplicados ao desenvolvimento e manutenção de Web sites. Em particular, este sistema suporta a especificação declarativa do conteúdo e estrutura de um site e, posteriormente, a sua geração automática. Um princípio base do Strudel é a separação entre os dados do site, a composição e estrutura do site (páginas e ligações) e a apresentação das suas páginas [FFKL98].

Os modelos do Strudel (quer para a estrutura de dados quer para a composição e estrutura navegacional das páginas) são baseados em grafos orientados e etiquetados. Trata-se de um modelo semi-estruturado, permitindo a modelação quer de dados estruturados, quer de dados não-estruturados, isto é, que não obedecem a um esquema regular.

Os dados que formam o conteúdo do site são especificados através de um grafo designado *data graph*. Este modelo permite a integração de dados de múltiplas e distintas origens (e.g. bases de dados, ficheiros textuais, páginas HTML, etc).

A definição do conteúdo das páginas e da estrutura do site é realizada através de um conjunto de *queries* sobre este modelo de dados, utilizando a linguagem *StruQL*

(*Strudel's Query Language*). Estas *queries* permitem a selecção dos dados a serem incluídos e a definição das hiperligações existentes. Como resultado destas *queries* é criado um novo grafo (*site graph*) que descreve a estrutura do site.

A apresentação das páginas é realizada através de *templates* HTML, constituídos pelos habituais elementos da linguagem e por um conjunto de marcas próprias do sistema que interligam os dados definidos nas *queries* anteriores. Com base nestes *templates* e nas *queries* definidas anteriormente é gerado o site.

2.3 WebML

A *Web Modeling Language* (WebML) é uma linguagem de especificação de aplicações Web, permitindo uma descrição de alto-nível da aplicação segundo cinco vertentes. Para cada uma destas vertentes existe um modelo associado [CFB00]:

- estrutura de dados (*Structural model*);
- descrição das páginas da aplicação (*Composition model*)
- estrutura de hiperligações entre páginas (*Navigation model*)
- apresentação (*Presentation model*)
- personalização (*Personalization model*)

Através dos diversos modelos, é feita uma descrição a um nível conceptual dos vários componentes da aplicação (modelo de dados, composição de páginas, etc). Obtém-se assim, independência em relação às linguagens de implementação de cada componente.

Todos os conceitos do WebML têm associados uma notação gráfica e, paralelamente, uma descrição textual baseada em XML. Com base nesta especificação, uma aplicação gerará o código necessário à sua implementação.

O *Structural model* descreve a estrutura dos dados que constituem o conteúdo da aplicação. Tem como base modelos de dados já existentes em várias áreas da engenharia de *software*. É baseado no modelo Entidades/Relações, mas apresentado como sendo compatível com o diagrama de classes UML, utilizado na modelação orientada a objectos.

No modelo *Composition Model* são especificadas as páginas que constituirão o site. As páginas são compostas por *content units*, isto é, componentes que indicam quais os dados a serem inseridos na página (retirados do E/R) e a forma de apresentação (listas, índices, etc). Existem seis tipos de *content units*:

- *Data units*: mostram informação de apenas uma instância de uma entidade;
- *Multi-data units*: permitem visualizar várias instâncias de uma entidade;
- *Index units* apresentam várias instâncias de um determinado objecto (entidade ou relação) sob a forma de uma lista; não apresentam a informação detalhada de cada instância mas apenas um subconjunto dos seus atributos;
- *Scroll units* disponibilizam comandos para percorrer um conjunto ordenado de objectos (primeiro, último, anterior, próximo, *i-ésimo*);
- *Filter units* apresentam um conjunto de campos para introdução de valores que serão utilizados para pesquisar os objectos que contêm a informação introduzida pelo utilizador
- *Direct units* expressam a ligação entre um objecto e outro que lhe esteja relacionado através de uma relação um-para-um ou n-para-um

As *units* e as páginas são ligadas entre si através de *ligações* ("*links*") formando a estrutura hipertextual da aplicação.

O modelo navegacional do WebML, *Navigation model*, permite dois tipos de ligações:

– Ligações contextuais ("*Contextual links*"): transportam informação (contexto) entre a unidade origem e a unidade destino;
– Ligações não-contextuais ("*Non-contextual links*"): ligam páginas onde não existe uma relação directa entre a página origem e a página destino;

De forma a permitir operações de inserção, remoção ou actualização pelo utilizador (ou mesmo outras operações mais complexas), o WebML estende os seus modelos de composição de páginas e navegação com elementos que permitam especificar operações e a entrada de dados [BCFM00].

A apresentação das páginas é gerada de acordo com folhas de estilo que definem a composição gráfica das páginas. Estas folhas de estilo são construídas utilizando XSL. Existem dois tipos de folhas de estilo; umas descrevem as páginas de uma forma genérica, independentemente do seu conteúdo; um segundo tipo, mais detalhado, descreve páginas atendendo aos elementos de informação e conceitos nela contidos.

De forma a permitir que uma aplicação disponibilize conteúdos personalizados o WebML fornece um conjunto de funcionalidades que permitem ao programador especificar os mecanismos de personalização segundo três vertentes [CFP99]:

– Modelação de perfis de utilizador e de grupo;
– Especificação de conteúdos que são dependentes do utilizador;
– Especificação de acções para responder a eventos que o utilizador desencadeia quando está a interagir com a aplicação;

3 WASL - *Web Application Specification Language*

Na figura 1, apresentam-se os diversos componentes do sistema associado à linguagem WASL.

De acordo com a análise de aplicações existentes e com o estudo das metodologias e ferramentas descrito na secção 2, pode-se concluir que existem três grandes vertentes nas aplicações para WWW:

– dados: a informação que se disponibiliza na aplicação;
– navegação: a forma de percorrer as diversas páginas que constituem a aplicação;
– apresentação: o *look and feel* das páginas e a *interface* com o utilizador;

Assim, o ponto de partida para a utilização desta linguagem será a descrição da base de dados relacional que armazena os dados da aplicação. Pressupõem-se neste sistema que essa base de dados relacional já existe implementada num SGBD relacional. Essa descrição é realizada através de um documento XML, que define a estrutura de tabelas, campos, chaves primárias e chaves estrangeiras. O vocabulário XML utilizado é descrito na secção 3.1.

O segundo componente do sistema é a descrição das diversas páginas que compõem a aplicação, considerando a interacção com a base de dados, isto é, as operações que são realizadas sobre a base de dados relacional, e a vertente de navegação hipertextual da aplicação. Para tal, utiliza-se um outro documento XML, cuja sintaxe se descreve na secção 3.2. Esta separação entre os documentos visa permitir aos utilizadores a especificação de várias aplicações com base na mesma especificação da base de dados relacional.

Com base nestes dois documentos, um componente de transformação de XML, encarrega-se de gerar o código que implementa a aplicação. Este código tem duas componentes:

– visualização e interacção com o utilizador (HTML[4]);

[4] Efectivamente, é gerado XHTML 1.0. Como esta linguagem é apenas uma redefinição de HTML segundo as regras da linguagem XML, utiliza-se neste documento apenas HTML

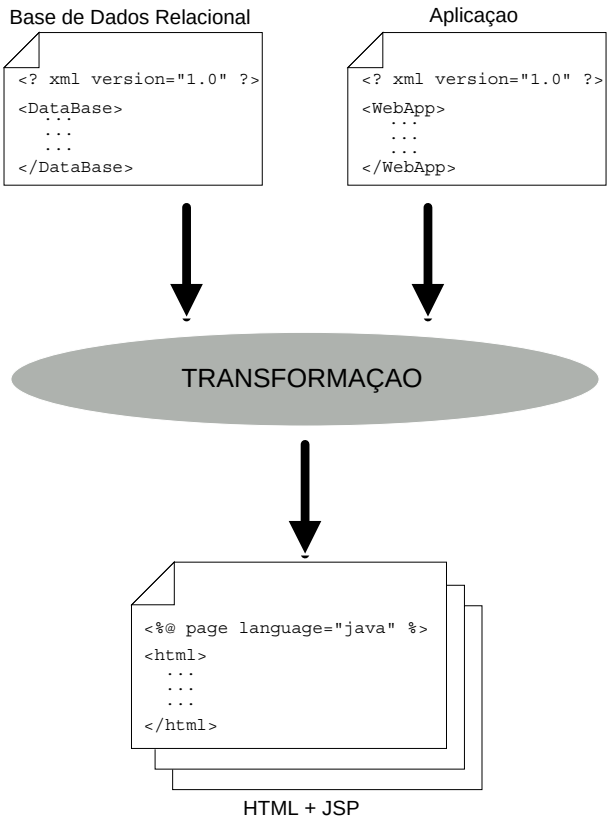


Figura 1. Arquitectura do sistema WASL

– execução das operações sobre a base de dados relacional (*Java Server Pages* - JSP).

A definição da apresentação da aplicação será definida recorrendo às linguagens de estilo existentes para a apresentação de HTML e XML (CSS e XSL). Como o código gerado é apenas XHTML 1.0, apenas são utilizadas marcas de definição da estrutura do documento. O utilizador pode assim definir folhas de estilo onde define a apresentação de cada uma destas marcas.

3.1 Especificação XML da base de dados relacional

De forma a permitir a geração do código, é necessário ao sistema ter uma descrição dessa base de dados que permita identificar as diversas tabelas, os seus campos, chaves primárias e chaves estrangeiras. Assim, foi criado um DTD que estabelece a sintaxe dos documentos XML que descrevem a base de dados relacional.

Uma vez que não era objectivo do sistema a geração de scripts SQL para a criação da base de dados, alguns componentes da base de dados não foram incluídos na linguagem. Por exemplo, neste DTD não se considerou certas particularidades da base de dados, tais como, *stored procedures*, transacções ou índices. Com esta especificação, pretendia-se apenas conhecer os elementos necessários à geração do código (quer HTML, quer JSP); para tal era apenas necessário descrever a estrutura de tabelas da BD, considerando os seguintes aspectos:

– Campos;

– Chaves primárias;
– Chaves estrangeiras de forma a conhecer as relações entre as diversas tabelas;
– Tipos de dados dos campos e tamanho, de forma a permitir, efectuar algumas validações de tipos de dados e para a construção dos *forms* HTML.

Assim, o documento que descreve a base de dados relacional deve obedecer ao DTD que seguidamente se apresenta.

O elemento `DataBase` contém um conjunto de atributos cujos valores permitem definir a ligação ao servidor da base de dados relacional, tais como o *url* do servidor, *user*, *password* e *driver*.

O elemento `Field` contém um conjunto de atributos que permitem caracterizar um campo: tipo de dados, tamanho, valor por omissão, se permite valores nulos, etc. O conteúdo do campo corresponde ao nome descritivo do campo que surge nos *forms* como rótulo, enquanto que `id` poderá ser uma versão simplificada desse nome, de forma a ser mais facilmente referenciado.

O elemento `PrimaryKey` permite definir a chave primária da tabela indicando os indentificadores do(s) campo(s) que a formam (`fieldId`).

Relativamente ao elemento `ForeignKey` que define uma chave estrangeira, é composto pelo elemento `FieldRef` que representa o(s) campo(s) da tabela em questão, enquanto que os elementos `ForeignTable` e `ForeignField` indicam, respectivamente a tabela estrangeira e o(s) seu(s) campo(s).

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ENTITY % Types "(CHAR|VARCHAR|BIT|BITVARYING|NUMERIC|
                   DECIMAL|INTEGER|SMALLINT|FLOAT|DATE|
                   TIME|TIMESTAMP)">

<!ELEMENT DataBase (Table)+>
<!ATTLIST DataBase
          id CDATA #REQUIRED
          url CDATA #REQUIRED
          user CDATA #REQUIRED
          passwd CDATA #REQUIRED
          driver CDATA #REQUIRED>

<!ELEMENT Table (Field+, PrimaryKey, ForeignKey*)>
<!ATTLIST Table
          id ID #REQUIRED>

<!ELEMENT Field (#PCDATA)>
<!ATTLIST Field
          id CDATA #REQUIRED
          type %Types; #REQUIRED
          size CDATA #IMPLIED
          decimal CDATA #IMPLIED
          null (yes | no) "yes"
          defaultValue CDATA #IMPLIED>

<!ELEMENT PrimaryKey (KeyField)+>

<!ELEMENT KeyField EMPTY>
<!ATTLIST KeyField
          fieldId CDATA #REQUIRED>

<!ELEMENT ForeignKey (FieldRef+, ForeignTable)>
```

```
<!ELEMENT FieldRef EMPTY>
<!ATTLIST FieldRef
          fieldId CDATA #REQUIRED>

<!ELEMENT ForeignTable (ForeignField)+>
<!ATTLIST ForeignTable
          tableId CDATA #REQUIRED>

<!ELEMENT ForeignField EMPTY>
<!ATTLIST ForeignField
          fieldId CDATA #REQUIRED>
```

Exemplo de especificação Para exemplificar, apresenta-se uma hipotética base de dados relacional, descrita em XML segundo este DTD. Nesta base de dados pretende-se armazenar informação de filmes e dos actores que desempenharam papéis nesse filme. Assim, a base de dados é constituída por três tabelas: `Actor`, onde se armazena a informação dos diversos actores; `Filme`, onde se armazenam dados de cada filme; `FilmeActor` que permite relacionar os filmes com os actores que nele participam.

Exemplo 31 *Especificação XML de uma base de dados relacional*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DataBase SYSTEM "bd.dtd">

<DataBase id="teste" url="jdbc:postgresql:teste" user="xpto"
          passwd="xpto" driver="org.postgresql.Driver">

  <Table id="Filme">
    <Field id="idFilme" type="INTEGER">Código</Field>
    <Field id="titulo" type="VARCHAR" size="40">Título</Field>
    <Field id="anoFilme" type="INTEGER">Ano</Field>
    <PrimaryKey>
      <KeyField fieldId="idFilme"/>
    </PrimaryKey>
  </Table>

  <Table id="Actor">
    <Field id="idActor" type="INTEGER">Código</Field>
    <Field id="nome" type="VARCHAR" size="25">Nome</Field>
    <Field id="dataNasc" type="DATE">Data de Nascimento</Field>
    <PrimaryKey>
      <KeyField fieldId="idActor"/>
    </PrimaryKey>
  </Table>

  <Table id="FilmeActor">
    <Field id="idFilme" type="INTEGER">Código de Filme</Field>
    <Field id="idActor" type="INTEGER">Código de Actor</Field>
    <Field id="personagem"
           type="VARCHAR" size="20">Personagem</Field>
    <PrimaryKey>
      <KeyField fieldId="idFilme"/>
      <KeyField fieldId="idActor"/>
    </PrimaryKey>
    <ForeignKey>
      <FieldRef fieldId="idFilme"/>
      <ForeignTable tableId="Filme">
        <ForeignField fieldId="idFilme"/>
      </ForeignTable>
```

```
    </ForeignKey>
    <ForeignKey>
      <FieldRef fieldId="idActor"/>
      <ForeignTable tableId="Actor">
        <ForeignField fieldId="idActor"/>
      </ForeignTable>
    </ForeignKey>
  </Table>

</DataBase>
```

□

3.2 Especificação XML da aplicação

Após se ter construído o documento XML que especifica a base de dados, constrói-se o documento XML que descreve as páginas Web que constituirão a aplicação (recordar figura 1). Nesta secção descreve-se a sintaxe concebida para essa especificação.

Para manter um bom nível de clareza e abstracção, apresenta-se a especificação da linguagem usando a notação EBNF da linguagem.

Uma aplicaãoo para WWW é definida por um conjunto de páginas interligadas entre si através de hiperligações. Cada página (`Page`) é definida pelo seu conteúdo (`Content`) e por um conjunto de hiperligações (`MenuLink`).

Para definir a aplicação são ainda considereados um conjunto e atributos como o seu nome, a página de entrada `homePage`, o ficheiro contendo a especificação da base de dados associada (`dbSpec`), uma designação (`name`) e a localização onde o código gerado será armazenado (`rootURL`).

```
WebApplication ::= name homePage dbSpec rootURL Page+

Page ::= pageId title Content MenuLink*
```

Como referido anteriormente, o principal objectivo da linguagem é a especificação dos componentes de interacção com a base de dados. Estes componentes correspondem às operações que se podem realizar sobre a base de dados. Assim, o conteúdo das páginas estará dependente destas operações. Existem fundamentalmente dois tipos de conteúdos: o resultado de uma consulta à base de dados (`Query`); ou um conjunto de componentes de interacção (`Action`), que permitem ao utilizador a realização de inserções, remoções, alterações ou a pesquisa de valores.

De forma a permitir a inserção de informação estática numa página, achou-se conveniente, permitir a importação de conteúdo textual existente num ficheiro. Este conteúdo poderá estar escrito em linguagem HTML ou XML (valores possíveis para `format`). A localização do ficheiro é definida por `url`.

```
Content ::= Include* (Query | Action)?

Include ::= format url

format ::= 'HTML' | 'XML'
```

Para definir uma consulta à base de dados, o elemento `Query` terá de conter a especificação dos campos e respectivas tabelas, que farão parte da consulta. Para seleccionar todos os campos de uma tabela, sem ter de especificar todos individualmente, pode-se utilizar o elemento `SelectAll`, indicando apenas a tabela a consultar.

```
Query ::= id (QueryField+ | SelectAll)
```

Um `QueryField` é definido por um par de identificadores tabela/campo existente na base de dados. Por sua vez um `SelectAll` é definido apenas pelo identificador da tabela.

Para permitir a existência de hiperligações nos valores de um determinado campo retornado pela base de dados, para uma outra página que apresenta informação relativa àquele item, utiliza-se o elemento `Link`.Permite-se assim a especificação de navegação contextual inserida no conteúdo apresentado.

```
QueryField ::= tableId fieldId Link?

SelectAll ::= tableId
```

Para criar uma hiperligação contextual, é necessário passar o valor onde está a hiperligação para a página destino. No entanto, a página que apresenta os dados de um determinado item necessita de saber o identificador do item a apresentar.

Em consequência disto é necessário que seja definido um elemento `Parameter` que poderá ser especificado nas hiperligações contextuais com o valor do parâmetro a passar. Para além disso, na especificação de uma página também poderá existir este elemento de forma a indicar que a página em questão necessita de receber um parâmetro. Em consequência, acrescenta-se à definição de `Page` o elemento `Parameter`

```
Page ::= pageId title Parameter* Content MenuLink*
```

Por sua vez, a definição de `Link` e `MenuLink` passa a:

```
Link ::= pageId Parameter*

MenuLink ::= pageId descriptiveName Parameter*
```

O `descriptiveName` define o texto que aparece na hiperligação, por exemplo: "Voltar", "Ver actores deste filme"ou "Anterior".

Um `Parameter` é definido pelo par identificador da tabela/campo e eventualmente um valor (`value`) quando se pretende passar um parâmetro com um valor sempre constante.

```
Parameter ::= tableId fieldId value?
```

Relativamente a `Action`, este elemento permite definir uma das restantes operações sobre a base de dados.

```
Action ::= Insert | Update | Delete | Search
```

`Insert` permite especificar páginas para inserção de valores numa determinada tabela, ou em mais do que uma tabela caso estejam relacionadas. A relação entre as tabelas, deverá ser do tipo *Master-Detail* (ou *1 para N*). Quando é concluída a operação de inserção, a aplicação deverá apresentar uma nova página. Caso tenha sido concluída com sucesso a página a apresentar será a definida por `successPage`, caso contrário, será `failurePage`.

```
Insert ::= id successPage failurePage Table+ LookUpValue*

successPage ::= pageId
failurePage ::= pageId
```

O elemento `LookUpValue` permite definir que para um determinado campo se pretende consultar uma tabela para seleccionar o valor a inserir nesse campo. Por exemplo, numa inserção numa tabela de facturas, permitir uma pesquisa na tabela de clientes para seleccionar o código de cliente a quem respeita a factura. Será constituído pelo campo para o qual se pretende pesquisar um valor (`fieldId`), pela tabela a pesquisar (`lookUpTable`) e pelo campo dessa tabela a retornar (`returnField`).

```
LookUpValue ::= fieldId lookUpTable returnField
```

O elemento `Delete` permite definir a remoção de um registo numa tabela. `successPage` e `failurePage` têm a mesma funcionalidade referida anteriormente.

```
Delete ::= id successPage failurePage tableId
```

Por sua vez, `Update` permite especificar a actualização de um determinado registo de uma tabela, ou tal como na inserção, em mais do que uma tabela desde que estejam relacionadas.

```
Update ::= id successPage failurePage tableId+ LookUpValue*
```

Estas definições pressupõem que as hiperligações para uma remoção ou actualização deverão existir numa página onde se apresente um determinado registo. Ao clicar, por exemplo, em "Remover" o registo será removido e surgirá a respectiva página de sucesso (ou insucesso no caso de ter ocorrido algum erro).

Por último, `Search` permite especificar uma procura de registos numa tabela segundo determinados critérios. Para tal especificam-se quais os campos (`SearchField` ou `All`) da tabela (`SearchTable`) onde se introduzirão os critérios assim como o resultado que se pretende retornar ao utilizador (`SearchResult`).

```
Search ::= id SearchTable SearchResult

SearchTable ::= tableId (SearchField+ | All)

SearchField ::= fieldId
```

A definição do resultado que será apresentado corresponde a uma lista dos campos que serão retornados. Estes poderão, tal como nas consultas, conter hiperligações contextuais para outras páginas.

```
SearchResult ::= ResultField+ | All

ResultField ::= fieldId Link?
```

Com base nesta definição abstracta da linguagem foi construído o DTD que permite validar os documentos XML que descrevem a aplicação.

Resumidamente, os elementos terminais da sintaxe deram origem a atributos, enquanto os elementos não terminais deram origem a elementos de um DTD.

Exemplo de especificação De forma a complementar a descrição da linguagem, apresenta-se a especificação de uma hipotética aplicação que recorre à base de dados exemplo sobre filmes e actores, apresentada na secção anterior (exemplo 31).

No elemento raiz deste documento destaca-se a indicação o documento XML que contém a especificação da base de dados e a página de entrada na aplicação (atributo `homepage`). O conteúdo dessa página está contido no ficheiro `index-input.html` e contém duas hiperligações para a página `ActoresMain` e `FilmesMain` (que hipoteticamente seriam as página de entrada nas secções de Actores e Filmes, respectivamente).

Exemplo 32 *Elemento raiz da especificação XML de uma aplicação*

```
<WebApplication name="Teste" homePage="index" dbSpec="bd1.xml"
                rootURL="Output/">
  <Page id="index" title="Homepage">
    <Content>
      <Include format="HTML" url="index-input.html"/>
    </Content>
    <MenuLink pageId="ActoresMain" descriptiveName="Actores"/>
    <MenuLink pageId="FilmesMain" descriptiveName="Filmes"/>
 </Page>

 ...

</WebApplication>
```

□

Supondo que se pretende uma página que permita a pesquisa de filmes com base nos critérios `titulo` e `anoFilme` introduzidos pelo utilizador e que o resultado da pesquisa deveria devolver uma lista com o identificador de cada filme e o título, a especificação será:

Exemplo 33 *Especificação de uma página para pesquisa de filmes*

```
<Page id="PesquisaFilme" title="Procurar Filme">
  <Content>
    <Action>
      <Search id="pesqFilme">
        <SearchTable tableId="Filme">
          <SearchField fieldId="titulo"/>
          <SearchField fieldId="anoFilme"/>
        </SearchTable>
        <SearchResult>
          <ResultField fieldId="idFilme"/>
          <ResultField fieldId="titulo">
            <Link pageId="DadosFilme">
              <Parameter tableId="Filme" fieldId="idFilme"/>
            </Link>
          </ResultField>
        </SearchResult>
      </Search>
    </Action>
  </Content>
  <MenuLink pageId="index" descriptiveName="Home"/>
  <MenuLink pageId="ListaFilmes" descriptiveName="Ver todos"/>
  <MenuLink pageId="FilmesMain" descriptiveName="Filmes"/>
</Page>
```

□

De referir que na lista de resultados, os nomes contêm uma hiperligação para a página `DadosFilme` que permite ver os dados de um determinado filme. Para além disso, existem duas hiperligações: uma para uma página `ListaFilmes` que apresenta uma lista completa com todos os filmes existentes (sem nenhum critério de pesquisa); outra para a página `FilmesMain`.

3.3 Sistema de processamento

Com base nos dois documentos XML que especificam a base de dados relacional e a aplicação propriamente dita, existe um componente de transformação, desenvolvido em XSLT, que se encarrega de gerar o código XHTML e JSP que implementa a aplicação.

O documento XML que serve de *input* à folha de estilo XSLT é o que descreve a aplicação. A própria folha de estilo é que, sempre que necessário, recorre ao documento que contém a descrição da base de dados relacional para obter informações necessárias à geração do código (e.g. descobrir a relação entre duas tabelas, obter o tipo de dados de um determinado campo, etc).

No desenvolvimento do protótipo deste projecto, optou-se pela utilização de XHTML 1.0 Strict, uma vez que não se pretendia gerar qualquer tipo de marcas de apresentação, permitindo assim que o utilizador usasse folhas de estilo CSS ou XSL para esse efeito.

Paralelamente, utilizou-se a tecnologia Java Server Pages para realizar a interligação entre as páginas XHTML e a base de dados relacional. Foi utilizada a biblioteca de marcas denominada *DBTags Tag Library* [Fou00], que permite o acesso a uma base de dados relacional através da execução de comandos na linguagem SQL.

4 *Case Study*

No sentido de validar a linguagem e o sistema de transformação criados, foi necessário especificar um conjunto de aplicações para WWW e testar o código gerado, de forma a verificar se eram atingidos os objectivos esperados. Seleccionou-se um conjunto de aplicações existentes, uma vez que assim seria mais correcta a avaliação dos resultados obtidos. Procurou-se que essas aplicações tivessem conteúdos fundamentalmente dinâmicos, em consonância com os propósitos da linguagem criada.

Nesta secção apresenta-se um extracto de um desses *case-studies* de forma a melhor visualizar os resultados obtidos através da utlização da linguagem WASL.

A aplicação *Foto@pt*[5] é dirigida à comunidade de interessados pela arte de fotografia, permitindo que os utilizadores coloquem as suas fotografias *on-line* e que troquem, quer comentários sobre as fotografias dos restantes membros, quer vários tipos de informação relacionada com a fotografia (e.g. classificados, *links* úteis, etc).

4.1 Base de dados relacional

Nesta aplicação, não existia uma prévia descrição da base de dados, pelo que foi desenhada uma base de dados relacional através da análise das principais entidades e relações que era possível depreender da utilização da aplicação[6].

As principais tabelas são `Autor` e `Foto`.Para além disso um autor tem associada uma lista de temas em que classifica as suas fotos; uma fotografia também se classifica segundo um determinado assunto. Enquanto os temas são criados pelos autores a lista de assuntos é comum a todos.

As restantes tabelas, permitem guardar outras informações, tais como: os comentários que os restantes membros fazem de um determinado autor (tabela `AutorComentario`); os diversos comentários que são efectuados a cada fotografia (tabela `FotoComentario`); os temas da galeria de cada autor (tabela `TemasAutor`).

4.2 Especificação XML da aplicação

A especificação desta aplicação baseou-se na secção relativa ao Membros. Isto é, não foram especificadas, as secções de Classificados, Concursos, Destaques, Informações, etc. Esta secção é onde se encontram as principais funcionalidades da aplicação, no que respeita às principais entidades da aplicação (fotografias e autores). Falta também a especificação de algumas funcionalidades apenas disponíveis aos utilizadores registados (e,g, inserir comentários, inserir fotos), uma vez que à data da escrita deste documento, a aplicação se encontrava com algumas funcionalidades desactivadas.

No *case-study* original foram ainda especificadas outras páginas para além das apresentadas. No entanto, nesta secção apenas são especificadas três dessas páginas de forma a não tornar esta descrição demasiado extensa. Descreve-se a especificação das páginas de pesquisa de autores, a pégina do autor e a página com a lista de temas da sua galeria.

A primeira página especificada permitia efectuar uma pesquisa aos autores existentes, com base nos critérios *id*, *nome* e *email*, como se pode ver na figura 2.

Na especificação em WASL, esta página consistia num componente de pesquisa (`Search`) à tabela `Autor`.

```
<Page id="PesquisaMembro" title="Pesquisar Membros">
  <Content>
    <Action>
      <Search id="PesqAutor">
        <SearchTable tableId="Autor">
          <SearchField fieldId="idAutor"/>
          <SearchField fieldId="nomeCompleto"/>
          <SearchField fieldId="email"/>
```

[5] www.fotopt.net

[6] Não se inclui neste documento a especificação XML devido à sua extensão.

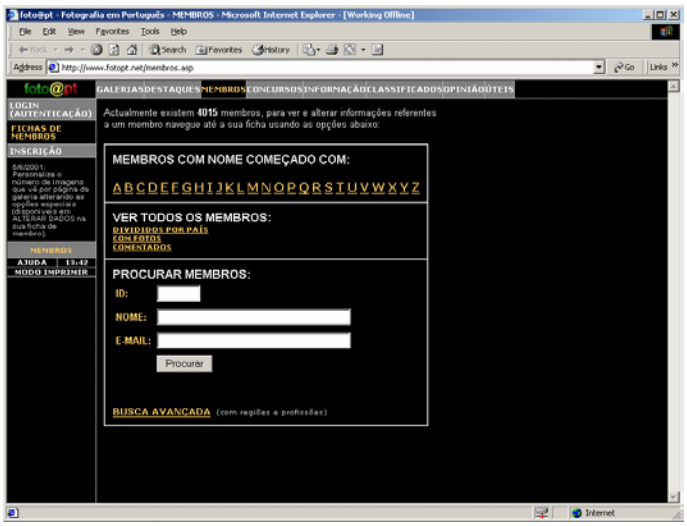


Figura 2. Página de pesquisa de membros

```
        </SearchTable>
        <SearchResult>
          <ResultField fieldId="idAutor"/>
          <ResultField fieldId="nomeSite">
            <Link pageId="Autor">
              <Parameter tableId="Autor" fieldId="idAutor"/>
            </Link>
          </ResultField>
        </SearchResult>
      </Search>
    </Action>
  </Content>
  <MenuLink pageId="PesquisaMembros2"
            descriptiveName="Busca Avançada"/>
</Page>
```

Após a geração foi possível obter a página apresentada na figura 3. No entanto não foi possível especificar e gerar a pesquisa por letra de abecedário, por país, com fotos comentadas e o texto com a informação do número de membros existentes.

Tal como na aplicação original que devolvia uma lista com todos os nomes no site que correspondessem ao critério, também a aplicação gerada retornava essa lista, como se pode verificar na figura 4.

A busca avançada, apenas acrescentava os campos de pesquisa `Regiao` e `Profissao`, pelo que a especificação foi semelhante à anterior, apenas acrescentando os `SearchFields` respectivos.

A partir desta lista o utilizador acedia à página do autor. Nesta página eram apresentados apenas alguns campos da sua ficha e o seu retrato, como se pode ver na figura 5:

Para esta página a especificação em WASL foi:

```
<Page id="Autor" title="Autor">
  <Parameter tableId="Autor" fieldId="idAutor"/>
  <Content>
    <Query id="autor">
      <QueryField tableId="Autor" fieldId="idAutor"/>
      <QueryField tableId="Autor" fieldId="retrato"/>
      <QueryField tableId="Autor" fieldId="nomeCompleto"/>
```

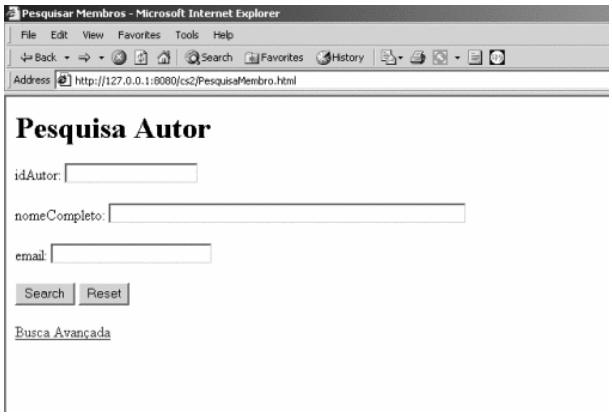


Figura 3. Página `PesquisaMembro` gerada

Figura 4. Resultado de uma pesquisa de membros

```
      <QueryField tableId="Autor" fieldId="nomeSite"/>
      <QueryField tableId="Autor" fieldId="email"/>
      <QueryField tableId="Autor" fieldId="pais"/>
      <QueryField tableId="Autor" fieldId="dataInsc"/>
    </Query>
  </Content>
  <MenuLink pageId="GaleriaAutor"
            descriptiveName="Ver galeria deste autor">
    <Parameter tableId="Autor" fieldId="idAutor"/>
  </MenuLink>
  <MenuLink pageId="LerInserirComentariosAutor"
     descriptiveName="Ler ou inserir comentários ao autor">
    <Parameter tableId="Autor" fieldId="idAutor"/>
  </MenuLink>
  <MenuLink pageId="FotosPreferidas"
            descriptiveName="Fotos preferidas">
    <Parameter tableId="Autor" fieldId="idAutor"/>
  </MenuLink>
</Page>
```

O resultado obtido por esta geração corresponde ao apresentado na figura 6.

Figura 5. Ficha de Autor

Figura 6. Página `Autor` gerada

Nessa figura podemos ver a ficha do autor com o `id 2`, com um retrato cujo ficheiro está localizado em `Imagens/retrato2.jpg`. Este autor tem como nome completo `Alvaro Pereira Mendes`; o nome no site é `Alvaro`; o *email* é `alvaro@nowhere.com`; país `Portugal` e data de inscrição `2002-08-20`.

Na página original (figura 5), surgem diversas hiperligações no menu lateral, designadamente, *Contribuição do Membro*, *Link para esta ficha* e *Recuperar password*, que não foram especificadas. A primeira apresentava uma lista de todas as contribuições que o membro tinha feito na aplicação, tais como número de fotos inseridas, número de comentários a fotos efectuados, etc. Tratava-se de uma consulta à base de dados com contagem de registos, o que não era possível especificar em WASL.

Através da hiperligação *Link para esta ficha*, o utilizador acedia a uma página que apresentava o *url* completo para aceder directamente a esta página. Isto servia para estabelecer hiperligações de outros sites directamente para a página deste autor. A hiperligação *Recuperar password* liga a uma página onde o utilizador pode requisitar o reenvio da sua *password* para autenticação no site. Estas duas páginas não foram especificadas, uma vez que não era claro que a informação apresentada fosse dependente de acesso à base de dados.

O retrato do autor também não é apresentado na página gerada, mas apenas o *url* do local onde está armazenada essa imagem. Numa implementação real ter-se-ia de incluir esse *url* dentro da respectiva marca HTML que permite apresentar imagens.

Partindo da página do autor poder-se-ia aceder à sua galeria de fotografias. Nesta página, para além de alguma informação sobre o autor, apresentava-se a lista de temas do autor. A imagem da aplicação original pode ser visualizada na figura 7.

Caso o utilizador não pretende-se percorrer a galeria por temas tinha também a hipótese de percorrer todas as fotografias mas sem a divisão por temas. Para além disso, poder-se-ia aceder a um conjunto de fotos seleccionadas pelo próprio autor.

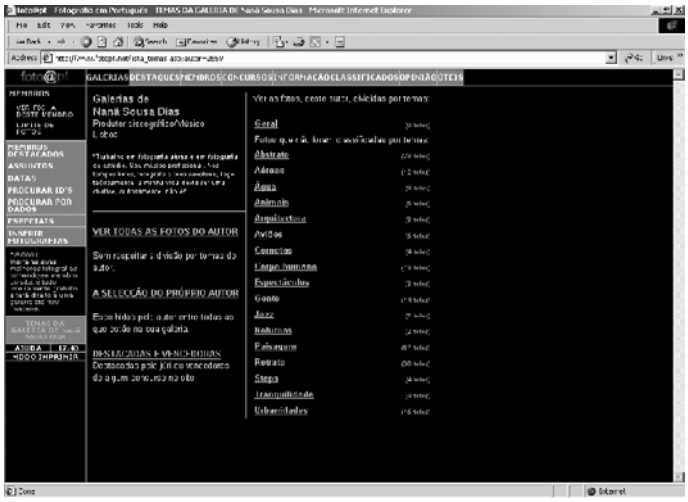


Figura 7. Temas da galeria de um autor

A especificação em WASL consistia na consulta às tabelas `Autor` e `TemasAutor`, recebendo como parâmetro o identificador do autor em questão:

```
<Page id="GaleriaAutor" title="Galeria de Autor">
  <Parameter tableId="Autor" fieldId="idAutor"/>
  <Content>
    <Query id="GaleriaAutor">
      <QueryField tableId="Autor" fieldId="idAutor"/>
      <QueryField tableId="Autor" fieldId="nomeCompleto">
        <Link pageId="Autor">
          <Parameter tableId="Autor" fieldId="idAutor"/>
        </Link>
      </QueryField>
      <QueryField tableId="Autor" fieldId="profissao"/>
      <QueryField tableId="Autor" fieldId="regiaoPais"/>
      <QueryField tableId="Autor" fieldId="apresentacao"/>
      <QueryField tableId="TemasAutor" fieldId="idTema"/>
      <QueryField tableId="TemasAutor" fieldId="nomeTema">
        <Link pageId="FotosDeTema">
          <Parameter tableId="TemasAutor" fieldId="idTema"/>
        </Link>
      </QueryField>
    </Query>
  </Content>
  <MenuLink pageId="TodasDeAutor"
    descriptiveName="Ver todas as fotos do autor
                     sem respeitar divisão por temas">
    <Parameter tableId="Autor" fieldId="idAutor"/>
  </MenuLink>
```

```
    <MenuLink pageId="FotosSeleccionadas"
              descriptiveName="A selecção do próprio autor">
      <Parameter tableId="Autor" fieldId="idAutor"/>
    </MenuLink>
  </Page>
```

A página gerada (figura 8) permitia obter os mesmos dados da base de dados, à excepção do número de fotos existentes em cada tema. Esta funcionalidade implica uma consulta com contagem de registos que não era possível especificar em WASL. Na imagem podemos ver os dados do autor com identificador `2` e nome completo `Alvaro Pereira Gomes` que tem dois temas na sua galeria: o tema `3` (`Grafismos`) e o tema 4 (`Solitárias`).

Figura 8. Página `GaleriaAutor` gerada

Na página original, a hiperligação *Destacadas e vencedoras* permitia aceder à lista de imagens que obtiveram prémios nos concursos promovidos pelo site ou que foram alvo de destaque. Os concursos e destaques não foram especificados uma vez que tornariam a dimensão deste *case study* demasiado extensa.

Em termos globais pode-se concluir que foi possível especificar e gerar uma boa parte das funcionalidades de consulta e navegação nos dados. No entanto, este *case study* permitiu identificar alguns problemas existentes no sistema WASL, nomeadamento a necessidade de permitir um maior detalhe na especificação de consultas, neste caso, permitindo a contagem de registos.

5 Conclusão

Os resultados obtidos da especificação de três aplicações para WWW existentes foram suficientemente esclarecedores para permitir concluir que a linguagem criada permite a especificação dos componentes de uma aplicação para WWW que interagem com a base de dados relacional que a suporta. Através da WASL e seu sistema de processamento foi possível gerar o código necessário, quer à consulta de dados existentes na base de dados, quer à sua actualização. Paralelamente, foi também possível especificar e gerar mecanismos de navegação contextual entre esses dados.

Por outro lado, permitiram identificar algumas lacunas no sistema desenvolvido e, consequentemente, delinear algumas evoluções possíveis a serem desenvolvidas em trabalhos futuros.

Por um lado, melhorias e correcções ao actual sistema desenvolvido, de forma a permitir a especificação de um maior conjunto de situações. Esta vertente abrange quer a própria linguagem, incluindo novos elementos ou redefinindo alguns aspectos da sua sintaxe, quer o seu sistema de processamento e código gerado. Deste conjunto de desenvolvimentos futuros, destaca-se:

– evitar alguma redundância na especificação dos parâmetros de uma página, permitindo que não fosse necessária a declaração do parâmetro dentro de um `Link` ou `MenuLink`, cabendo ao sistema de processamento, analisar a página destino da hiperligação e gerar o código necessário à passagem desse parâmetro para essa página;
– permitir a especificação de várias operações sobre a base de dados numa mesma página (e.g. várias consultas, consulta e inserção)
– enriquecer o elemento `Query`, adicionando alguns sub-elementos que permitissem especificar, por exemplo, critérios específicos para um ou mais campos.

Um dos aspectos salientado pela especificação e geração de uma aplicação existente de vendas on-line é que alguma da interacção da aplicação com a base de dados está directamente relacionada com os mecanismos de interacção com o utilizador. Nessa aplicação, o utilizador vai seleccionando as diversas opções e apenas no fim do processo, depois de todas as validações efectuadas, é que é actualizada a base de dados. As diversas opções que o utilizador ia seleccionando eram guardadas em variáveis de sessão e apenas no fim eram processadas e inseridas na base de dados.

Neste caso, existe uma relação directa entre o mecanismo de interacção entre o utilizador e a aplicação e a forma como são efectuadas as operações na base de dados. Assim, para se poder definir correctamente este tipo de operações, ter-se-á de estudar estes tipos de interface com o utilizador.

Outra vertente das aplicações para WWW que está directamente ligada aos dados existentes na base de dados e às operações que são efectuadas sobre esses dados, é a personalização de conteúdos.

A definição de uma linguagem que permita especificar este tipo de mecanismos de interacção com o utilizador e a sua relação com a base de dados, permitirá alargar a capacidade de especificação da linguagem proposta neste projecto e assim permitir a especificação de um maior número de aplicações.

Referências

[AMM97a] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Design and maintenance of data-intensive web sites. Technical Report 25, Dipartimento di Informatica e Automazione, Università di Roma Tre, June 1997. Também em Proc. of 6th International Conference on Extending Database Technology, EDBT'98, Valencia, Spain, March, 1998.

[AMM97b] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To weave the web. In *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB'97, Athens, Greece*, pages 206–215. Morgan Kaufmann, August 1997.

[BCFM00] Aldo Bonglio, Stefano Ceri, Piero Fraternali, and Andrea Maurino. Modeling data entry and operations in webml. In *Proceedings of WebDB 2000, Dallas, USA*, pages 201–214, 2000.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (webml): a modeling language for designing web sites. In *Proceedings of WWW9, Amsterdam*, May 2000.

[CFP99] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. Data-driven one-to-one web site generation for data-intensive applications. In *Proceedings of Conference on Very Large Data Bases, VLDB'99, Edinburgh, Scotland, UK*, pages 615–626. Morgan Kaufmann, September 1999.

[FFKL98] Mary Fernández, Daniela Florescu, Jaewoo Kang, and Alon Levy. Catching the boat with strudel: Experiences with a web-site management system. In *Proceedings of ACM SIGMOD Conference on Management of Data, Seattle, USA*, 1998.

[FLM98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Databse techniques for the world wide web: A survey. *SIGMOD Record*, 27(3):59–74, September 1998.

[Fou00] Apache Software Foundation. Dbtags taglib documentation. http://jakarta.apache.org/taglibs/doc/dbtags-doc/index.html, 2000.

[Fra99] Piero Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Computing Surveys*, 31(3):227–263, September 1999.

[Mar02] Alexandre M. P. Martins. Especificação xml de aplicações para www. Master's thesis, Faculdade de Engenharia - Universidade do Porto, 2002.

[MMAC99] Giansalvatore Mecca, Paolo Merialdo, Paolo Atzeni, and Valter Crescenzi. The (short) araneus guide to web-site development. In *Informal Proceedings of ACM SIGMOD Workshop on The Web and Databases (WebDB'99), Philadelphia, USA*, June 1999.

Xexam - uma linguagem de suporte para exames online (e-learning)

Daniel P. Soares
deps@mail.pt
DI/UM

M. Conceição Mota
conceicao_mota@aeiou.pt
DI/UM

José Carlos Ramalho
jcr@di.uminho.pt
DI/UM

10 de Fevereiro de 2003

Resumo

Hoje o E-learning, para além de ser um chavão utilizado por muitas instituições, também tem sido usado como estandarte de muitas instituições de ensino que se dizem na vanguarda.

Poderíamos começar a discutir o que é o E-Learning mas isso seria assunto para uma tese. Para o nosso trabalho, vamos considerar a definição mais consensual de ensino à distância onde se utiliza a Internet como veículo primordial para a comunicação entre professor e aluno e entre alunos. Nessa perspectiva, uma plataforma de E-Learning tem vários componentes: gestão de alunos e cursos, publicação de conteúdos, fóruns de discussão, chats, controle de acessos, diversas estatísticas e avaliação. É precisamente neste último componente que incide o trabalho descrito neste artigo. A avaliação online vai levantar sempre problemas de creditação mas discutir esse assunto não é o objectivo deste artigo.

Neste trabalho, apresenta-se uma linguagem XML — Xexam, que pode servir de suporte à criação de exames (por exemplo para auto-avaliação do aluno). Além da linguagem apresentam-se algumas ferramentas de suporte que fornecem algumas funcionalidades ao autor do exame: publicação em papel, publicação na Internet, correcção automática ao aluno que desenvolver o exame online e publicação do exame com correcção.

No fim, tecem-se algumas considerações sobre a integração desta aplicação numa metodologia de E-Learning.

1 Introdução

Actualmente, em Portugal, assiste-se a uma competição crescente entre as mais variadas instituições de ensino. Nesta competição, as instituições tentam destacar-se umas das outras pela qualidade. Esta qualidade tem-se vindo a traduzir, essencialmente, em dois aspectos: conteúdos e acessibilidade desses conteúdos.

Neste contexto, o modelo do E-Learning, já adoptado na Europa e América do Norte, está a sofrer uma grande adesão. Esta grande adesão está relacionada com alguns factores: a captação de alunos além das barreiras geográficas, a diminuição de custos ao nível da manutenção e suporte das instalações físicas e um maior controle, por parte do professor, das acções dos alunos.

No Departamento de Informática da Universidade do Minho, ao longo dos últimos oito anos, têm vindo a ser utilizadas algumas das metodologias de E-Learning (essencialmente, boas práticas tecnológicas): disponibilzação de conteúdos, afixação de notas e sumários, grupos de discussão e, iniciativas do tipo "o problema semanal"[RH95].

Nos últimos tempos, tem-se feito um grande esforço na automatização de algumas destas metodologias como, por exemplo, a produção de conteúdos.

Neste artigo, descreve-se um primeiro protótipo para uma possível automatização do módulo de avaliação.

O artigo está organizado num série de secções. Nas primeiras, descreve-se o conceito de avaliação no contexto apresentado e justificam-se as tecnologias que se irão utilizar no protótipo. Nas secções seguintes, descrevem-se as ferramentas desenvolvidas e que, no seu todo, constituem a implementação do protótipo. No fim, tiram-se algumas conclusões do trabalho desenvolvido, tecem-se algumas comparações com as iniciativas internacionais de normalização nesta área e, traçam-se algumas linhas para trabalho futuro.

2 Exames (online, e não só ...

À partida, por exame ”online”, quer-se dizer exames que estão disponíveis na Internet. Neste ponto, pode-se, desde já, fazer uma divisão: exames estáticos (o aluno apenas pode ler e/ou imprimir o exame) e exames dinâmicos (com interacção: o aluno pode resolver o exame, enviar a sua resolução e obter um feed-back desta). Mais à frente, na discussão da implementação do protótipo, veremos que estas duas classes de exames têm implicações tecnológicas distintas.

Quer o exame seja estático ou dinâmico e independentemente da tecnologia e da maneira de como irá ser disponibilizado, pode ainda ser mais categorizado como pertencente a um determinado tipo. Antes de avançar é importante detalhar cada um destes tipos. Assim, e recorrendo a uma classificação consensual no seio das entidades percursoras do E-Learning e numa perspectiva estrutural, temos os seguintes tipos de exame:

Exame de escolha múltipla – este tipo de exame é composto por uma sequência de questões em que cada questão é, por sua vez, composta por uma sequência de alíneas das quais apenas uma contem a resposta certa e é esta que o aluno deve assinalar.

Exame de questões verdadeiro ou falso – este tipo de exame é composto por uma sequência de questões em que cada questão é, por sua vez, composta por uma sequência de alíneas; cada alínea pode ser verdadeira ou falsa e é este valor lógico que o aluno deve indicar.

Exame de questões de desenvolvimento – este tipo de exame é composto por uma sequência de questões em que cada questão tem um enunciado que apresenta um problema; o aluno deverá desenvolver cálculos ou discursos que lhe permitam dar uma solução ao problema.

Numa perspectiva funcional, podemos ter os seguintes tipos de exame:

Exame de uma tentativa – neste tipo de exame o aluno dispõe apenas de uma tentativa (é o tradicional).

Exame de várias tentativas – neste tipo de exame o aluno dispõe de várias tentativas; depois de submeter o exame e perante o feed-back o aluno tem a oportunidade de alterar algumas respostas e reenviar o exame.

Exame com tempo limite – neste tipo de exame o aluno tem que resolver o exame dentro de um determinado limite temporal; o sistema de suporte monitoriza o tempo e após o limite dado interrompe a resolução do exame.

Exame progressivo – este tipo de exame corresponde a uma exame faseado; ao aluno é apresentada uma parte do exame e perante o resultado obtido o aluno poderá continuar a resolução do mesmo ou não.

Exame com aleatoridade na ordem das questões – neste tipo de exame, o sistema de suporte, sempre que um determinado exame é solicitado, baralha a ordem das questões antes de apresentar o exame ao aluno.

É claro que um exame não precisa de pertencer estritamente a um destes tipos. Podemos ter exames que são dum tipo híbrido apresentando características de vários tipos. Por exemplo: um exame que combina questões de desenvolvimento com questões de resposta múltipla ou de verdadeiro e falso.

Neste trabalho, e tratando-se de um protótipo que visa testar a tecnologia envolvida (que será descrita nas secções seguintes) no contexto que temos vindo a descrever, havia que limitar os tipos possíveis de exame. Por isso, os tipos de exame que foram considerados na implementação foram os de múltipla escolha, os de desenvolvimento e os correspondentes às combinações destes dois.

Na secção seguinte, apresenta-se a metodologia desenvolvida para a produção de exames.

3 Xexam

Normalmente, as plataformas de E-Learning são completamente abertas quanto aos formatos em que aceitam o conteúdos. Consequentemente, estes podem ser textos em HTML, animações em JAVA, documentos em PDF ou apresentações em PowerPoint.

No nosso grupo de trabalho, temos vindo a produzir conteúdos em XML [RH02] sempre que tal se justifique. Para isso, avaliamos sempre muito bem qual o resultado final pretendido.

Para a selecção do formato em que iriam ser produzidos os exames houve que ter em conta o resultado final pretendido. À partida, pretendiam-se quatro tipo de resultados:

1. Uma versão do exame em papel, igual à que tradicionalmente se utiliza nos exames presenciais. Este resultado é essencial para garantir a compatibilidade com o passado, evitando a criação de rupturas com o que os docentes vêm fazendo desde sempre. Se um docente, que está a usar este módulo, quiser produzir um exame tradicional para entregar em papel aos alunos o sistema deve permiti-lo sem qualquer custo adicional para o docente.

2. Uma versão online sem interacção. Equivalente à anterior só que em HTML. Para os casos em que o docente quer usar o sistema para publicar electronicamente o exame, após o exame ter sido realizado presencialmente ou para exames de auto-avaliação do aluno.

3. Um versão online com interacção. Neste caso, o exame é apresentado ao aluno sob a forma de um formulário HTML que aquele deverá preencher e enviar. Após o envio o sistema dá algum feed-back ao aluno: cotação obtida nas perguntas que foi possível corrigir automaticamente, marcação das respostas erradas com a respectiva solução, ...

4. Uma versão em papel, que corresponde à versão que fica armazenada para o professor, de um exame realizado online. Nesta versão, o exame é impresso com as perguntas e as respostas dadas pelo aluno.

Como vemos, logo à partida, queremos que o sistema produza quatro resultados distintos para o mesmo conteúdo. Sempre que tal acontece, estão criadas as condições para a utiliação de uma linguagem neutra, de anotação descritiva, para a produção de conteúdos.

Neste momento, a meta-linguagem XML é a solução adoptada quando este panorama se verifica.

Em termos mais formais, pode-se definir o XML como uma linguagem de anotação descritiva extensível, tendo, portanto, todas as características que tornam desejáveis este tipo de linguagens: independência relativamente às plataformas de software e de hardware utilizadas, longevidade, baixos custos de manutenção, facilidade na reutilização, ...

Para mais detalhes sobre o XML aconselhamos a leitura da vasta bibliografia existente, por exemplo [Wil01, HM01].

Como dissemos, o XML é uma meta-linguagem, não fornece, portanto, nenhum conjunto de marcas pré-definido. Fornece sim, os meios necessários à criação desses conjuntos de marcas os quais são designados por linguagens ou dialectos XML.

No nosso caso, vamos definir uma linguagem que nos permita realizar a anotação descritiva de um exame.

3.1 Definição da linguagem de anotação

Para definir uma liguagem de anotação há que criar o DTD (”Document Type Definition”) ou o Schema para essa linguagem. Esta fase corresponde à especificação da gramática se quisermos traçar um paralelo com as linguagens de programação.

Optou-se por definir um Schema devido ao suporte de Namespaces e às possibilidades de desenvolvimento modular que estes têm face aos DTDs.

Como já referimos, os tipos de exame que foram considerados neste projecto foram os exames de múltipla escolha, os de desenvolvimento e os correspondentes às combinações destes dois.

Assim, o Schema para exames é constituído por três partes:

Cabeçalho – onde se encontra toda informação relativa à metainformação que deverá estar associada ao exame, isto é, nome da disciplina, cursos a que se destina, ano lectivo, professor, chamada, data, hora, duração, sala e observações.

Corpo – onde o docente irá colocar as questões, que são o cerne de um exame; existem duas alternativas para a definição do corpo que estão relacionadas com tipo de exame que se pretende criar: um exame com apenas questões de múltipla escolha ou, um exame com questões de desenvolvimento e questões de mútipla escolha.

Anexos – onde o docente poderá colocar formulários, código, gráficos, etc...

Na apresentação dos vários elementos, utilizam-se figuras uma vez que o Schema é demasiado extenso para ser incluído neste artigo.

A figura 1 apresenta esquematicamente o elemento raiz para os documento do tipo exame.

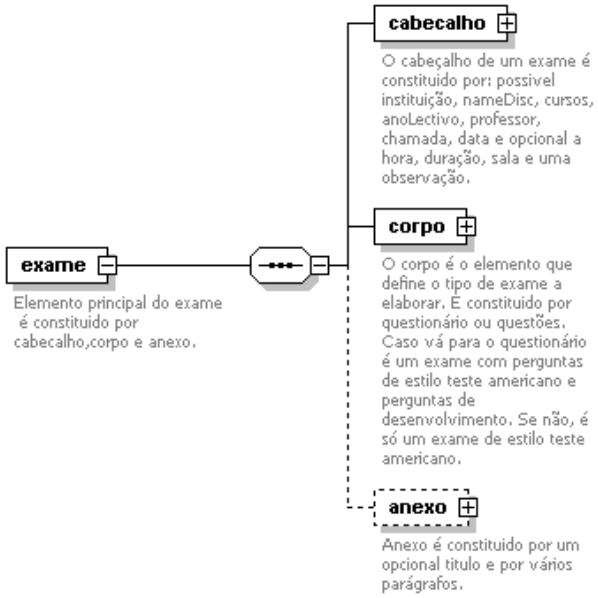


Figura 1: **Xexam: elemento raiz**

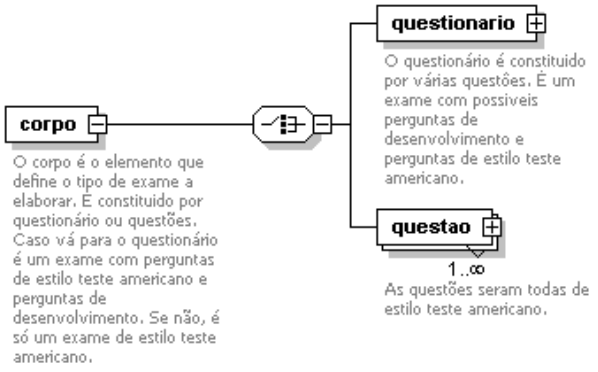


Figura 2: **Xexam: elemento corpo**

O corpo do exame é alternativamente composto por questões de múltipla escolha ou por um questionário em que estas poderão estar misturadas com questões de desenvolvimento (fig. 2).

A questão, por sua vez, é constituída por um enunciado e opcionalmente por um conjunto de alíneas (fig. 3).

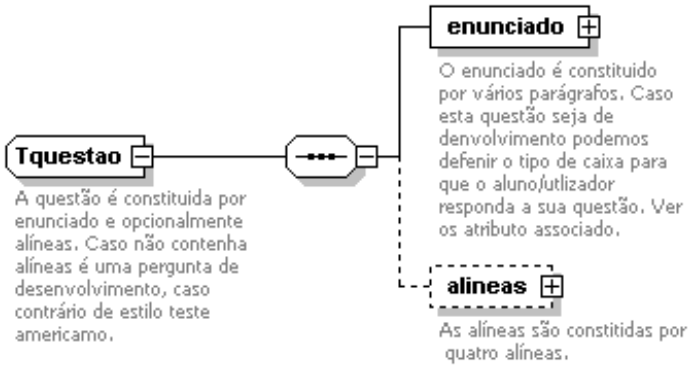


Figura 3: **Xexam: elemento questão**

Caso a questão seja de múltipla escolha tem que ter obrigatoriamente quatro alíneas em que cada alínea é constituída por um parágrafo (a resposta encerrada na alínea) e um atributo `validação` do tipo `boolean` onde o docente deverá assinalar a solução (fig. 4). As questões de desenvolvimento apenas têm um enunciado.

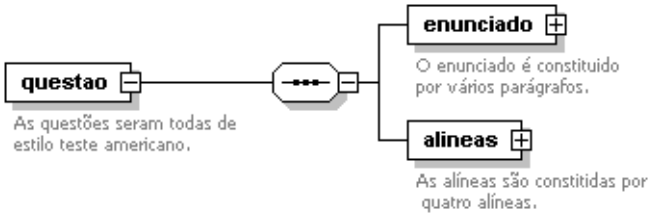


Figura 4: **Xexam: elemento questão**

Terminamos a discussão da linguagem XML apresentando o DTD (bem mais leve que o Schema), que ajudará a ilucidar o leitor no que respeita a elementos mais internos e aos atributos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT exame (cabecalho, corpo, anexo?)>
<!ATTLIST exame
  nomeExame CDATA #REQUIRED>

<!ELEMENT cabecalho (instituicao?, nameDisc, cursos, anoLectivo,
                     professor, chamada, data, hora?, duracao?,
                     sala?, observacao?)>

<!ELEMENT corpo (questionario | questao+)>

<!ELEMENT anexo (titulo?, para+)>
<!ATTLIST anexo
  numeroA ID #REQUIRED>

<!ELEMENT cursos (curso+)>

<!ELEMENT observacao (para+)>

```

```
<!ELEMENT questionario (questao+)>

<!ELEMENT para (#PCDATA | destaque | ref | xref | lista_seq
                | figura | codigo | tabela)*>

<!ELEMENT curso (nome, ano, codigoDisc?)>

<!ELEMENT questao (enunciado, alineas?)>
<!ATTLIST questao
  numeroQ ID #REQUIRED>

<!ELEMENT alineas (alinea+)>

<!ELEMENT destaque (#PCDATA)>
<!ATTLIST destaque
  tipo (carregado | sublinhado | italico) #IMPLIED>

<!ELEMENT ref EMPTY>
<!ATTLIST ref
  destino IDREF #REQUIRED
  label CDATA #IMPLIED>

<!ELEMENT xref (legenda, url)>

<!ELEMENT lista_seq (item+)>
<!ATTLIST lista_seq
  tipo (numerada | nnumerada) #IMPLIED>

<!ELEMENT figura (legenda, grafico)>

<!ELEMENT tabela (linhas+)>
<!ATTLIST tabela
  titulo CDATA #REQUIRED
  numCol CDATA #REQUIRED>

<!ELEMENT enunciado (para+)>
<!ATTLIST enunciado
  tbox (pequena | media | grande) #IMPLIED>

<!ELEMENT alinea (para+)>
<!ATTLIST alinea
  validacao (true | false) #REQUIRED>

<!ELEMENT item (para+)>

<!ELEMENT grafico EMPTY>
<!ATTLIST grafico
  formato (jpeg | gif | bmp | wmf | png) #IMPLIED
  path CDATA #REQUIRED>

<!ELEMENT linhas (coluna+)>

<!ELEMENT coluna (#PCDATA | para)*>
```

Na secção seguinte, apresenta-se um documento XML exemplo criado de acordo com o Schema definido.

3.2 Exames em XML

Para ilustrar melhor a linguagem desenvolvida apresenta-se agora um exemplo de exame.

Este exame corresponde a um exame real dado aos alunos da licenciatura em relações internacionais.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<exame
    nomeExame="Xexam2.xml"
    xsi:noNamespaceSchemaLocation="Xexam.xsd">
  <cabecalho>
    <instituicao>Universidade do Minho</instituicao>
    <nameDisc>Políticas Económicas da União Europeia</nameDisc>
    <cursos>
      <curso>
        <nome>Licenciatura em Relações Internacionais</nome>
        <ano>4ºAno</ano>
      </curso>
    </cursos>
    <anoLectivo>2001/2002</anoLectivo>
    <professor>Dra. Sandrina Soares</professor>
    <chamada>1ª Chamada</chamada>
    <data>24 Janeiro</data>
    <hora>9h30</hora>
    <duracao>2 horas</duracao>
    <sala>Sala 2209 e 2210</sala>
    <observacao>
      <para>Para cada questão selecione a afirmação verdadeira.</para>
    </observacao>
  </cabecalho>
...
```

Notas:

linha 1 – A linha 1 exibe a declaração XML indicando que conteúdo textual usa o alfabeto latino.

linha 4 – A linha 4 associa o Schema desenvolvido a este documento. Um validador usa esta informação para garantir que o exame especificado está de acordo com a estrutura previamente estabelecida.

linhas 5 a 24 – Nestas linhas, define-se a metainformação associada a este exame.

No bloco seguinte, apresenta-se a representação em XML de uma questão.

```
...
  <corpo>
    <questionario>
      <questao numeroQ="Q1">
        <enunciado>
          <para>No Tratado de Roma assinado em 1957 pelos seis
          Estados-membros de então, ficava expresa a intenção em realizar:</para>
        </enunciado>
        <alineas>
          <alinea validacao="false">
            <para>uma união aduaneira;</para>
          </alinea>
          <alinea validacao="false">
            <para>uma união aduaneira, um mercado comum e uma
            união económica;</para>
          </alinea>
          <alinea validacao="true">
            <para>uma união aduaneira e um mercado interno;</para>
          </alinea>
          <alinea validacao="false">
            <para>um mercado comum, uma união económica e uma união
```

```
            económica e monetária.</para>
          </alinea>
        </alineas>
      </questao>
      ...
    </questionario>
  </corpo>
</exame>
```

Na secção seguinte, veremos como obter os vários resultados a partir deste tipo de documentos XML.

4 Geração das versões em papel e em HTML

Uma vez criado o exame em XML, torna-se necessário transformá-lo noutro formato conforme a utilização final. No início do artigo, discutiram-se os formatos pretendidos. Estes podem ser agrupados em dois grupos: papel e HTML. Para o papel optou-se por gerar LaTeX e deste PDF.

Assim, para obtermos a versão do exame em papel precisamos de uma transformação de XML para LaTeX, e para obtermos a versão Web, uma transformação de XML para HTML.

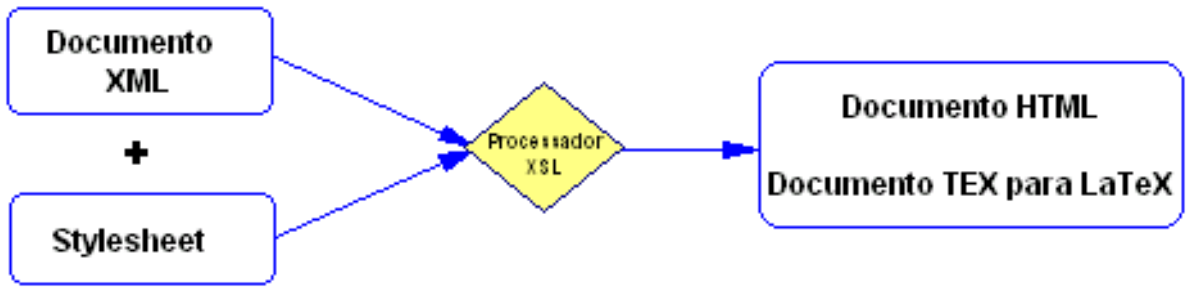


Figura 5: **Processo de Transformação**

De entre as várias ferramentas e correspondentes metodologias de transformação optou-se por utilizar XSL por várias razões: uma transformação é fácil de prototipar em XSL e, como uma folha de estilo XSL é um documento XML, pode ser transportada entre plataformas sem qualquer alteração. Assim, especificaram-se duas stylesheets que permitem a geração da versão HTML e da versão LaTeX a partir dos documentos XML já mencionados.

Dada a sua semelhança, apresenta-se a seguir, apenas a folha de estilo que transforma o exame numa página HTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output encoding="ISO-8859-1" method="html"
              indent="yes" version="1.0"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="cabecalho/nameDisc"/>
        </title>
      </head>
      <body style="MARGIN-LEFT: 10%; MARGIN-RIGHT: 10%;
                   FONT-FAMILY: 'Helvetica', 'Arial';">
```

```
18        <xsl:apply-templates/>
19      </body>
20    </html>
21  </xsl:template>
```

Notas:

linhas 9-21 – Template responsável por criar a estrutura da página HTML.

```
22  <xsl:template match="cabecalho">
23    <h4 align="center">
24      <xsl:if test="instituicao">
25        <xsl:value-of select="instituicao"/>
26      </xsl:if>
27      <br/>
28      <xsl:value-of select="nameDisc"/>
29      <br/>
30    </h4>
31    <font size="2">
32      <center>
33        <xsl:apply-templates select="cursos"/>
34        <xsl:value-of select="anoLectivo"/>
35        <br/>
36        <xsl:value-of select="professor"/>
37        <br/><br/>
38        <xsl:value-of select="chamada"/>
39        <xsl:if test="data">
40          <xsl:text> (</xsl:text>
41          <xsl:value-of select="data"/>
42          <xsl:text>)</xsl:text>
43        </xsl:if>
44        <br/>
45        <xsl:value-of select="hora"/>
46        ...
47    </font>
48    <br/>
49    <hr color="#b0c4de"/>
50    <br/>
51  </xsl:template>
```

Notas:

linhas 22-51 – Template que trata do cabeçalho; faz o a impressão estilizada da metainformação do exame; para os campos opcionais testa se estes existem antes de os processar.

```
52  <xsl:template match="corpo">
53    <xsl:choose>
54      <xsl:when test="//questionario">
55        <form action="http://localhost/cgi-bin/cgi2.cgi" method="post">
56          <fieldset style="BACKGROUND-COLOR: #f0f8ff">
57            <legend>Identificação do aluno:</legend>
58            <pre>
59              Nome:<input type="text" size="60" name="nome"/>
60              Curso:<input type="text" size="15" name="curso"/>
61              Numero:  <input type="text" size="15" name="numero"/>
62            </pre>
63          </fieldset>
64          <br/><br/><hr color="#b0c4de"/><br/>
65          <xsl:apply-templates/>
```

```
66          <p align="right">
67            <xsl:element name="input">
68              <xsl:attribute name="type">hidden</xsl:attribute>
69              <xsl:attribute name="name">exame</xsl:attribute>
70              <xsl:attribute name="value">
71                <xsl:value-of select="../@nomeExame"/>
72              </xsl:attribute>
73            </xsl:element>
74            <input type="reset" value="Limpar"/>
75            <xsl:text>  </xsl:text>
76            <input type="submit" value="Submit"/>
77          </p>
78        </form>
79      </xsl:when>
80      <xsl:otherwise>
81        <form action="http://localhost/cgi-bin/resultado.cgi" method="post">
82          <xsl:apply-templates/>
83          <p align="right">
84            <xsl:element name="input">
85              <xsl:attribute name="type">hidden</xsl:attribute>
86              <xsl:attribute name="name">exame</xsl:attribute>
87              <xsl:attribute name="value">
88                <xsl:value-of select="../@nomeExame"/>
89              </xsl:attribute>
90            </xsl:element>
91            <input type="reset" value="Limpar"/>
92            <xsl:text>  </xsl:text>
93            <input type="submit" value="Submit"/>
94          </p>
95        </form>
96      </xsl:otherwise>
97    </xsl:choose>
98  </xsl:template>
```

Notas:

linhas 52-98 – Template que gera o formulário HTML.

linhas 54-55 – Se o exame tiver perguntas de desenvolvimento, os dados fornecidos pelo aluno serão armazenados para posterior tratamento (na secção seguinte discutiremos a gestão da interacção através de CGIs – `cgi2.cgi`).

linhas 81-82 – Se o exame for composto apenas por perguntas de múltipla escolha, então é possível dar um feedback imediato ao aluno sobre o seu desempenho – `resultado.cgi`.

```
99  <xsl:template match="questao">
100   <table border="1" width="100%" bgcolor="#f0f8ff">
101     <tr><td><b>
102       <xsl:element name="a">
103         <xsl:attribute name="name">
104           <xsl:value-of select="@numeroQ"/>
105         </xsl:attribute>
106         <i>Questão <xsl:number/>
107           <xsl:text>.   </xsl:text>
108         </i>
109       </xsl:element>
110       </b>
111       <xsl:text>   </xsl:text>
112       <xsl:apply-templates select="enunciado"/>
```

```
      </td>
      </tr>
    </table>
    <xsl:apply-templates select="alineas"/>
    <xsl:if test="not(alineas)">
      <p>
        <font size="4">
          <i>Resposta:</i>
        </font>
      </p>
      <xsl:element name="textarea">
        <xsl:attribute name="name">
          Q<xsl:value-of select="count(preceding::questao)+1"/>
        </xsl:attribute>
        <xsl:attribute name="rows">7</xsl:attribute>
        <xsl:attribute name="cols">74</xsl:attribute>
      </xsl:element>
      <br/><br/>
    </xsl:if>
    <br/><hr color="#b0c4de"/><br/>
  </xsl:template>
```

Notas:

linhas 99-133 – Template que vai gerar os campos do formulário para cada questão.

linha 112 – Apresenta o enunciado da questão.

linha 116 – Manda processar as alíneas.

linhas 117-131 – Se a pergunta não tiver alíneas, então é uma pergunta de desenvolvimento; neste caso, é criada uma caixa para o aluno poder desenvolver a sua resposta.

```
  <xsl:template match="alinea">
    <tr>
      <td><li><xsl:apply-templates/></li></td>
      <td>
        <p align="right">
          <font size="-2">
            <xsl:text>V</xsl:text>
            <xsl:element name="input">
              <xsl:attribute name="type">radio</xsl:attribute>
              <xsl:attribute name="name">
                <xsl:value-of select="count(preceding::alinea)+1"/>
              </xsl:attribute>
              <xsl:attribute name="value">true</xsl:attribute>
            </xsl:element>
            <xsl:text>F</xsl:text>
            <xsl:element name="input">
              <xsl:attribute name="type">radio</xsl:attribute>
              <xsl:attribute name="name">
                <xsl:value-of select="count(preceding::alinea)+1"/>
              </xsl:attribute>
              <xsl:attribute name="value">false</xsl:attribute>
            </xsl:element>
          </font>
        </p>
      </td>
    </tr>
  </xsl:template>
```

```
161 ...
162 </xsl:stylesheet>
```

Notas:

linhas 134-160 – Nesta template, é tratada cada uma das alíneas; para cada alínea são criados dois botões com os valores `Verdadeiro` e `Falso` respectivamente.

A transformação para LaTeX é muito semelhante bastando trocar as anotações geradas por outras.

Após a transformação do documento XML para HTML do exame, vamos, na próxima secção, ver como é tratada a interacção respeitante à correcção do exame e do feed-back a dar ao aluno.

5 Correcção online

Um formulário HTML implica interacção, neste caso com o servidor do exame. Assim, do lado servidor, terá que haver um programa que receba os dados do aluno, os processe e envie uma resposta ao aluno.

A tecnologia necessária para a implementação deste mecanismo está há muito desenvolvida e dá pelo nome de CGI (”Common Gateway Interface”). Uma CGI pode ser implementada com uma linguagem de `Scripting`: Perl, Python, C, Java, ASP, ... No nosso caso, optamos pelo Perl porque vai ser necessário processar o documento XML do exame (para ir buscar as soluções das questões e cruzá-las com as respostas do aluno) e existe um módulo Perl de nome XML::DT para processar XML que tem vindo a ser desenvolvido no Departamento de Informática [RA99].

Recorrendo a esta tecnologia foi possível montar uma arquitectura funcional com a seguinte dinâmica:

1. o docente cria o exame em XML, recorrendo a um editor estruturado ou a um simples editor de texto, e submete-o ao sistema;
2. o sistema gera a versão HTML do exame que fica disponível num determinado URL;
3. o aluno acede ao URL do exame;
4. o servidor envia-lhe o formulário correspondente;
5. o aluno utiliza o formulário para responder ao exame e submete-o ao sistema (vai ser uma CGI que vai atender este pedido);
6. o servidor recebe o exame resolvido e encaminha-o para a respetiva CGI;
7. a CGI vai cruzar a informação recebida com as soluções deixadas pelo docente no exame, realiza os cálculos necessários e envia uma resposta;
8. o aluno recebe a resposta indicando a cotação obtida (para as questões de escolha múltipla) e a resposta correcta às questões que errou.

Na secção seguinte, apresentam-se com mais detalhe as CGIs desenvolvidas.

5.1 Elaboração das CGI’s do Xexam

Aquando da elaboração da folha de estilo para gerar a página HTML do exame, a template que transforma o nodo `corpo` continha as seguintes linhas:

```
1 ...
2   <form action="http://localhost/cgi-bin/cgi2.cgi" method="post">
3 ...
```

Repare no atributo `action` do elemento `form` que indica ao servidor qual a CGI a executar quando o pedido é enviado, e é de notar também o elemento `input` que permite definir parâmetros a passar ao servidor.

Após este pequeno extracto de código, vamos ver como elaborar a CGI’s para a correcção automática.

5.1.1 Correcção do exame

Apresenta-se a seguir um extracto da CGI que faz a correcção do exame.

```
1  ...
2  my %handler=(
3      -type => {  alineas => 'SEQ',
4                  questao => 'MAP',
5                exame => 'SEQ',
6                corpo => 'SEQ',
7                xref => 'SEQ',
8             },
9       '-outputenc' => 'ISO-8859-1',
10      '-default' => sub{$c},
11      'cabecalho' => sub{""},
12      'anexo' => sub{""},
13      'alinea' => sub{ $sol{++$k}=$v{validacao};
14                       +{val => $v{validacao}, texto => $c}},# remember $v{validacao}
15      'questao' => sub{+{n=>$v{numeroQ}, %$c }},# remember $v{numeroQ}
16      'lista_seq' => sub{lista_seq($c,$v)},# remember $v{tipo}
17      'tabela' => sub{tabela($c,$v)},# remember $v{titulo}
18      'item' => sub{li(dd,$c)},
19      'linha_cod' => sub{$c},
20      'grafico' => sub{grafico($v)},# remember $v{path}
21      'figura' => sub{figura($c)},
22      'xref' => sub{xref($c)},
23      'ref' => sub{ref_($v)},# remember $v{label},$v{destino}
24      'linhas' => sub{'<tr>'.$c.'</tr>'},
25      'coluna' => sub{td($c)},
26      'codigo' => sub{$c},
27      'destaque' => sub{destaque($c,$v)},# remember $v{tipo}
28 );
29 my $tree = dt($par{exame},%handler);
30 my $tre = @{$tree}[1]; # para trabalhar apenas o corpo
```

Notas:

linhas 2-28 – Nestas linhas define-se o array associativo `handler`; este array corresponde a uma associação entre o nome de um elemento e a respectiva função de transformação; neste caso, as funções de transformação calculam as estruturas de dados necessárias à correcção do exame.

linhas 29-30 – Realizam as travessias ao documento XML aplicando as transformções definidas acima.

```
31 ...
32 print header,
33 start_html( -title=>'teste',
34             -author=>'Sao@Edgar.pt',
35             "style='margin-left: 10%; margin-right: 10%'",
36           );
37 start_form(),
38 classifica(),
39 end_form(),
40 end_html();
```

Notas:

linhas 32-40 – Estas linhas estão a gerar a página de resposta; como se pode ver a função `classifica` é que vai calcular o resultado obtido.

```
sub classifica{
   my $erros = 0;      # para contar o nº de respostas erradas
   my $nresp = 0;      # para contar o nº de respostas não respondidas
   my $respC = 0;      # para contar o nº de respostas Correctas
   my $cotacao = 0;    # somar as cotações das respostas certas
   my $valAlinea = 20/$k; # valor a atribuir a cada alinea

    print br.br.
         "<table align='center'
           border='2'
         BORDERCOLOR ='b0c4de'
         width='80%'>
         <tr>
         <td BORDERCOLOR='#FFFFFF'>".br;

   for(keys %sol){
     if((($sol{$_} eq "true") && ($par{$_} eq "false"))
         or (($sol{$_} eq "false") && ($par{$_} eq "true")))
        {$erros++;
    }
    else {
       if(!defined($par{$_}))
        {$nresp++;}
       else{ $cotacao = $cotacao+$valAlinea;
             $respC++;}
    }
   }
   if(($cotacao-($valAlinea*$erros)) > 9.4){
      print h3({-align=>'center'},
               font({-color=>'green'},"Aprovado!!")),
        br,dd.dd.dd.font({-size=>'4'},i("Resultado: ")),
        font({-size=>'3'},
        ' ',$cotacao-$valAlinea*$erros," valores");
    }
   else {
      if(($cotacao-($valAlinea*$erros)) < 0){
        print h3({-align=>'center'},
             font({-color=>'red'},"Reprovado!!")),
              br,dd.dd.dd.font({-size=>'4'},i("Resultado: ")),
          font({-size=>'3'},' ',"0 valores");
          }
      else{ print h3({-align=>'center'},
                     font({-color=>'red'},"Reprovado!!")),
                   font({-size=>'4'},br,dd.dd.dd.i("Resultado: ")),
          font({-size=>'3'},
          $cotacao-$valAlinea*$erros,' '," valores");
          }
     }
  print br,font({-size=>'3'},
    dd.dd.dd.dd.$respC,' '," respostas correctas",br,
        dd.dd.dd.dd.$erros,' '," respostas erradas",br,
    dd.dd.dd.dd.$nresp,' '," respostas não respondidas"),br;

  print "</table></tr></td>";
  print br,br;
  mkform($tre,\%sol,\%par);
 }
 ...
```

Este último excerto de código mostra a função classifica que faz uma travessia às estruturas de dados criadas na travessia do documento XML do exame, e calcula o resultado final.

Este artigo termina com a próxima secção onde se tiram algumas conclusões e se faz uma síntese do que foi feito e do que falta fazer.

6 Conclusão

Ao longo deste artigo nunca foram feitas comparações com soluções já desenvolvidas. Isto deve-se ao facto de, na comunidade de E-Learning, não existirem standards, existem sim conjuntos de recomendações que as entidades poderão seguir ou não. Apesar disso, existem dois grandes grupos que integram, cada um, algumas centenas de instituições de ensino: SCORM e IMS.

No que à avaliação diz respeito, estes grupos publicaram um DTD de uma linguagem que, segundo eles, deverá ser utilizada na produção de exames. No entanto, a linguagem definida nesse DTD é bastante complexa (é o que normalmente acontece quando se produzem DTDs para grandes comunidades). Além disso, também não disponibilizam ferramentas para colocar o sistema a funcionar.

Foi por estas razões, que no contexto do projecto aqui retratado se decidiu desenvolver uma pequena linguagem para testar algumas ideias.

O protótipo desenvolvido demonstrou a sua viabilidade e utilidade. Num futuro próximo, estará a ser utilizado no apoio à auto-avaliação dos alunos.

Há, no entanto, muitas coisas a melhorar e a implementar. Tais como:

- Aceitar mais tipos de exame: com perguntas aleatórias, com questões de verdadeiro e falso, progressivos, com tempo limite, ...
- Adicionar um módulo para autenticação do aluno.
- Criar uma base de dados para registo das interacções com os alunos.
- Desenvolver um módulo para correcção automática para perguntas de desenvolvimento de cálculo ou de programação.

Referências

[HM01] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*. O'Reilly, 1ª edição edition, Janeiro 2001.

[RA99] J.C. Ramalho and J.J. Almeida. Xml::dt - a perl down translation module. In *XML Europe'99*, Granada - Espanha, Abril 1999.

[RH95] José Carlos Ramalho and Pedro Rangel Henriques. Uma experiência de utilização da internet na gestão pedagógica e sua formalização. In *Conferência Nacional WWW*. Universidade do Minho, 1995.

[RH02] José Carlos Ramalho and Pedro Rangel Henriques. *XML e XSL: da teoria à prática*. 2002.

[Wil01] Heather Williamson. *XML: the complete reference*. Osborne/McGraw-Hill, 2001.

XML Templates for Constraints (XTC), *Um nível de abstracção para Linguagens de Especificação de Restrições*

Marta H. Jacinto* José C. Ramalho
Pedro R. Henriques

Departamento de Informática
Universidade do Minho
4710-057 Braga
marta.jacinto@itij.mj.pt,{jcr, prh}@di.uminho.pt

Resumo

Os DTDs permitem etiquetar um texto e validar a sua estrutura contra uma gramática.
As linguagens de especificação de restrições (XML Constraint Specification Languages), nomeadamente o XCSL, o Schematron e os XML-Schemas, *num nível mais elevado*, já permitem validar aspectos não estruturais dos documentos XML, tais como: relações entre elementos, ou atributos, pertencentes a diferentes contextos; invariantes sobre modelos de dados; e restrições ao valor dos elementos, ou atributos.
O sistema XCSL (XML Constraint Specification Language) nasceu no seio do nosso grupo de investigação [7]. No entanto esta linguagem foi testada em pé de igualdade com Schematron e XML-Schema. Usou-se um conjunto considerável de casos de estudo para testar e comparar estas três linguagens em termos: dos tipos de restrições especificáveis; da facilidade de aprendizagem/utilização; da informação devolvida ao utilizador. Os resultados mais significativos foram descritos em [3].
Fazendo esta comparação, apercebemo-nos que em cada linguagem e para cada tipo de restrição há um texto fixo e um conjunto de partes variáveis, sendo este último comum às várias linguagens. Tendo em conta estas partes variáveis, criámos *templates* para cada tipo de restrição, em cada uma das três linguagens. Com estes *templates* é possível gerar a especificação de restrições, em qualquer uma daquelas linguagens, a partir de um conjunto finito de parâmetros.
Neste artigo mostramos os **templates** para cada par tipo-de-restrição

*Presentemente a trabalhar no ITIJ - Instituto das Tecnologias de Informação na Justiça, Ministério da Justiça

/ linguagem. A partir das partes comuns desses *templates* construímos um conjunto de **templates genéricos**, designado XTC—XML Templates for Constraints—, um para cada tipo de restrição independentemente da linguagem escolhida. Com um documento XTC pode gerar-se todos os ficheiros de especificação de restrições, ou seja, um ficheiro de especificação para cada linguagem. Apresentamos, então, vários exemplos escritos em XTC.
A implementação final usa aquilo a que chamamos *sistema de folhas de estilo XSL de terceira geração*, três níveis de folhas de estilos. Com a primeira folha de estilos (a do XTC) e o documento XTC geramos o documento de especificação na linguagem pretendida; com este último e a segunda folha de estilos (específica da linguagem pretendida) geramos a terceira folha de estilos (documento com o qual já se vai poder validar a semântica das várias instâncias); por fim, aplicamos esta última folha de estilos aos vários documentos da família em estudo.
Terminamos o artigo mostrando como construímos esta arquitectura baseada apenas em XML e XSL.

Palavras-chave:
XML, DTD, XML-Schema, Schematron, XCSL, validação semântica, XSL

1 Introdução

O SGML e o XML alteraram profundadamente a forma como as pessoas pensam e lidam com documentos. Estes standards permitiram que nascesse uma nova metodologia e modelo de processamento de documentos.

Tornaram ainda possível a independência entre contexto e estrutura, validação estrutural, longevidade dos documentos, independência tanto de hardware como de software, e muito mais. Contudo, esta validação estrutural nem sempre e suficiente para obter a qualidade final desejada para os documentos.

Recentemente, propostas como DCD[1], XDuce[2], XML-Schema[3], TREX[4] e RelaxNG[5], foram submetidas ao W3C numa tentativa de resolver o problema. Daquelas, XML-Schema e RelaxNG têm sido as mais utilizadas pela comunidade XML.

No entanto, nenhuma destas duas linguagens é suficiente para atingir o tipo de validação que tratamos neste artigo. Para isso é necessária uma extensão, uma linguagem (e respectiva ferramenta) que permita especificar (e validar) restrições sobre documentos XML.

Por forma a formatar um documento, é necessário seleccionar fragmentos e aplicar-lhes um estilo ou, se quisermos questionar o conteúdo do documento, é necessário seleccionar os nodos desejados da árvore documental. Para isto temos o XPath — a linguagem de padrões inserida no XSLT, poderosa o suficiente para

[1] www.w3.org/TR/NOTE-dcd
[2] http://xduce.sourceforge.net/
[3] http://www.w3.org/XML/Schema
[4] http://thaiopensource.com/trex/
[5] http://www.oasis-open.org/committees/relax-ng/

seleccionar qualquer contexto dentro da árvore documental.

Começamos por descrever uma arquitectura operacional simples que permite aos utilizadores especificar restrições e testá-las dentro de um sistema XML-XSL (secção 2).

Depois, na secção 3, apresentamos as três linguagens de especificação de restrições que analisámos: XCSL, Schematron e XML-Schema, numa breve abordagem.

Na secção 4 mostramos os **templates** para cada par tipo-de-restrição / linguagem. A partir das partes comuns desses *templates* construímos um conjunto de **templates genéricos** (secção 5), que designámos por XTC—XML Templates for Constraints—, um para cada tipo de restrição independentemente da linguagem escolhida. Na secção 6 apresentamos a linguagem XTC e exemplos escritos nessa linguagem.

A implementação final usa aquilo a que chamamos *sistema de folhas de estilo XSL de terceira geração*: três níveis de folhas de estilos. Com a primeira folha de estilos (a do XTC) e o documento XTC, geramos o documento de especificação na linguagem pretendida; com este último e a segunda folha de estilos (específica da linguagem pretendida) geramos a terceira folha de estilos (documento com o qual já se vai poder validar a semântica das várias instâncias); por fim, aplicamos esta última folha de estilos aos vários documentos da família em estudo. Este processo encontra-se descrito na secção 7.

A secção 7 mostra ainda como construímos esta arquitectura baseada apenas em XML e XSL.

Por fim, generalizamos a ideia, especificando um método para implementar o modelo de processamento de restrições com qualquer ferramenta que suporte XSLT.

A conclusão pode ser encontrada na secção 8.

2 Restringindo o conteúdo

Em [6], foi estabelecido um paralelismo entre o Processamento de Linguagens Formais e o Processamento de Documentos.

Esta hipótese ajuda a perceber que o processamento de documentos actual é muito semelhante ao dos primeiros tempos da construção de compiladores.

Até há poucos anos, tinhamos apenas análise léxica e sintáctica. Os DTDs permitiram especificar o léxico e a gramática (quase abstracta) de uma determinada linguagem de anotação. Depois, em 1996, o DSSSL apareceu permitindo a especificação do que denominamos por semântica dinâmica, i.e., com DSSSL podemos especificar transformações sobre a estrutura produzida pelo *parser* quando este analisa o documento.

No entanto, este avanço não colmatou uma outra falha: a semântica estática.

Por Semântica Estática entendemos um conjunto de regras que impõem restrições sobre o conteúdo específico de cada documento, limitando o valor ab-

soluto ou relativo que certos elementos (ou componentes) podem ter. Para descrever essas regras sobre os valores admissíveis, é necessária uma linguagem que permita escrever condições sobre o conteúdo exacto das várias componentes estruturais, condições essas que são tipicamente dependentes do contexto preciso onde o elemento aparece. Para conceber uma dessas linguagens, a XCSL, Ramalho sentiu necessidade de classificar os tipos de restrições existentes [7]. Entretanto esta classificação foi refinada ([3, 4]). Relativamente à restrição de conteúdo, podemos assim classificar as condições contextuais a impor de acordo com as categorias seguintes:

Restrição de domínio Impõe que um determinado conteúdo esteja entre dois valores, isto é, num determinado domínio. Este é o tipo mais comum de restrição e, normalmente, aplica-se a dados numéricos ou do tipo data. Exemplo: a idade e a altura têm que ser valores positivos.

Dependência entre dois elementos ou atributos Permite relacionar o valor de um atributo ou elemento com o de outro atributo ou elemento localizado num ramo diferente da árvore documental (restrições dependentes do contexto) do qual ele depende. Exemplo: nome e verbo devem concordar em número.

Concordância com padrões descritos por Expressões Regulares Permite garantir que o conteúdo segue um determinado formato que pode ser padronizado por uma ER. Exemplo: Os números de telefone em Portugal têm nove dígitos e começam por 2, 7, 8, 91, 93 ou 96.

Restrições complexas Este grupo reúne todas as restrições que não podem pertencer a nenhum dos três grupos anteriores. Podem dividir-se em três casos (os dois primeiros são os casos particulares mais vulgares):

- **Restrições de conteúdo misto** Quando, por exemplo num elemento de conteúdo misto, se quer garantir determinada cardinalidade e/ou ordem dos elementos que podem aparecer no meio do texto. Exemplo: O fecho de um pedido de Certidão Fiscal é composto por texto livre, no qual têm que aparecer os sub-elementos *local* e *data*, uma vez cada e exactamente por esta ordem.
- **Restrições de unicidade** Quando se quer garantir que cada ocorrência de algum elemento, em determinado contexto, tem um valor diferente do de todas as outras ocorrências desse elemento nesse mesmo contexto. Exemplo: Não pode haver valores repetidos nos elementos chave de uma determinada tabela.
- **Restrições mistas** Sempre que uma restrição não pertence a nenhuma das categorias já enumeradas, ou resulta da mistura de algumas dessas categorias. Exemplo: Um poema deve ser constituído por duas quadras e dois tercetos (exactamente por esta ordem), mas apenas se for do tipo "soneto".

3 Linguagens de Especificação de Restrições

Nesta secção apresentam-se as três linguagens de especificação de restrições estudadas.

3.1 XCSL

O XCSL (XML Constraint Specification Language) é uma linguagem que foi desenhada e implementada por José Carlos Ramalho na Universidade do Minho, Portugal. Combina capacidades poderosas de validação com uma sintaxe e implementação simples. O XCSL foi definido formalmente em [7] e [5] e noutros artigos publicados. O DTD e XML-Schema que o definem podem ser encontrados em http://www.di.uminho.pt/~jcr/PROJS/xcsl-www/.

Para validar instâncias XML utilizando XCSL, é necessário executar dois passos. O primeiro consiste em compilar o documento de restrições com um gerador que vai produzir um validador específico em XSL. O segundo consiste em validar cada instância contra esse validador específico da família de documentos. O utilizador tem que efectuar os dois passos explicitamente invocando uma ferramenta como o Saxon[6] na linha de comandos.

3.2 Schematron

O Schematron é uma linguagem XML desenhada e implementada por Rick Jelliffe na Academia Sinica Computing Centre, Taiwan. Combina capacidades poderosas de validação com uma sintaxe e implementação simples. O Schematron está descrito em [1] e muitos outros artigos e foi recentemente submetido para standard ISO como ISO Schematron. O DTD e XML-Schema que o definem podem ser encontrados em http://www.ascc.net/xml/schematron/.

Toda esta abordagem (linguagem e processamento) é muito semelhante à anterior e por isso, para validar instâncias XML utilizando Schematron, também é necessário executar dois passos. O primeiro consiste em compilar o documento de restrições de modo a gerar um validador apropriado. O segundo consiste em validar cada instância contra esse validador específico da família de documentos. Para a execução destes dois passos, pode escolher-se uma de duas opções: Topologi Schematron Validator é a primeira — o processo de dois passos é transparente para o utilizador e tudo o que este tem que fazer é fornecer o documento com as restrições e as instâncias; e o Schematron-report — o utilizador tem que efectuar os dois passos explicitamente. Enquanto que este último, Schematron-report, é processado invocando uma ferramenta como o Saxon na linha de comandos; o Topologi Schematron Validator é um ambiente interactivo baseado em janelas.

[6] http://users.iclway.co.uk/mhkay/saxon

3.3 XML-Schema

O XML-Schema [2] do W3C foi criado uma vez que a sintaxe dos DTDs não satisfazia as necessidades dos utilizadores de XML. Quando se usa um DTD e se pretende especificar restrições semânticas mesmo que muito simples, é necessário usar uma linguagem de especificação de restrições (como o XCSL ou o Schematron) e, consequentemente, são necessários dois instrumentos para validar completamente um documento XML. Por outro lado, quando se usa antes um XML-Schema, será necessário, para um conjunto específico de restrições, apenas um documento para conseguir validar completamente as instâncias XML em vez de dois.

Há vários *parsers* disponíveis para validar as instâncias XML contra um XML-Schema. Esses *parsers* são semelhantes aos validadores de XML tradicionais mas agora, em vez do DTD, utilizam o XML-Schema.

4 Templates

Nesta secção mostramos os **templates** para cada par tipo-de-restrição / linguagem [4], organizado por linguagem.

4.1 XCSL

4.1.1 Restrição de domínio

```
<constraint>
 <selector selexp="path to the element"/>
 <cc>. | @attname relop value</cc>
 <action>
   <message>Message...
     <value selexp="path to any
       element/attribute | any expression
       applied to any element/attribute"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. Em *cc* especifica-se a restrição propriamente dita: o elemento (.) ou um atributo (@attname), o operador lógico e o valor (*value*) ou valores que limitam o domínio. A mensagem a ser mostrada sempre que a restrição não se verifica é descrita usando o elemento *message* onde se pode escrever texto livre (*Message...*) e/ou o valor de qualquer elemento/atributo no documento (*path to any element/attribute*) ou qualquer expressão a ele aplicada (*any expression applied to any element/attribute*).

4.1.2 Dependência entre dois elementos ou atributos

```
<constraint>
 <selector selexp="path to the 1st element"/>
 <cc>. | @attname relop path to the 2nd element
    [/@attname]
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição propriamente dita (*cc*) é agora: o elemento (.) ou um atributo (@attname), o operador lógico e o caminho para o segundo elemento/atributo (*path to the 2nd element*). A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.3 Concordância com padrões descritos por Expressões Regulares

```
<constraint>
 <selector selexp="path to the element"/>
 <cc>
   substring(.| @attname,i,n1)
   =literal_value and
   (string-length(number(substring(.| @attname,j,n2)))
   = value
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

Novamente, o caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição (*cc*) tem um determinado número de expressões (que dizem respeito a um elemento ou atributo) ligadas pelo operador lógico "e" (*and*); estas expressões podem ser:

– para uma parte fixa — *substring(. | @attname,i,n1) = literal_value*
(por exemplo se os primeiros quatro caracteres do elemento seleccionado tiverem que ser '253-', escreve-se *substring(.,1,4) = '253-'*);

– ou para um conjunto contíguo de dígitos — *string-length(number(substring(. | @attname,j,n2))) = value*
(por exemplo se os 6 caracteres do atributo x do elemento seleccionado que começam na posição 5 tiverem que ser dígitos, escreve-se *string-length(number(substring(@x,5,6))) = 6*).

A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.4 Restrições complexas - conteúdo misto

```
<constraint>
 <selector selexp="path to the parent element"/>
 <cc>
   (count(elt1)=c_elt1) and (count(elt2)=c_elt2)
   and ... (count(eltn)=c_eltn) and
   name(elo1[1]/following::*)='elo2' and
   name(elo2[1|2]/following::*)='elo3' and
   ... and
   name(elo(m-1)[1|2|...|m-1]/following::*)='elom'
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O elemento *selector* serve agora para seleccionar o caminho para o elemento pai (*path to the parent element*). A restrição (*cc*) tem duas partes ligadas pelo operador lógico "e" (*and*):

– a cardinalidade dos sub-elementos, constituída por várias expressões *(count(elti)= c_elti)* ligadas por "e" (*and*)
(por exemplo se o elemento *lugar* tiver que aparecer três vezes no elemento pai, escreve-se *(count(lugar)=3)*);

– a ordem dos sub-elementos, constituída por várias expressões *name(elo(j-1)[1|2|...|j-1]/ following::*)='eloj'*[7] (deve indicar-se a ordem exacta pela qual os sub-elementos devem ocorrer — elo1, elo2, ...) também ligadas

[7] [1|2|...|j-1], significa que a ocorrência do sub-elemento que se está a testar pode ser a 1^a, 2^a, ...,(j-1)-ésima ocorrência desse sub-elemento no elemento pai.

pelo operador "e" (*and*)
(por exemplo se a terceira ocorrência do elemento *lugar* tiver que ser seguida de uma ocorrência do elemento *estado*, escreve-se *name(lugar[3]/ following::*)='estado'*).

A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.5 Restrições complexas - unicidade

```
<constraint>
 <selector selexp="path to X branch"/>
 <let name="nameKey1"
      value="elementX | @attributeX"/>
 <cc>(count(path to Y branch[elementY
     | @attributeY = $nameKey1]) = 1)
 </cc>
 <action>
   <message>
     Message...
     element | @attribute:
     <value selexp="$nameKey1"/>.
   </message>
 </action>
</constraint>
```

Para este tipo de restrição, suponha-se que se quer forçar todos os valores de determinado elemento/atributo (element|@attribute) que ocorram no ramo X a existir também no ramo Y (uma vez). Primeiro, selecciona-se o caminho para o ramo X (*path to X branch*) no elemento *selector*. Depois, usa-se o elemento *let* para guardar numa lista todos os valores que esse elemento/atributo assume naquele ramo X (*elementX | @attributeX*). O elemento *cc* serve agora para contar o número de vezes que cada ocorrência do elemento/atributo no ramo Y ocorre na lista previamente definida e comparar esse valor com 1. O elemento *message* pode ter texto livre (Message... element|@attribute:) e/ou o valor que não aparece no ramo Y ($nameKey1).

4.1.6 Outras Restrições complexas

Não é possível escrever *templates* para este tipo de restrições uma vez que, virtualmente, podem ser qualquer coisa, o que é impossível de normalizar.

Para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, tem-se:

```
<constraint>
 <selector selexp="path to the element"/>
```

```
 <cc>
   not (path/@attname | . | @attname = value) or
   (constraint)
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição (*cc*) é agora: o operador lógico "não" (*not*) seguido pela especificação do valor que certo elemento/atributo, naquele ou noutro ramo, toma (o valor único para o qual a restrição vai ser tida em linha de conta), seguida pelo operador "ou" (*or*) e, finalmente, a restrição. A mensagem tem as mesmas características que a referida em 4.1.1.

4.2 Schematron

Nesta sub-secção mostram-se os *templates* para a linguagem Schematron.

4.2.1 Restrição de domínio

```
<pattern name="pattern title">
 <rule context="path to the element">
 <assert test=" . | @attname relop value" diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any
     element/attribute | any expression applied
     to any element/attribute"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. No atributo *test* do elemento *assert* especifica-se a restrição propriamente dita: o elemento (.) ou um atributo (@attname), o operador lógico e o valor (*value*) ou valores que limitam o domínio. A mensagem, a ser mostrada sempre que a restrição não se verifica, é descrita usando o elemento *diagnostic* onde se pode escrever texto livre (Message...) e/ou o valor de qualquer

elemento/atributo no documento (*path to any element/attribute*) ou qualquer expressão a ele aplicada (*any expression applied to any element/attribute*).

4.2.2 Dependência entre dois elementos ou atributos

```
<pattern name="pattern title">
 <rule context="path to the 1st element">
   <assert test=" . | @attname relop path to the 2nd element
               [/@attname] " diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição propriamente dita (*test*) é agora: o elemento (.) ou um atributo (@attname), o operador lógico e o caminho para o segundo elemento/atributo (*path to the 2nd element*). A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.3 Concordância com padrões descritos por Expressões Regulares

```
<pattern name="pattern title">
 <rule context="path to the element">
   <assert test="
     substring(.| @attname,i,n1)=literal_value and
     (string-length(number(
     substring(. | @attname,j,n2))) = value"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

Novamente, o caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição (*test*) tem um determinado número de

expressões (que dizem respeito a um elemento ou atributo) ligadas pelo operador lógico "e" (*and*); estas expressões podem ser:

- para uma parte fixa — *substring(. | @attname,i,n1) = literal_value*
  (por exemplo se os primeiros quatro caracteres do elemento seleccionado tiverem que ser '253-', escreve-se *substring(.,1,4) = '253-'*);
- ou para um conjunto contíguo de dígitos — *string-length(number(substring(. | @attname,j,n2))) = value*
  (por exemplo se os 6 caracteres do atributo x do elemento seleccionado que começam na posição 5 tiverem que ser dígitos, escreve-se *string-length(number(substring(@x,5,6))) = 6*).

A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.4 Restrições complexas - conteúdo misto

```
<pattern name="pattern title">
 <rule context="path to the parent element">
   <assert test="(count(elt1)=c_elt1) and
     (count(elt2)=c_elt2) and ...
     (count(eltn)=c_eltn) and
     name(elo1[1]/following::*)='elo2' and
     name(elo2[1|2]/following::*)='elo3' and
     ... and
     name(elo(m-1)[1|2|...|m-1]/following::*)='elom'"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O atributo *context* serve agora para seleccionar o caminho para o elemento pai (*path to the parent element*). A restrição (*test*) tem duas partes ligadas pelo operador lógico "e" (*and*):

- a cardinalidade dos sub-elementos, constituída por várias expressões *(count(elti)= c_elti)* ligadas por "e" (*and*)
  (por exemplo se o elemento *lugar* tiver que aparecer três vezes no elemento pai, escreve-se *(count(lugar)=3)*);

– a ordem dos sub-elementos, constituída por várias expressões *name(elo(j-1)[1|2|...|j-1]/ following::*)='eloj'*[8] (deve indicar-se a ordem exacta pela qual os sub-elementos devem ocorrer — elo1, elo2, ...) também ligadas pelo operador "e" (*and*)
(por exemplo se a terceira ocorrência do elemento *lugar* tiver que ser seguida de uma ocorrência do elemento *estado*, escreve-se *name(lugar[3]/ following::*)='estado'*).

A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.5 Restrições complexas - unicidade

```
<pattern abstract="true" id="uID">
 <rule context="path to Y branch">
   <key name="nameKey1"
        path="elementY | @attributeY"/>
 </rule>
</pattern>
<pattern name="pattern title">
 <rule context="path to X branch">
   <assert test="count(key('nameKey1',elementX
                 | @attributeX ) = 1)"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   element | @attribute:
   <value-of select="elementX | @attributeX"/>
 </diagnostic>
</diagnostics>
```

Para este tipo de restrição, suponha-se que se quer forçar todos os valores de determinado elemento/atributo (element|@attribute) que ocorram no ramo X a existir também no ramo Y (uma vez). Primeiro selecciona-se o caminho para o ramo Y (*path to Y branch*) no atributo *context* de *rule* descendente de um elemento *pattern*. Depois usa-se o elemento *key* (descendente do referido elemento *rule*) para guardar numa lista todos os valores que esse elemento/atributo assume naquele ramo Y (*elementY | @attributeY*). O atributo *context* usual (atributo de um elemento *rule* descendente de um novo elemento *pattern*) serve para seleccionar o caminho para o ramo X (*path to X branch*) e o elemento *assert* serve agora para contar quantas vezes cada ocorrência da lista previamente

[8] [1|2|...|j-1], significa que a ocorrência do sub-elemento que se está a testar pode ser a 1^a, 2^a, ...,(j-1)-ésima ocorrência desse sub-elemento no elemento pai.

definida ocorre no ramo X e comparar esse valor com 1. O elemento *diagnostic* pode ter texto livre (Message... element|@attribute:) e/ou o valor que não aparece no ramo Y (elementX | @attributeX).

4.2.6 Outras Restrições complexas

Não é possível escrever *templates* para este tipo de restrições uma vez que, virtualmente, podem ser qualquer coisa, o que é impossível de normalizar.

Para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, tem-se:

```
<pattern name="pattern title">
 <rule context="path to the element">
   <assert test="
       not (path/@attname | . | @attname = value)
       or (constraint)"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição (*test*) é agora: o operador lógico "não" (*not*), seguido pela especificação do valor que certo elemento/atributo, naquele ou noutro ramo, toma (o valor único para o qual a restrição vai ser tida em linha de conta), seguida pelo operador "ou" (*or*) e, finalmente, a restrição. A mensagem tem as mesmas características que a descrita em 4.2.1.

4.3 XML-Schema

Nesta secção mostram-se os *templates* para a linguagem XML-Schema.

4.3.1 Restrição de domínio

```
<xs:simpleType name="name of the type of the
                          element/attribute">
 <xs:restriction base="xs:integer">
   <xs:minInclusive value="value"/>
 </xs:restriction>
</xs:simpleType>
```

Cria-se um tipo simples para o elemento ou atributo em causa, por restrição, a partir de um conjunto conhecido.

4.3.2 Dependência entre dois elementos ou atributos

```
Não é possível especificar...
```

4.3.3 Concordância com padrões descritos por Expressões Regulares

```
<xs:simpleType name="telement">
  <xs:restriction base="xs:string">
    <xs:pattern value="literal_value\d{value}"/>
  </xs:restriction>
</xs:simpleType>
```

Cria-se, por restrição, um tipo simples para o elemento ou atributo em causa a partir do conjunto *xs:string*. Depois, especifica-se, no atributo *value*:

- para cada parte fixa — literal_value (por exemplo '253-');
- para cada conjunto contíguo de dígitos — \d{value} (por exemplo \d{6}).

4.3.4 Restrições complexas - conteúdo misto

```
<xs:complexType name="tparent element"
                                  mixed="true">
 <xs:sequence>
   <xs:element name="elo1" type="telo1"
       minOccurs="elo1Min" maxOccurs="elo1Max"/>
   <xs:element name="elo2" type="telo2"
       minOccurs="elo2Min" maxOccurs="elo2Max"/>
   ...
   <xs:element name="elon" type="telon"
       minOccurs="elonMin" maxOccurs="elonMax"/>
 </xs:sequence>
</xs:complexType>
```

Cria-se um tipo complexo para o elemento pai. Depois, especifica-se a sequência de sub-elementos referindo, para cada um, o nome (*name*), e o número mínimo (*minOccurs*) e máximo (*maxOccurs*) de ocorrências contínuas.

4.3.5 Restrições complexas - unicidade

```
Não é possível especificar...
```

4.3.6 Outras Restrições complexas

Aparentemente, nenhuma das restantes restrições tratadas nas outras duas abordagens pode ser especificada com XML-Schema. Por exemplo, para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, não se pode produzir uma restrição.

5 Templates Gerais

Olhando para os templates da secção anterior, é fácil aperceber-nos das fortes semelhanças entre as linguagens de especificação de restrições. Do ponto de vista do construtor de compiladores, podemos dizer que partilham a sintaxe abstracta com algumas diferenças de pouco relevo.

Estando interessados em criar templates para os vários tipos de restrições já enumerados, extraímos a gramática abstracta de cada template. Até ao momento considerámos apenas as linguagens XCSL e Schematron. As linguagens como o XML-Schema exigem que seja produzido o documento completo para efectuar a validação das instâncias, não são escaláveis para um conjunto de restrições semânticas.

Na subsecção que se segue especifica-se a gramática abstracta para as restrições de domínio e de dependência entre dois elementos ou atributos numa notação formal de gramáticas.

5.1 Restrição de domínio

A sintaxe abstracta das restrições de domínio é a seguinte:

```
Constraint -> address1 address2 Test

Test -> min | max | minin |maxin
```

em que:

address1 caminho para o nodo relevante na árvore documental abstracta;

address2 complementa o caminho escrito em *address1* — pode ser o nodo actual (".") ou um seu atributo;

TestList lista na qual se escrevem os limites do domínio (*minin* e *maxin* são usados para incluir os limites).

5.2 Dependência entre dois elementos ou atributos

Para as restrições de dependência entre dois elementos ou atributos, tem-se a sintaxe abstracta seguinte:

```
Constraint -> address1.1 adress1.2 address3 relop
```

em que:

address1.1 caminho para o nodo relevante (o ponto principal para a comparação) na árvore documental;

address1.2 complementa o caminho escrito em *address1* — pode ser o nodo actual (".") ou um seu atributo;

address3 caminho para o segundo argumento da comparação.

relop operação relacional a avaliar.

Para os outros tipos de restrições a sintaxe abstracta é obtida de modo análogo, pelo que nos abstemos de a colocar aqui.

6 A linguagem XTC

Para descrever os templates genéricos em XML, desenvolveu-se o XTC — uma linguagem com a qual se descrevem os parâmetros das restrições pretendidas e se indica a linguagem de restrições escolhida, sem ter que conhecer a sintaxe específica dessa linguagem.

Com um documento XTC pode gerar-se todos os ficheiros de especificação de restrições, ou seja, um ficheiro de especificação de restrições XCSL, Schematron ou ambos.

O DTD que se segue representa a gramática desta linguagem (extraímos as linhas dos elementos terminais, que são *#PCDATA*):

```
<!ELEMENT xtc (rest)+>
<!ATTLIST xtc
    lang (XCSL | Schematron) #REQUIRED
    date CDATA #IMPLIED
    version CDATA #IMPLIED>
<!ELEMENT rest (path, comp?, relop?, fixed*, digits*, subel*, seq?,
    message?)>
<!ATTLIST rest
    kind (domain | dependency | regexp | mixedcont | unicity)
        #REQUIRED>
<!ELEMENT path (path1, node1?, path2?, node2?)>
<!ELEMENT comp (min?, max?)>
<!ATTLIST comp
    inc (min | max | both) #IMPLIED>
<!ELEMENT fixed (first, number, value, relop?)>
<!ELEMENT digits (first, numberc, numberd, relop?)>
<!ELEMENT subel (name, card)>
<!ELEMENT seq (elt)+>
```

```
<!ELEMENT message (submesg)+>
<!ATTLIST submesg
    type (1 | 2 | 3) #REQUIRED>
```

Não podendo apresentar um exemplo completo, com todos os tipos de restrições, devido à sua extensão, escolhemos adaptar o exemplo da Certidão Fiscal por forma a mostrar como é que se processa a escrita das restrições em XTC. Nas sub-secções que se seguem apresenta-se uma restrição de domínio e uma restrição de dependência entre dois elementos ou atributos, escritas em XTC.

Como se verá mais à frente, cada um dos documentos XML que se apresentam nas sub-secções seguintes, ao passar pela arquitectura de três níveis, é transformado num validador para aquela restrição na linguagem escolhida.

Um requerimento de Certidão Fiscal é feito aquando do falecimento de uma pessoa que deixou bens, depois da relação de bens, para comprovar que determinados bens constam de uma herança e que foram cumpridos os deveres fiscais devidos.

6.1 Restrição de domínio

Vamos supor que toda e qualquer data do documento (neste caso a data do falecimento e a data do pedido) deve estar entre os anos 1900 e 2990. Esta é uma restrição de domínio, pelo que deveremos usar os parâmetros (a que correspondem elementos XTC) *path*, *comp* e *message*, como se mostra no exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE xtc SYSTEM "xtc.dtd">
<xtc lang="XCSL">
  <rest kind="domain">
    <path>
      <path1>//data</path1>
      <node1>@valor</node1>
    </path>
    <comp inc="min">
      <min>19000000</min>
      <max>29900000</max>
    </comp>
    <message>
      <submesg type="1">A data do pedido: </submesg>
      <submesg type="2">/cert_fis/corpo/pedido/data</submesg>
      <submesg type="1">não está entre 1900 e 2990!</submesg>
    </message>
  </rest>
</xtc>
```

Este documento XML especifica a restrição de domínio enunciada: *path* tem os sub-elementos *path1* e *node1* instanciados; em *comp* especifica-se que o ano de-

verá ser posterior ou igual (o atributo *inc* vale 'min') a 1900 e anterior a 2990; em *message* (mensagem a devolver ao utilizador) especifica-se, nos sub-elementos *submesg* com atributo *type* igual a '1' o texto corrido, e naquele com atributo *type* igual a '2', o caminho para o elemento cujo valor se pretende apresentar. Neste caso atribuímos o valor 'XCSL' ao atributo *lang*, pedindo assim a geração de um ficheiro de restrições em XCSL.

Recebendo este documento, o processador XTC gera um documento escrito em XCSL, como pretendido:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    generated by: XTC language processor
    Copyright (C) 2003
    Marta Jacinto, José C. Ramalho, Pedro Henriques
    GePL (DI, Uminho, Portugal)
    Version: 1.1  2003.02.09
-->
<!DOCTYPE cs SYSTEM "xcsl.dtd">
<cs>
  <constraint>
    <selector selexp="//data"/>
    <cc>
      ( @valor &gt;= 19000000 )
      and
      ( @valor &lt; 29900000 )
    </cc>
    <action>
      <message>A data do pedido:
        <value selexp="/cert_fis/corpo/pedido/data"/>
        não está entre 1900 e 2990!
      </message>
    </action>
  </constraint>
</cs>
```

6.2 Dependência entre dois elementos ou atributos

Num requerimento de Certidão Fiscal, exige-se que a data do pedido seja posterior à data de falecimento da pessoa a quem se refere o pedido. Para garantir que esta restrição se verifica, torna-se necessário comparar as duas datas referidas acima. Esta é uma restrição de dependência entre dois elementos ou atributos, pelo que deveremos usar os parâmetros (a que correspondem elementos XTC) *path*, *relop* e *message*, como se mostra no exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

```
<!DOCTYPE xtc SYSTEM "xtc.dtd">
<xtc lang="Schematron">
  <rest kind="dependency">
      <path>
          <path1>//pedido/data</path1>
          <node1>@valor</node1>
          <path2>/cert_fis/fecho/data/@valor</path2>
      </path>
      <relop>&lt;</relop>
      <message>
          <submesg type="1">A data de falecimento indicada: </submesg>
          <submesg type="2">/cert_fis/corpo/pedido/data</submesg>
          <submesg type="1">é posterior à data do pedido: </submesg>
          <submesg type="2">/cert_fis/fecho/data</submesg>
      </message>
  </rest>
</xtc>
```

Este documento XML especifica a restrição de domínio enunciada: *path* tem os sub-elementos *path1*, *node1* e *path2* instanciados (os dois primeiros para o primeiro ponto da comparação e o terceiro para o segundo ponto da comparação); *relop* especifica o operador a utilizar para a comparação (neste caso menor, que se representa pela entidade <); em *message* (mensagem a devolver ao utilizador) especifica-se, nos sub-elementos *submesg* com atributo *type* igual a '1' o texto corrido, e naqueles com atributo *type* igual a '2', caminhos para os elementos cujo valor se pretende apresentar. Neste caso atribuímos o valor 'Schematron' ao atributo *lang*, pedindo assim a geração de um ficheiro de restrições em Schematron.

Recebendo este documento, o processador XTC gera um documento escrito em Schematron, como pretendido:

```
<?xml version="1.0" encoding="iso-8859-1"?>
...
<schema>
  <pattern name="p2">
    <rule context="//pedido/data">
    <assert test="@valor&lt;/cert_fis/fecho/data/@valor"
                                          diagnostics="d2"/>
  </rule>
  </pattern>
  <diagnostics>
    <diagnostic id="d2">A data de falecimento indicada:
    <value-of select="/cert_fis/corpo/pedido/data"/> é posterior à data
    do pedido: <value-of select="/cert_fis/fecho/data"/>
    </diagnostic>
  </diagnostics>
</schema>
```

7 Implementação

As linguagens de especificação de restrições baseadas em regras como o XCSL e o Schematron estão implementadas sobre o XSLT naquilo a que se pode chamar uma arquitectura de dois níveis.

7.1 Arquitecturas de dois níveis

Por arquitectura de dois níveis, queremos dizer que alguém (o utilizador ou algum programa como por exemplo uma página web que ofereça uma interface ao utilizador) está consciente de que existem (e tem que seguir) dois passos por forma a obter o resultado final.

Como se disse na secção 3, o XCSL e o Schematron partilham a metodologia de implementação. Ambos os autores se aperceberam que se usassem XSLT para especificar as restrições, os processadores funcionariam, sem qualquer alteração, em qualquer plataforma.

Contudo, o utilizador ter que especificar uma folha de estilos XSL cada vez que pretende verificar algumas restrições, pode ser uma tarefa complexa e morosa. Assim, uma linguagem XML que "esconda" é tremendamente útil já que os detalhes "chatos" e a complexidade ficam do lado do processador XSL (único para aquela linguagem XML).

Em suma, se pretendermos usar o XCSL ou o Schematron, basta-nos seguir estes passos:

1. Especificar as restrições num documento XML (usando XCSL ou Schematron).
2. Usar um processador XSL para processar o documento XML usando a folha de estilos especial para a linguagem que estamos a utilizar. Este processador vai produzir uma folha de estilos que implementa completamente as restrições especificadas no passo anterior.
3. Validar as instâncias XML aplicando a folha de estilos gerada no passo anterior às várias instâncias XML da família de documentos.

Neste artigo propomos um terceiro nível de abstracção — abstracção da linguagem de especificação de restrições para "esconder" os detalhes do utilizador.

7.2 O processador XTC, uma arquitectura de três níveis

A arquitectura de três níveis que propomos pode ser descrita pelos passos que descrevemos de seguida e que será necessário seguir para obter a validação dos documentos XML:

1. Especificar as restrições num documento XML (usando o XTC). A escolha da linguagem final é feita através de um parâmetro.

2. Usar um processador XSL para processar o documento XML do passo anterior usando a folha de estilos especial: uma folha de estilos geradora de restrições de nível 2. Este processador vai produzir um documento XML em XCSL, Schematron ou XML-Schema (por enquanto ainda só numa das duas primeiras) dependendo do valor do parâmetro.

3. Usar um processador XSL para processar o documento XML gerado no passo anterior usando a folha de estilos especial para a linguagem que estamos a utilizar. Este processador vai produzir uma folha de estilos que implementa completamente as restrições especificadas no primeiro passo.

4. Validar as instâncias XML aplicando a folha de estilos gerada no passo anterior às várias instâncias XML da família de documentos.

Na secção 6 mostraram-se os dois primeiros passos desta arquitectura.

Poder-se-ia dizer que o preço que pagamos por ter mais níveis de abstracção é o aumento de complexidade na altura da implementação, mas esta abordagem possibilitará prosseguir com o estudo das transacções em tempo real.

8 Conclusão

Presentemente estão a ser discutidos nos comités W3C e ISO uma nova linguagem de esquema XML e talvez até mesmo um novo modelo de processamento XML. Tendo avaliado o que havia de diferente em várias linguagens e criado uma abstracção com as características melhores de cada uma delas, podemos contribuir activamente para esta nova linguagem.

Outra ideia que surgiu deste trabalho tem a ver a especificação de uma estrutura XML Pipe. As arquitecturas em vários níveis estão a ser desenvolvidas cada vez mais e há a necessidade de tornar os vários níveis transparentes para o utilizador. Há uma proposta no W3C para uma linguagem chamada "XML Pipe Definition Language", mas ainda não há qualquer implementação. Não obstante, a implementação de tal sistema levanta questões interessantes como: o que acontece se um componente falhar? (a resposta mais óbvia é abortar todo o processo, mas há respostas mais interessantes como ignorar ou tentar compensar); podemos ter sequências de componentes paralelas? (nesta situação, o contexto de falha é mais complexo...). Trabalhar nesta linha, criar uma álgebra para transacções em tempo real e encontrar uma implementação adequada para essa álgebra é bastante interessante e é o que se seguirá no futuro próximo.

No que diz respeito ao XTC, continuaremos a incorporar os tipos de restrições.

Referências

[1] L. Dodds. Schematron: Validating xml using xslt. In *XSLT UK Conference*, Keble College, Oxford, England, 2001.

[2] Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Ian Stokes-Rees, Kevin Williams, Kurt Cagle, Nikola Ozu, and Jeni Tennison. *Professional XML-Schemas*. Wrox Press, 2001.

[3] M. Jacinto, G. Librelotto, J. C. Ramalho, and P. Henriques. Constraint specification languages: comparing XCSL, Schematron and XML-Schemas. In *XML Europe'2002*, Barcelona, Spain, May 2002.

[4] M. H. Jacinto. Validação semântica em documentos xml. Master's thesis, Dep. Informática, Escola de Engenharia, Universidade do Minho, Outubro 2002.

[5] M. H. Jacinto, G. R. Librelotto, J. C. L. Ramalho, and P. R. Henriques. XCSL: XML Constraint Specification Language. In *XXVIII Conferencia Latino Americana de Informática (CLEI02)*, Uruguai, 2002.

[6] J. C. Ramalho. *Anotação Estrutural de Documentos e sua Semântica*. PhD thesis, Dep. Informática, Escola de Engenharia, Universidade do Minho, July 2000.

[7] J. C. Ramalho and P. Henriques. Constraining content: Specification and processing. In *XML Europe'2001*, Internationales Congress Centrum (ICC), Berlin, Germany, Apr. 2001.

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

14 Fev.	S6(11.00h) XML e Bases de Dados	[*IXDIRQL: Uma linguagem interactiva de perguntas para o XML*] - **Alda Lopes Gançarski**, LIP6 - Universidade Pierre et Marie Curie Paris 6, Paris, França; [*XML Topic Map Builder: Specification and Generation*] – **Giovani Rubert Librelotto**, DI-UM; **José Carlos Ramalho**, DI-UM; **Pedro Rangel Henriques**, DI-UM [*HaQuery: A Haskell Combinator Library for Querying XML\\ Extended Abstract*] - **João Saraiva**, DI-UM; **David Lopes**, DI-UM; **António Faria**, DI-UM;

IXDIRQL: Uma linguagem interactiva de perguntas para XML

Alda Lopes Gançarski - e-mail : Alda.lopes@lip6.fr
LIP6 - University P. et M. Curie Paris 6, France
Pedro Rangel Henriques - e-mail : prh@di.uminho.pt
University of Minho, Portugal

10 de Fevereiro de 2003

Resumo

Este artigo apresenta uma linguagem interactiva de perguntas para XML, a IXDIRQL. A IXDIRQL é uma extensão à linguagem XQL para permitir ao utilizador a construção interactiva das perguntas. A interacção é feita da seguinte forma: o utilizador tem acesso aos resultados intermédios de cada pergunta, podendo analisá-los e selecionar o subconjunto de elementos que ele considera interessantes. Este procedimento ajuda o utilizador a obter a pergunta que conduz ao resultado final desejado. A IXDIRQL também permite operações de similaridade textual da Procura de Informação tradicional. O resultado de uma operação de similaridade é uma lista de elementos organizada de forma a facilitar ao utilizador a detecção da informação relevante.

Keywords: Procura de informação, documentos estruturados, XML, linguagens de perguntas para XML

1 Introdução

Quando vários documentos contêm o mesmo tipo de informação, isto é servem o mesmo fim, é natural querer-se descrevê-la da mesma forma em todos eles. Por exemplo, os manuais técnicos dos produtos de uma empresa, ou as convocatórias e actas das reuniões de uma associação, são três famílias de documentos dentro das quais todos os membros têm a mesma estrutura; por isso, é conveniente especificá-la de uma forma normalizada. Esta ideia esteve na origem da meta-linguagem *Standard Generalized Markup Language* (SGML, [Her94]) para definir linguagens de anotação descritiva (que expõe a

estrutura lógica dos documentos). Assim se tornou mais fácil o intercâmbio de documentos estruturados do mesmo tipo. A definição dos componentes estruturais de um tipo de documentos é feita na especificação *Document Type Definition* (DTD). Mais tarde surgiu a linguagem *Hypertext Markup Language* (HTML, [RHJ99]), uma aplicação SGML para mostrar documentos na *Web*. Esta linguagem proporciona essencialmente anotações que descrevem como o conteúdo dos documentos deve ser apresentado. Recentemente, foi criada a linguagem *eXtensible Markup Language* (XML, [BPSM98]), uma simplificação do SGML adaptada para o intercâmbio de documentos na *Web*. Os documentos XML têm uma descrição mais rica do que os documentos HTML, cujas anotações são simples mas limitadas.

O objectivo do nosso trabalho é a **procura de informação (PI) em documentos XML** [BYRN99] . A PI (conhecida por *Information Retrieval* em inglês) consiste na procura e entrega ao utilizador dos documentos que se julgam relevantes para satisfazer o seu pedido (ou pergunta, do inglês *query*), minimizando o número de documentos não relevantes entregues. Além disso, os documentos são ordenados por ordem decrescente da sua relevância. Neste processo, tipicamente, o utilizador vai lendo cada documento, a partir do início da lista, até encontrar informação relevante suficiente, ou até à altura em que começa a verificar que a maior parte dos documentos não é relevante, (a quantidade é subjectiva); nesse caso, normalmente o utilizador abandona a lista e refina a pergunta. Uma pergunta consiste numa expressão em linguagem natural que descreve o assunto procurado. Usualmente, a relevância dum documento é obtida através duma função de similaridade entre ele (uma sua síntese, ou caracterização) e a pergunta.

Para tirar partido da informação estrutural dos documentos XML, o formato das perguntas foi enriquecido de modo a poder-se aceder a (referir) partes específicas dos documentos. Uma pergunta consiste, então, numa sequência de restrições estruturais (caminhos, filtros, grupos) sobre o conteúdo (comparações com valores textuais ou numéricos). Actualmente, o *W3C Consortium* está a desenvolver a XQuery [BCF+02] como a futura norma de perguntas para XML. A XQuery baseia-se em diferentes linguagens existente, entre as quais se destacam a XQL [Rob99] e a XML-QL [DFLS98] (ver [FSW99] para uma revisão sobre linguagens de interrogação para XML). A XQL foi desenvolvida para aplicações XML onde as anotações servem essencialmente para descrever a estrutura lógica dos documentos (a visão do XML centrada no documento [FG02]). A XML-QL, por sua vez, foi criada pela comunidade de bases de dados para intercambiar dados numa forma estruturada (a visão do XML centrada nos dados). Todas estas linguagens são consideradas como de *procura de dados* (em inglês, *data retrieval*) porque as

respostas às perguntas são todos os elementos que *satisfazem* a respectiva pergunta. Não há, portanto, nestas linguagens a noção de relevância. Recentemente, alguns trabalhos ([TW00], [FG02], [CK01]) propõem extensões às linguagens de perguntas acima referidas onde se incluem restrições de similaridade textual como é tradicional no contexto da PI. O resultado de uma restrição de similaridade é uma lista de elementos ordenados pela sua relevância.

1.1 A nossa proposta

Tendo em conta as diferentes linguagens de perguntas para XML que surgiram até ao momento, propomos a linguagem IXDIRQL (do inglês *Interactive XML Data and Information Retrieval Query Language*), uma extensão à linguagem XQL que permite dois novos tipos de operações: (1) similaridade textual da PI tradicional, onde a lista de elementos resultante é organizada de forma a facilitar ao utilizador a detecção da informação relevante; (2) seleção imediata do subconjunto de elementos que satisfazem o utilizador, perante cada resultado intermédio do processamento de uma pergunta.

A construção de perguntas para procurar membros numa família de documentos XML, nem sempre é uma tarefa fácil para um utilizador comum porque, entre outras razões: ele nem sempre conhece suficientemente bem a linguagem de perguntas; ele não sabe bem a priori o que deseja; ele pensa saber o que deseja mas, face ao resultado final da pergunta, apercebe-se de que não é o que procurava. Adoptando o paradigma da interacção, cada operação que o utilizador especifica conduz a um resultado intermédio que ele pode analisar; assim, pode escolher melhor a próxima operação, alterar uma operação já inserida, ou fazer seleção dos elementos interessantes, até chegar ao resultado final pretendido.

Até ao momento, não temos conhecimento de nenhum trabalho que proponha uma construção interactiva das perguntas para XML. Existem trabalhos que permitem refinar sucessivamente as perguntas, onde os resultados de cada pergunta podem ser introduzidos como base de procura para a seguinte (como em [MP00]). No entanto, estes trabalhos não permitem operações de selecção nem de similaridade.

A IXDIRQL importa da linguagem XQL as operações sobre a estrutura e as comparações textuais. Escolhemos a XQL por se tratar de uma linguagem simples e concisa, proporcionando diversas formas de aceder a dados em XML. Contudo, é possível acrescentar à linguagem XQuery as novas operações da IXDIRQL, quando a primeira vier a ser a norma de linguagem de interrogação para documentos XML.

As secções seguintes descrevem as diferentes operações contempladas pela linguagem IXDIRQL, nomeadamente as que são importadas da linguagem XQL (secção 2), a similaridade textual (secção 3) e a seleção (secção 4). Para melhor compreensão, serão apresentados exemplos simples baseados no DTD da figura 1. Este DTD define um artigo composto por um título, uma lista de autores, uma data, uma lista de secções, uma conclusão e uma lista de referências. Cada secção tem um título, um componente textual e uma lista opcional de subsecções. Uma subsecção é formada por um título e um componente textual. Finalmente, uma referência tem o título, os autores, a data e o livro do respectivo artigo. Os elementos título, autor, data, conclusão e livro são puramente textuais.

<!DOCTYPE artigo [
<!ELEMENT artigo (titulo, autor+, data, secção+, conclusão, referência+)>
<!ELEMENT titulo (#PCDATA) >
<!ELEMENT autor (#PCDATA) >
<!ELEMENT data (#PCDATA) >
<!ELEMENT secção (titulo, #PCDATA, subsecção) >*
<!ELEMENT subsecção (titulo, #PCDATA) >
<!ELEMENT conclusão (#PCDATA) >
<!ELEMENT referência (titulo,autor +, data, livro) >
<!ELEMENT livro (#PCDATA) >>]

Figura 1: Exemplo de um DTD que descreve a estrutura de artigos.

2 Operações importadas da linguagem XQL

A IXDIRQL importa da XQL operações: de selecção estrutural (caminho, filtro e grupo); lógicas (e, ou e não); e sobre conjuntos (união, intersecção e negação).
As operações de selecção estrutural permitem ao utilizador a especificação do tipo de elementos que ele quer. Por exemplo, o caminho:

/artigo/titulo

refere-se aos elementos anotados como sendo títulos de artigos. É possível restringir os elementos de um certo tipo àqueles que satisfazem uma determinada condição. Essa condição é incluída num filtro imediatamente a

seguir ao tipo de elementos que se quer restringir. Se, por exemplo, o utilizador quer aceder às referências apenas dos artigos com o título "Aplicações XML", a pergunta é a seguinte:

$$/artigo[titulo = "Aplicações\ XML"]/referência$$

A restrição desta pergunta é uma simples comparação textual, embora outras mais complexas possam ser especificadas.
Supondo, agora, que se pretende o título, os autores e a data de cada artigo, faz-se a pergunta que se segue:

$$/artigo\{titulo \mid autor \mid data\}$$

Esta pergunta permite o agrupamento das ocorrências de elementos de tipos diferentes, inseridos no conteúdo do mesmo elemento. O símbolo "|" representa a operação de união de conjuntos. Outras operações de conjuntos são a intersecção e a negação.
Quanto às operações lógicas, dá-se um exemplo a seguir, em que se procuram os artigos escritos em 2001 pelo autor "Silva". A pergunta correspondente utiliza um *e* lógico (*and*) das duas comparações textuais:

/artigo[data = "2001" and autor = "Silva"]

3 A similaridade textual

A primeira extensão que é necessário fazer à linguagem XQL é a inclusão de operações de similaridade textual ao nível dos elementos. A IXDIRQL permite procura por similaridade orientada à relevância (i.e., procurar elementos de qualquer tipo) e procura de elementos de um tipo especificado. O cálculo da relevância é independente da IXDIRQL. No entanto, nós consideramos importante a organização da lista ordenada dos elementos resultantes porque é o ponto de partida para o utilizador detectar a informação relevante. De forma idêntica à PI tradicional, espera-se que o utilizador procure a informação relevante de cada elemento começando pelo início da lista.

Não havendo informação acerca da relevância das suas partes, o conteúdo de um elemento é lido percorrendo, em profundidade, a árvore correspondente da esquerda para a direita.

Nas sub-secções seguintes assume-se a relevância de um elemento calculada por uma determinada função. Para cada novo operador apresentado, explicar-se-á a construção da lista de elementos ordenada (LOrd) resultante.

3.1 O operador *ROSearch*

O operador *ROSearch* (do inglês *Relevance-Oriented Search* [FG02]) permite fazer a procura orientada à relevância.

Até ao momento, todos os trabalhos propõem a LOrd resultante ordenada pela relevância dos elementos. Neste tipo de LOrd, torna-se difícil explorar a distribuição da informação relevante devido às relações hierárquicas entre os diferentes elementos envolvidos. Por exemplo, a pergunta que se segue diz respeito à procura de elementos de qualquer tipo sobre "Compilação":

ROSearch "Compilação"

Assuma-se a seguinte LOrd resultante:

<secção 2, subsecção 3.2, secção 4, subsecção 2.4, ..., subsecção 2.1, ...>

Analisando esta LOrd, o utilizador vai obter a secção 2 sem saber como a informação relevante está distribuída no seu conteúdo. Então, o utilizador vai ler toda esta secção para obter a sua informação relevante. Continuando a analisar a LOrd, após outros elementos, surgem as subsecções da mesma secção, mas o utilizador já as analisou. Isto mostra que muita informação na LOrd não é utilizada, uma vez que o utilizador não vai analisar o mesmo elemento mais do que uma vez. A nossa proposta é tirar partido dos valores de relevância de forma a que o utilizador leia o menos possível elementos não relevantes, até obter informação suficiente. Isto consegue-se construindo recursivamente LOrds de acordo com as seguintes regras:

- (1) Os elementos são ordenados por ordem decrescente de relevância.
- (2) Cada elemento com um conteúdo estruturado (i.e., composto por mais do que um elemento) é inserido na LOrd dentro de uma nova lista de elemento (LElem); a LElem de um elemento é construída unindo a LOrd dos seus descendentes ao próprio elemento.
- (3) Cada elemento é ordenado uma e uma só vez.

Vejamos primeiro como construir uma LOrd através de um exemplo para depois se perceberem as suas vantagens.

Suponha-se que o utilizador faz uma pergunta orientada à relevância para o documento representado na árvore da figura 2. Os nodos desta árvore correspondem aos elementos desse documento. Cada nodo está associado ao

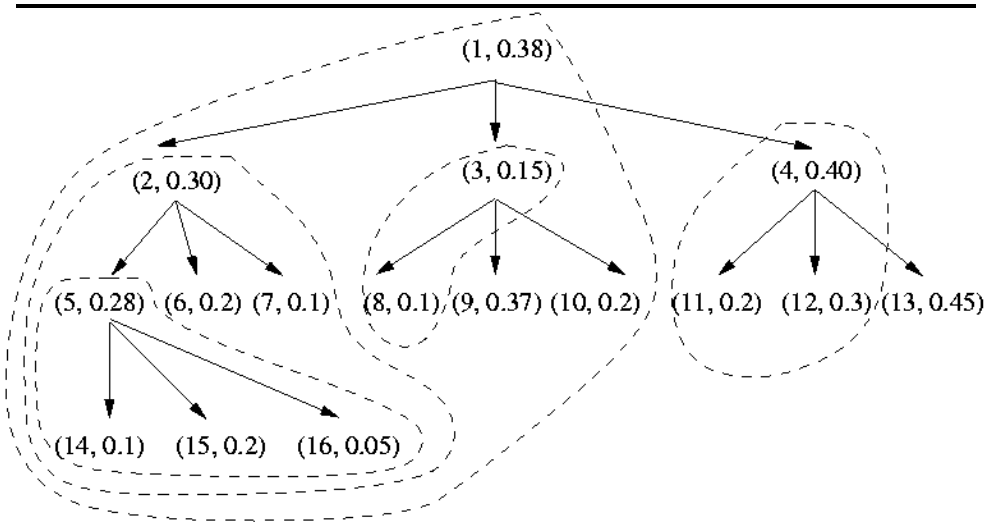


Figura 2: Árvore representativa de um documento exemplo.

seu identificador (de 1 a 16) e o valor da relevância do elemento respectivo (no intervalo [0..1]). Cada LElem é rodeada por uma linha quebrada.

O primeiro elemento da LOrd do documento é o 13 porque é o que tem a maior relevância (0.45). Isto respeita a primeira regra acima apontada. Dos restantes, o elemento 4 é o que tem maior relevância. Assim, de acordo com a segunda regra, ele é incluído na LOrd do documento dentro da sua LElem (repare-se que o elemento 4 tem um conteúdo estruturado). Esta LElem é construída unindo a LOrd dos descendentes do elemento 4 (<12, 11>) ao próprio elemento, obtendo-se <12, 11, 4>. A LOrd do elemento 4 exclui o elemento 13 porque ele já foi ordenado antes, como indica a terceira regra. De facto, os descendentes de um elemento que tenham uma relevância maior, já foram apresentados ao utilizador, não havendo necessidade de os incluir de novo. Neste momento, a LOrd do documento é <13, <12, 11, 4> >. O elemento 1 é o próximo a ser ordenado, inserindo-se a sua LElem na LOrd do documento. Para isso, constroi-se primeiro a LOrd dos descendentes do elemento 1 (<9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3> >) respeitando as mesmas regras que temos vindo a usar; em seguida, a LElem do elemento 1 (<9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3>, 1>) é obtida unindo a LOrd dos seus descendentes com o próprio elemento; por fim, a LElem do elemento 1 é inserida na LOrd do documento no estado anterior (<13, <12, 11, 4> >), obtendo-se:

<13, <12, 11, 4>, <9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3>, 1> >

Para melhor se compreenderem as vantagens de uma LOrd, representêmo-la por uma árvore onde: cada nodo é um elemento; a raiz significa o conjunto de elementos para analisar; as LElems são divididas no correspondente elemento (representado por um nodo) e na respectiva LOrd dos seus descendentes (representada pela subárvore do mesmo nodo). A figura 3 mostra a árvore representativa da LOrd do exemplo anterior. Esta árvore tem o mesmo número de nodos para o utilizador analisar do que a árvore do documento (rever figura 2); os seus nodos são também lidos da esquerda para a direita em profundidade; contudo, nesta árvore o processo pode ser acelerado quando o utilizador está a analisar a sequência de filhos de um nodo. De facto, se, a partir de uma certa altura, a maior parte desses filhos não for relevante (como em PI tradicional, a quantidade é subjectiva), o utilizador pode decidir analisar directamente o nodo pai. Isto deve-se ao facto de que os restanes filhos têm cada vez menor probabilidade de serem relevantes.

Por exemplo, imagine-se que o utilizador está a procurar informação relevante no elemento 1 do mesmo exemplo (figura 2). Se os elementos 9 e 2 forem considerados não relevantes, o utilizador pode decidir considerar também não relevantes os restantes elementos filhos (10 e 3), analisando directamente o elemento pai (elemento 1).

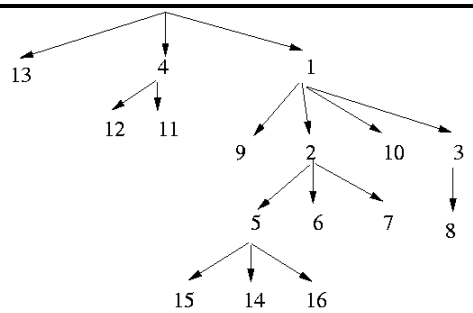


Figura 3: A árovore que representa a LOrd do documento da figura 2.

3.2 O operador *ROSearchTE*

A procura orientada à relevância pode ainda ser feita pelo operador *ROSearchTE*. Este operador separa o resultado final numa LOrd por cada tipo de elemento da colecção. Cada LOrd ordena os elementos por ordem decrescente

de relevância. Esta operação, é interessante quando os elementos são logicamente muito diferentes, como secções e referências dum artigo. As LOrds serão apresentadas ao utilizador começando pelos elementos textuais (as folhas nas árvores dos documentos) e subindo na hierarquia até ao elemento raiz. Assim, facilita-se a procura da informação relevante em cada elemento das LOrd pois os elementos do seu conteúdo já foram todos lidos e analisados antes. Por exemplo, uma operação de ROSearchTE sobre uma colecção de artigos desfinidos pela DTD da figura 1 conduz a uma LOrd por cada tipo de elemento definido, ou seja:

- (1) elementos textuais (título, livro, data, autor, conclusão);
- (2) elementos que incluem os elementos textuais (referência, subsecção);
- (3) elemento que inclui elementos dos pontos anteriores (secção);
- (4) o elemento raiz (artigo).

Se o utilizador analisar as LOrds pela ordem indicada, o facto de que uma LOrd não seja completamente percorrida (se o utilizador parar quando começarem a surgir muitos elementos não relevantes) não constitui um problema ao analisar a LOrd seguinte. Isto porque os elementos não analisados têm pouca probabilidade de serem relevantes. Assim, ao ler um elemento da LOrd seguinte, os descendentes que não foram lidos foram assumidos como não relevantes.

Uma situação que pode acontecer frequentemente é a leitura das LOrds por uma ordem diferente da sugerida. Para que o utilizador possa também, nestes casos, aceder o mais rápido possível à informação relevante, cada elemento deve ser incluído na respectiva LElem (construída segundo as regras da secção anterior).

3.3 O operador *Sim*

O operador *Sim* permite fazer a procura por similaridade nos elementos de um certo tipo. A LOrd resultante ordena os elementos por ordem decrescente de relevância. Para, mais uma vez, se facilitar a detecção da informação relevante, os elementos são incluídos na respectiva LElem (construída segundo as regras da secção 3.1).

Por exemplo, seja a colecção C=*{1, 17, 30}* de documentos, onde se inclui o documento 1, figura 2 (identificado pelo seu nodo raiz). Se o utilizador quiser documentos sobre "Compilação", ele faz a pergunta:

/documento Sim "Compilação"

Suponha-se que o sistema de PI atribui aos documentos 1, 17 e 30 os valores de relevância 0.38, 0.4 e 0.2, respectivamente. O documento 17 é o mais relevante, seguido do documento 1 e, por fim, do documento 30. A LOrd resultante é, então, a seguinte:

< <..., 17>,
<13, <12, 11, 4>, 9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3>, 1>,
<..., 30> >

4 A seleção dos elementos interessantes

A IXDIRQL permite ao utilizador a selecção dos elementos interessantes de cada resultado intermédio do cálculo das perguntas. Esta é a parte central da IXDIRQL. Passamos a explicar, nas sub-secções seguintes, as três formas existentes para fazer essa selecção.

4.1 O operador *Select*

O operador *Select* permite a selecção baseada em critérios que o utilizador não sabe especificar (ou não quer, por ser complicado). Por exemplo, suponha-se que o utilizador pretende artigos escritos por "Silva", como indica a pergunta seguinte:

/artigo[autor = "Silva"]

Imagine-se que o resultado inclui artigos que o utilizador não quer devido, por exemplo, à qualidade da editora ou ao tema abordado. Então, ele pode fazer a escolha do conjunto de artigos que lhe interessam (por exemplo, {artigo 4, artigo 8, ...}) usando o operador *Select* da seguinte forma:

/artigo[autor = "Silva"] Select {artigo 4, artigo 8, ...}.

4.2 O operador *SelectN*

Outra forma de fazer a selecção é indicando o número de elementos desejado. Esta operação é interessante quando o resultado é maior do que o pretendido pelo utilizador, não havendo um critério específico associado. Assim, ele pode usar o resultado final directamente para outra aplicação com o tamanho adequado. Além disso, se o utilizador sabe à priori que apenas precisa de

um conjunto limitado de elementos, o operador *SelectN* permite-lhe reduzir o tamanho dos operandos em futuras operações da pesquisa. Isto traduz-se num aumento de eficiência.

Fazendo *SelectN* num resultado intermédio, obtém-se a lista dos seus n primeiros elementos, sendo n dado pelo utilizador.

Por exemplo, pegando na pergunta da secção anterior (*/artigo[autor = "Silva"]*), imagine-se que o resultado é uma lista muito grande de artigos e o utilizador está apenas interessado numa lista com 10 documentos, para qualquer aplicação posterior (p.ex., para escrever um *Curriculum Vitae*). Não havendo nenhum critério para escolher os artigos, o utilizador pode simplesmente pedir os 10 primeiros servindo-se do operador *SelectN*. A pergunta é, então, aumentada da forma seguinte:

/artigo[autor = "Silva"] SelectN 10

4.3 O operador *JudgeRel*

A selecção torna-se indispensável num resultado intermédio de uma operação de similaridade para se obter o resultado final desejado. Suponha-se que o utilizador quer referências feitas em artigos cujo título seja similar (contenha a palavra) "XML". Isto escreve-se em IXDIRQL da seguinte forma:

/artigo[titulo Sim "XML"]/referência

O resultado desta pergunta é a lista de referências ordenada por ordem decrescente da relevância do título do respectivo artigo. O utilizador não pode analisar a relevância destas referências, tendo que confiar nos valores de relevância dados pelo sistema.

Alternativamente, o utilizador pode, para cada referência, analisar o título do respectivo artigo; se for relevante, aceita a referência, caso contrário, rejeita-a. Isto torna-se complicado se houver mais do que uma condição de similaridade ou se o resultado for muito grande.

O operador *JudgeRel* resolve este problema pois permite a selecção imediata dos elementos relevantes de um resultado intermédio de similaridade. Assim, o resultado final é garantidamente relevante e pode ser directamente usado noutras aplicações.

Retomando a pergunta anterior, o utilizador pode, por exemplo, considerar relevante o conjunto {*title 4, title 9, ...*} de títulos, escrevendo, então a pergunta:

/artigo[titulo Sim "XML"JudgeRel {título 4, título 9, ...}]/referência.

As referências do resultado final desta pergunta correspondem aos artigos cujos títulos são relevantes (i.e., referem-se garantidamente ao tema "XML"e contém informação importante dentro das necessidades especícas no contexto actual do utilizador).

5 Conclusão e trabalho futuro

Neste artigo foi apresentada a IXDIRQL, uma linguagem de perguntas interactiva para procurar documentos XML. Esta linguagem é inovadora em dois aspectos: (1) permite *seleccionar* o subconjunto de elementos interessantes nos resultados intermédios de cada pergunta; (2) permite operações de *similaridade textual*, resultando dai uma lista de elementos ordenada por relevância, de forma a facilitar ao utilizador a detecção da informação pretendida.

O acesso aos resultados intermédios implica uma certa perda de eficiência. Isto é compensado pelo facto de que o tempo total de construção das perguntas é melhorado por este processo, quando à partida o utilizador não está seguro da pergunta adequada. Neste caso, é preferível perder algum tempo a analisar e filtrar os resultados intermédios do que a introduzir e compor muitas perguntas, até obter o resultado final pretendido.

As operações de similaridade e de seleção contribuem para a eficiência do sistema porque: por um lado, a organização da resposta em listas ordenadas (LOrds) ajuda o utilizador a detectar mais rapidamente os elementos relevantes; por outro lado, a selecção reduz o tamanho dos operandos para futuras operações.

Tendo por base a preocupação com os aspectos relacionados com a eficiência de um sistema de processamento da IXDIRQL, referem-se a seguir algumas pistas para trabalho futuro.

Para preservar uma boa eficiência, o sistema projectado deve permitir, quando o utilizador estiver seguro da pergunta adequada, a introdução de perguntas completas, como alternativa à sua construção incremental (desactivando as operações de selecção).

O cálculo de cada operação deve ser feito de forma eficiente, pelo que a construção e o acesso aos índices deve ser bem planeado.

O ambiente de edição da IXDIRQL terá de ter associado um sistema de cálculo incremental das perguntas, devido à eficácia deste paradigma. Isto significa que, cada vez que uma operação é introduzida ou alterada, apenas se (re)calcula a parte da pergunta dependente dessa operação.

Agradecimento

Os autores agradecem à "Fundação para a Ciência e Tecnologia (FCT)" o apoio financeiro concedido através da bolsa de doutoramento que suporta o trabalho da Alda Lopes.

Referências

[BCF+02] Scott Boag, Dor Chamberlin, Mary Fernandez, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language. W3C Working Draft. http://www.w3.org/TR/xquery/, November 2002.

[BPSM98] Tim Bray, Jean Paoli, C. Sperberg-McQueen, and Eve Maler. Extensible markup language (XML) 1.0 (second edition), W3C Recommendation. http//www.w3c.org/TR/REC-xml, October 1998.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[CK01] Taurai T. Chinenyanga and Nicholas Kushmerick. Expressive retrieval from xml documents. In *ACM SIGIR'01*, New Orleans, Louisiana, USA, September 2001.

[DFLS98] Alin Deutsch, Mary Fernandez, Alon Levy, and Dan Suciu. XML-QL: A query language for XML. W3C Note, August 1998.

[FG02] Norbert Fuhr and Kai Grobjohann. XIRQL: Xirql: An xml query language based on information retrieval concepts. In *ACM SIGIR 2002 Workshop on XML*, August 2002.

[FSW99] M. Fernandez, J. Simeon, and P. Wadler. XML query languages: Experiences and exemplars, 1999.

[Her94] Eric van Herwijnen. *Practical SGML*. Kluwer Academic Publishers, 1994.

[RHJ99] Dave Raggett, Arnaud Hors, and Ian Jacobs. HTML 4.01 specification w3c recommendation. http://www.w3.org/TR/html401, December 1999.

[Rob99] J. Robie. XQL'99 proposal. http://www.texcel.unc.edu/xql/xql-proposal.html, 1999.

[TW00] Anja Theobald and Gerhard Weikum. Adding relevance to XML. In *3rd International Workshop on the web and Databases (WebDB)*, 2000.

XML Topic Map Builder: Specification and Generation

Giovani Librelotto
University of Minho
Department of Informatics
Braga, Portugal, 4710-057
grl@di.uminho.pt

José C. Ramalho
University of Minho
Department of Informatics
Braga, Portugal, 4710-057
jcr@di.uminho.pt

Pedro R. Henriques
University of Minho
Department of Informatics
Braga, Portugal, 4710-057
prh@di.uminho.pt

ABSTRACT

Everyday thousands of new information resources are linked to the web. This way the web is growing very fast what makes search tasks more difficult. To solve the problem several initiatives were undertaken and a new area of research and development emerged: the one called Semantic Web.

When we refer to the semantic web we are thinking about a network of concepts. Each concept has a group of related resources and can be related to other resources, thought we can use this concept network to navigate among web resources or simply among information resources. From the undertaken initiatives one became an ISO standard: Topic Map ISO 13250.

The aim of this paper is to introduce a Topic Map (TM) Builder, that is a processor that **extracts topics and relations** from instances of a family of XML documents.

A Topic Map Builder is strongly dependent on the resources structure. So if we have an heterogeneous set of information resources we have to implement several Topic Map Builders. This is not very useful, and to overcome this problem we have created an XML abstraction layer for Topic Map Builders.

To describe that process, i.e. the extraction of knowledge from XML documents to produce a TM, we present a language to specify topic maps for a class of XML documents, that we call XSTM (XML Specification for Topic Maps).

We also discuss a XSL processor that automatically generates the Extractor from its formal specification, the XSTM-P.

1. INTRODUCTION

This paper is concerned with knowledge extraction from documents marked up in XML. To go straight to our target topic, we clearly assume that the reader is familiar with XML and companion for documents structuring and processing. For details about these topics we suggest the reading of [6] [3] [7].

There are many tools for creation of Topic Maps (TM for short) like Mapalizer[1]. However, we do not know anyone that from a XML specification of relevant items of the information creates automatically the Topic Map using just XML tools. There is a chapter about *Automated/Automatic Topic Map Construction* in [1] but it does not explain how it is possible to implement this topic map's constructor.

In this context, we understood the need for a Topic Map specification language to enable the systematic derivation of a TM Builder. XSTM (XML Specification of Topic Maps), the proposed language, is an XML language; so it becomes possible to create a Topic Map Builder that generates Topic Maps from XML documents using an XML dialect. That approach offers a complete XML framework to the user. The benefits of such an approach are obvious.

In section 2 there is an overview about Topic Maps. We give a brief introduction to the subject and we show some of its characteristics. At the end of the section we present a case study that will help the reader to understand how a Topic Map is created.

Section 3 describes the steps followed to develop the Topic Map Builder. The information extraction from XML resources depends of the DTD/Schema of the resources. To overcome the problem of having to code a new Extractor everytime we have a different class of XML resource we have created an abstraction layer. This layer is composed by one specification in XSTM, a new XML language we have created for this purpose.

A formal specification of the proposed language, XSTM, is provided in section 4, showing a diagram that depicts the XML-Schema [4], and listing the respective DTD; in that section, we also detail the elements introduced in the DTD and illustrate their use through examples. For those more familiar with grammar based language definitions, we include a CFG for XSTM as well.

The details concerning the implementation of the XSTM processor are introduced in section 5.

A synthesis of the paper and hints on future work are presented in the last part, section 6, together with some metrics about XSTM use.

2. XML TOPIC MAPS

A TM is a formalism to represent knowledge about the structure of an information resource and to organize it in "topics". These topics have occurrences and associations that represent and define relationships between them. Information about the topics can be inferred by examining the associations and occurrences linked to the topic. A collection of these topics and associations is called a topic map.

Topic Maps can be seen as a description of what is about a certain domain, by formally declaring topics, and by linking the relevant parts of the information set to the appropriate topics [2].

A topic map expresses someone's opinion about what the topics are, and which parts of the information set are relevant to which topics. Charles Goldfarb [5] usually compares Topic Maps to a GPS (Global Positioning System) applied to the information universe. Talking about Topic Maps is talking about knowledge structures. Topic Maps are the basis for knowledge representation and knowledge management.

Enabling to create a "virtual map" of information, the information resources stay in its original form and so they are not changed. Then, the same information resource can be used in different ways,

[1] http://www.topicmapping.com/mapalizer

for different topic maps. As it is possible and easy to change the map itself, information reuse is achieved.

Topic Map architecture was also designed to allow merging between topic maps without requiring the merged topic maps to be copied or modified.

2.1 The characteristics of Topic Maps model

A topic map is basically an XML document (or set of documents) in which different element types, derived from a basic set of architectural forms, are used to represent topics, occurrences of topics, and relationships (or associations) between topics [9].

Topics are the main building blocks of topic maps[8]. In its most generic sense, can be anything. A person, an entity, a concept, really anything regardless of whether it exists or has any other specific characteristic. It constitutes the basis for the topic maps creation. It can be seen as a "multi-headed link, that points to all its occurrences" [2]. This "link" aggregates information about a given subject (the thing that the topic is about).

Each topic has a topic type or perhaps multiple topic types. *Topic Type* could be seen like a typical class-instance relationship. Types represent the classes in which topics are grouped in, i.e., the category of one topic instance. Topic types are also topics (by standard definition).

A topic can have a name or more than one. However, topics do not have always names: a cross reference (e.g. - page 105), is considered to be a link to a topic that has no (explicit) name. The ability to specify more than one topic name can be used to name topics within different scopes, such as language, style, domain, geographical area, historical period, etc.

A topic can have one or more occurrences. One or more addressable information resources of a given topic, constitutes the set of *Topic Occurrences*. It might be a monograph devoted to a particular topic, for example, or an article about the topic in an encyclopedia; it could be a picture or video describing the topic, a simple mention of the topic in the context of something else, a commentary on the topic (if the topic were a law, say), or any of a lot of other forms in which an information resource might have some relevance to a topic [9]. A topic occurrence represents the information that is specified as relevant to a given subject.

Occurrences and topics exist on two different layers (domains), but they are "connected". Occurrences establish the routes from the topics to the information resources, enabling also to provide the reason why that route exist.

At this point it is very clear the separation in two layers of the topics and their occurrences, one of the great features of Topic Maps.

Among all occurrences of a given topic, a distinction can be made among subgroups. Each subgroup is defined by a common role. *Occurrence role* can be used to distinguish graphic from text, main occurrences from ordinary occurrences, mentions from definitions, etc. "The occurrence roles are user-definable and therefore can vary for each topic map" [2].

The standard also defines occurrence roles as topics. If an occurrence role is defined as a topic explicitly, topic map facilities can be used to say useful things about them (such as their names, and the relationships they participate in).

But to make the real distinction between different types of occurrences, Topic Maps uses also the concept of *occurrence role type*. This is different of the occurrence role in the sense that the last one is simply a mnemonic and the first one is a "reference to a topic in the map, which further characterizes the relevance of the role" [9].

The order used to specify topics and their associations is irrelevant; however, if necessary, a certain semantic order can be imposed.

Topic associations are almost ordinary links, except that they are constrained to only relate topics together. Because they are independent of the source documents in which topic occurrences are to be found, they represent a knowledge base, which contains the essence of the information that a someone is creating, and actually represents its essential value. An unlimited number of topics can be associated within "topic associations".

The power of topics maps increases with the creation of topic associations because that way, it is possible to group together a set of topics that are somehow related. This is of great importance in providing intuitive and user-friendly interfaces for navigating large pools of information.

As topic types group different kinds of topics and occurrences roles supports occurrences of different types, associations between topics can also be grouped according to their type (*Association Type*).

It is important to refer that each topic that participates in an association has a corresponding *association role* which states the role played by the topic in the association. Association roles are also regarded as topics in the topic map standard.

2.2 How to define a Topic Map

Before we start, we need to know exactly what we want to represent in the Topic Map. There are two phases to this: delimiting the scope of the TM (that is, deciding the extent of the domain it should cover); and designing the basic ontology. In TM terminology, an ontology is a precise description of the kinds of things that are found in the domain covered: in other words, the set of topics that are used to define classes of topics, associations, roles, and occurrences.

To illustrate all the ideas so far introduced and describe the TM building process, we will present a case-study when the subject is a university and its professors, research groups, departments, and courses. The scope can easily be extended to cover the students, the employees, the projects, etc. In the examples that follow, we will assume that a professor's name is *Pedro Rangel Henriques*, and that he is *doctor* and a *professor* at *Department of Informatics*, where he teaches courses in *LMCC* and *LESI* degrees. Also he is a member of *gEPL* research group. The basic ontology therefore consists of the topic types *Professor*, *Department*, *Course*, *Degree*, and *Group*, the association roles *works-at/employs*, *teaches/is-taught-by*, *has-title/is-title-of*, and *is-member-of/has-member*, and the association types *work*, *education*, *academic-title*, and *research*.

In the first step, we define the topics themselves (topic types and their instances), specifying their identifiers and base names. Below is an incomplete example:

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0"
       xmlns:xlink="http://www.w3.org/1999/xlink">
  <topic id="Professors">
    <baseName>
      <baseNameString>Professors</baseNameString>
    </baseName>
  </topic>
  <topic id="Pedro-Rangel-Henriques">
    <instanceOf>
      <topicRef xlink:href="#Professors"
                xlink:type="simple"/>
    </instanceOf>
    <baseName>
      <baseNameString>Pedro Rangel
          Henriques</baseNameString>
    </baseName>
  </topic>
```

```
</topicMap>
```

We only show the definition of the topic *Pedro-Rangel-Henriques* and its type *Professors*. The other topics and their types are created in a similar way.

In the next step, we add the occurrence definitions, using the element (*resourceRef*) that contains the URL (resource address) as the value for the attribute *xlink:href*. For instance, one occurrence for the topic *Pedro Rangel Henriques* is specified in XPath writing the path to the XML file used as the resource, as illustrated below.

```
<topic id="Pedro-Rangel-Henriques">
  ...
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#xpath"
                xlink:type="simple"/>
    </instanceOf>
    <resourceRef
xlink:href="/university[1]/professors[1]/prof[1]"/>
  </occurrence>
</topic>
```

Notice the use of *#xpath* as a *xlink:href*; it is possible because *xpath* is also a topic, more precisely it is a occurrence type.

The "..." in the code above stands for the definition of the topic *Pedro-Rangel-Henriques* as appeared in the previous specification fragment; we do not repeat it to keep the example as light as possible.

In the third step, we define the associations among topics, stating their type and their members (a topic with an explicit role). In the example below, we show the *work* association between *Department of Informatics* and *Pedro Rangel Henriques*. The first one *employs* the second that *works-at*.

```
<association>
  <instanceOf>
    <topicRef xlink:href="#work"/>
  </instanceOf>
  <member>
    <roleSpec>
      <topicRef xlink:href="#employs"/>
    </roleSpec>
    <topicRef
      xlink:href="#Department-of-Informatics"/>
  </member>
  <member>
    <roleSpec>
      <topicRef xlink:href="#works-at"/>
    </roleSpec>
    <topicRef
      xlink:href="#Pedro-Rangel-Henriques"/>
  </member>
</association>
```

The references *employs* and *works-at* are association role types, and they are declared like a topic type, i.e., with a identifier and a base-name only. The reference *work* is an association type, that defines the type of this association. The declaration of this topic is shown below:

```
<topic id="work">
  <baseName>
    <baseNameString>Work</baseNameString>
  </baseName>
  <baseName>
    <scope>
      <topicRef xlink:href="#employs"/>
    </scope>
    <baseNameString>employs</baseNameString>
  </baseName>
  <baseName>
    <scope>
      <topicRef xlink:href="#works-at"/>
    </scope>
    <baseNameString>works at</baseNameString>
  </baseName>
</topic>
```

The TM above explains that the Pedro Rangel Henriques works at Department of Informatics and, at the same time, indicates that the Department of Informatics employs Pedro Rangel Henriques.

3. THE TM BUILDER

After creating some Topic Maps by hand, it is easy to conclude that such task is time consuming and very repetitive. Thus gave us the idea to develop an Extractor of topics and relations from a set of XML resources, or by other words, a TM Builder.

In our context, a TM Builder is a converter from one XML language to another XML language. The TM Builder is a XSL stylesheet that receives an XML document as input and generates another XML file that contains a Topic Map. The reasons why the proposed TM Builder accepts XML documents as input are:

- XML is becoming the platform for information interchange;
- New data sources (non-XML) can be easily added to our extracting system just by using a translator to XML. Most of the actual information systems, like Database Management Systems, have facilities to dump their information in XML; so, for these cases the front-end is already there.

The main algorithm of the TM Builder[2] to extract a Topic Map from an XML document is shown below:

```
> Initially,
  for the given ontology creates all the:
  * topics types;
  * occurrences roles;
  * occurrences types;
  * associations types;
> During a document tree traversal,
  for each association, define the:
  * association type;
  * association members;
  for each element in the source that is seen
  as a topic, create the:
  * topic ID;
  * topic type;
  * topic names;
  * topic occurrences;
```

In our system, that algorithm was coded in a XSL stylesheet.

In practice we found that after the XSL processing, an XML Topic Map file will be generated. This Topic Map can have a problem: a set of topics with the same identifier.

This problem occurs when an element or attribute (that was defined as a topic) is found more than once in an input XML file. Each time that this element/attrubute is found, a new topic is created. So, many topics will be created with the same identifier. We must substitute all these topic definitions by just one definition: the identifier, topic type, and base name, shall appear once; different occurrence definitions must be created for each topic found.

To solve the problem, another XSL stylesheet was develop. This stylesheet is called when the first finishes the tree traversal; it will

[2]We assume that it contains the definition of the working ontology.

look for these problematic patterns in the generated XTM producing the final XTM file.

In the new stylesheet, for each set of topics with the same identifier, an unique topic will be created with the same topic type common to all, with the union of all individual occurrences as the occurrence set. All others topics are deleted.

Finally, we conclude that the Topic Map built by hand and partially listed in the subsection 2.2, could be automatically extracted by the two XSL stylesheets describe above (that constitute our TM Builder) from the XML source file below:

```
<?xml version="1.0" encoding="UTF-8"?>
<university>
  <dept>
    <name>Department of Informatics</name>
    <local>Campus de Gualtar</local>
  </dept>
  <professors>
    <prof title="Dr">
      <name>Pedro Rangel Henriques</name>
      <phone>4469</phone>
      <groupID>1</groupID>
    </prof>
    <prof title="Dr">
      <name>Jose Carlos Ramalho</name>
      <phone>4479</phone>
      <groupID>1</groupID>
    </prof>
    ...
  </professors>
  <courses>
    <course>
      <name>LESI</name>
      <class>MP1</class>
      <class>MP3</class>
    </course>
    <course>
      <name>LMCC</name>
      <class>PED1</class>
      <class>PL1</class>
    </course>
    ...
  </courses>
  <groups>
    <group>
      <id>1</id>
      <name>gEPL</name>
    </group>
    <group>
      <id>2</id>
      <name>gLMF</name>
    </group>
    ...
  </groups>
</university>
```

In the next section, we will introduce a language that can be used to specify the extraction process.

4. XSTM: AN XML LANGUAGE TO SPECIFY TOPIC MAP EXTRACTORS

The TM extractor discussed in the last section is tied to the structure of a specific XML source. The creation of a TM for an XML source with a different structure would imply the development of a new extractor. To solve this problem we have created an abstraction layer based on a new XML language: XSTM (XML Specification of Topic Maps).

The XSTM language supplies all the constructors that are needed to specify the extraction task, the Topic Map Builder process; it allows the definition of topics and their types and occurrences, as well as associations and their types and occurrence roles.

In a more formal way, we show below the CFG (Context Free Grammar) for that language:

```
xstm ::= topic+ topicType+ assoc* assocType*
         occurrenceRole*
topic ::= xpath TTypeID
topicType ::= TTypeID TTypeName
assoc ::= assocClass ATypeID LElem RElem
assocClass ::= "N2N" split=("true"|"false") |
      "one2one" type=("attribute"|"subelement") |
      "one2N" split=("true"|"false")
assocType ::= ATypeID ATypeName LElem RElem
LElem ::= TTypeID EName? ORoleID
RElem ::= TTypeID EName? Param? ORoleID
occurrenceRole ::= ORoleID ORoleName
```

Each XSTM specification is defined as an XML instance and the XSTM language is defined by a DTD and/or an XML-Schema.

Nowadays, XML-Schema has overcome the DTD approach for the definition of classes of the markup documents. We also made that upgrade; however, as XML-Schema is much more verbose than the correspondent DTD, we decided to include here the DTD and a respective XML-Schema. These diagrams (figure 1 to 8) were obtained with the XML Spy 5.0[3], from Altova.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT xstm (topicTypes, topics,
         occurrenceRoles?, assocTypes?, assocs?)>
<!ELEMENT topicTypes (topicType+)>
<!ELEMENT topicType (id, name)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT topics (topic+)>
<!ELEMENT topic (xpath, type)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT xpath (#PCDATA)>
<!ELEMENT occurrenceRoles (occurrenceRole+)>
<!ELEMENT occurrenceRole (id, name)>
<!ELEMENT assocTypes (assocType+)>
<!ELEMENT assocType (id, name, memberAssoc+)>
<!ELEMENT memberAssoc (scope, description)>
<!ELEMENT scope (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT assocs (one2one | one2N | N2N)+>
<!ELEMENT one2one (type, members11)>
<!ATTLIST one2one
type (attribute | subelement) #IMPLIED
>
<!ELEMENT one2N (type, members1N)>
<!ATTLIST one2N
split CDATA #IMPLIED
>
<!ELEMENT N2N (type, membersNN)>
<!ATTLIST N2N
split CDATA #IMPLIED
>

<!ELEMENT members11 (element, elementRef)>
<!ELEMENT members1N (one, N)>
<!ELEMENT membersNN (N)+>

<!ELEMENT element (topicAssoc, role)>
<!ELEMENT elementRef (topicAssoc, role)>
<!ATTLIST elementRef
target CDATA #IMPLIED
>
<!ELEMENT role (#PCDATA)>
<!ELEMENT one (topicAssoc, role)>
```

[3]http://www.xmlspy.com/

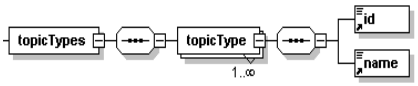


Figure 1: The topicTypes element.

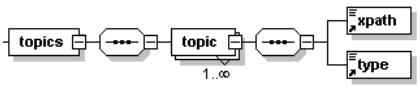


Figure 2: The topics element.

```
<!ELEMENT N (topicAssoc, role)>
<!ELEMENT topicAssoc (#PCDATA)>
<!ATTLIST topicAssoc
name CDATA #IMPLIED
id CDATA #IMPLIED
>
```

Notice that the XSTM DTD listed above is obtained direct and systematically from the CFG shown.

4.1 The *topicTypes* element

The *topicTypes* element is a sequence of *topicType* elements, where each one is a new topic type in the Topic Map. The topic type has an identifier and a name. The *id* element is the global identifier of this topic type in the Topic Map. This identifier will be referenced by others topics. The *name* element is the topic type name, that will be shown in a visualization of the Topic Map. Figure 1 shows the respective XML-Schema.

For instance, in our case-study the XSTM description of topic type is:

```
<topicTypes>
  <topicType>
    <id>Department</id>
    <name>Department</name>
  </topicType>
  <topicType>
    <id>Professor</id>
    <name>Professor</name>
  </topicType>
   <topicType>
    <id>Degree</id>
    <name>Degree</name>
  </topicType>
  ...
</topicTypes>
```

As was observed in the section 2, a topic type also is a topic. So, each topic type defined in XSTM language is transformed in a new topic if the new TM, which will be referenced by other topics (*instanceOf*).

4.2 The *topics* element

While topic types are abstract concepts defined by the ontology, topics are actual elements in the XML documents taken as input. To define them, we added the *topics* element, that is a sequence of *topic* element. This last one is a sequence of two required elements, as can be seen in the figure 2. The *xpath* element means the XPath expression that describes the path to the element that will be the topic, in the input XML document. The *type* element has a text content that specifies the identifier of its topic type.

Below we find the XSTM description of the topics for the case-study under consideration.

```
<topics>
  <topic>
    <xpath>//dept/name</xpath>
    <type>Department</type>
  </topic>
  <topic>
    <xpath>//prof/name</xpath>
    <type>Professor</type>
  </topic>
  <topic>
    <xpath>//prof/@title</xpath>
    <type>Degree</type>
  </topic>
  ...
</topics>
```

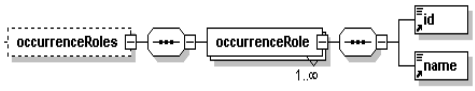


Figure 3: The occurrencesRoles element.

In this case we are selecting several XML elements at same time, through the use of an XPath expression. In the last section, we will discuss the gain we obtain from this kind of statement.

4.3 The *occurrenceRoles* element

The *occurrenceRoles* element defines the possible role types (once again an abstract concept) for the occurrences of each topic in an association. It has a child called *occurrenceRole* that is a sequence of an identifier and a name, as can be observed in the figure 3. It has the same structure of *topicType* element. The identifier (*id* element) can be referenced in an association and in a association type. The *name* element contains the name that will be displayed in the visualization of this Topic Map in a TM's application. Each occurrence role type will be a topic.

Below, there are the XSTM specification (we just show a fragment) of the *occurrence role types* in the ontology of our case-study.

```
<occurrenceRoles>
  <occurrenceRole>
    <id>works-at</id>
    <name>works at</name>
  </occurrenceRole>
  <occurrenceRole>
    <id>employs</id>
    <name>employs</name>
  </occurrenceRole>
  ...
</occurrenceRoles>
```

4.4 The *assocTypes* element

The *assocTypes* element allows to define another abstract concept in the ontology, the association types. It is composed by one or more of *assocType* that is a sequence of three elements (figure 4): an identifier (*id*) that will be referenced by an association; a name (*name*) that will be shown in a visualization of the Topic Map in a application like Omnigator[4] or TM-Design[5] and; and a member.

The *member* element means the member of the association and it has two child. The *scope* element specifies the reference to the topic that is the scope of this occurrence role. The other one, named *description*, is a name that will be displayed in a Topic Map's application.

[4]http://www.ontopia.net

[5]http://www.topicmap-design.com/en/topicmap-designer.htm

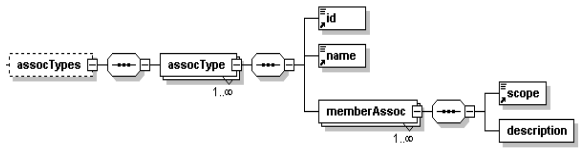


Figure 4: The assocTypes element.

An XSTM specification for *association types* can be seen below:

```
<assocTypes>
  <assocType>
    <id>work</id>
    <name>Work</name>
    <memberAssoc>
      <scope>employs</scope>
      <description>employs</description>
    </memberAssoc>
    <memberAssoc>
      <scope>works-at</scope>
      <description>works at</description>
    </memberAssoc>
  </assocType>
  ...
</assocTypes>
```

4.5 Relationships Types

The *assocs* element allows the specification of all actual associations that involve two or more topics and that can be found and extracted from the XML documents taken as input.

In the following, when we refer to relationships between tree nodes (XML elements and attributes) we are not talking about relations in the sense of the well-known entity-relationship model. So the usual names 1-to-1, N-to-1 and M-to-N, do not have exactly the same meaning used in that traditional perspective.

In our context, there are four kinds of relationships between elements, that are described by the three alternative children of the *assocs* element:

- associations between an element and its attribute. It is defined by the *one2one* element whose *type* attribute has the value *attribute*;
- associations between an element and a subelement referenced in the element's context. It is defined by the *one2one* element with the *subelement* value in the *type* attribute;
- associations one to N, defined by the *one2N* element;
- associations M to N, defined by the *M2N* element;

The *assocs* element contains specification of its type and all its members.

The *type* always means the association type (see the previous subsection), i.e., a reference to the identifier of the topic that represents its association type.

The members are a choice of *one2one*, *one2N*, or *M2N*, as shown in Figure 5; all of these elements have a similar structure: a sequence of a *type* and *membersXX* elements. The *XX* means the kind of relationship (1 to 1, 1 to N, M to N).

4.5.1 Relationships 1 to 1

This relationship represents two kinds of associations: between element and attributes, and between element and its subelement referenced.

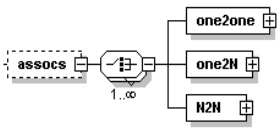


Figure 5: The assocs element.

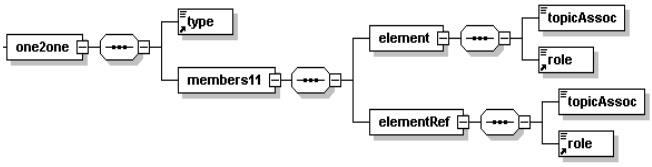


Figure 6: The one2one element.

The structure of the *one2one* element is a sequence of a association type and the members of this association, as can be seen in the figure 6. This element has an attribute named *type* that defines the kind of this association. The value of this attribute must be *attribute* or *subelement*.

The *members11* element (11 means 'one to one') is a sequence of *element* and *elementRef* elements. The first one specifies the main element in this relationship. The second one is an attribute or an subelement referenced by the first one.

The relationship between the topic types *Professor* and *Degree* illustrates a one2one relationship because actually it relates one element (*prof*) with one of its attributes (*title*). Below we present the XSTM specification for that association *academic-title*.

```
<one2one type="attribute">
  <type>academic-title</type>
  <members11>
    <element>
      <topicAssoc>Professor</topicAssoc>
      <role>has-title</role>
    </element>
    <elementRef>
      <topicAssoc>Degree</topicAssoc>
      <role>is-title-of</role>
    </elementRef>
  </members11>
</one2one>
```

We define the content of *type* attribute (of *one2one* element) as *attribute*, because the first one is an element and the second one is an attribute; the *element* specifies the topic *Professor* and its role, as well as the *elementRef* specifies the topic *Degree* and its role.

As said above, this kind of relationship also represents associations between an element and a subelement.

For instance, the relationship between the topic types *Professor* and *Group* illustrates another case of the one2one relationship because actually it relates one element (*prof*) with one of its subelement (*groupID*) that references the *group* element. Below we present the XSTM specification for that association *research*.

```
<one2one type="subelement">
  <type>research</type>
  <members11>
    <element>
      <topicAssoc>Professor</topicAssoc>
      <role>is-member-of</role>
    </element>
    <elementRef target="//prof/groupID">
```

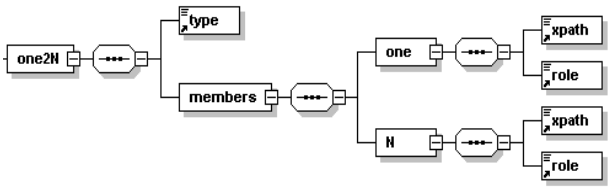


Figure 7: The one2N element.

```
      <topicAssoc name="./name" id="./id">
          Group</topicAssoc>
      <role>has-member</role>
    </elementRef>
  </members11>
</one2one>
```

We define the content of *type* attribute (of *one2one* element) as *subelement*, because the first one is an element and the second one is an subelement; the *element* specifies the topic *Professor* and its role, as well as the *elementRef* specifies the topic *Group* and its role.

In this case, there is one more attribute, *target*, of *elementRef*, whose value specifies the subelement (*//prof/groupID*) that makes the link between the two topics.

Notice that the element *topicAssoc* has two attributes: *name* to state the name to be shown by a visual tool; and *id* to specify the key of the index table.

4.5.2 Relationships 1 to N

This kind of relationship represents *one to many* associations between elements.

The *one2N* element is a sequence of an association type and the members of this association, as can be seen in the figure 7.

The *members1N* element (1N means 'one to N') is a sequence of *one* and *N* elements.

The *one2N* element has an attribute, named *split*, that specifies the format of this relationship. If the value of this attribute is `true`, it means that will be created an association for each occurrence of the second topic (referred in the *N* member) found in the input XML document. Otherwise, if the value is `false`, it means that will be created only one association grouping all the occurrences found for the second topic.

For instance, the association between the topic types *Department* and *Professor* illustrates a case of the *one2N* relationship, because actually it relates one element (*dept*) with multiple occurrences of the element *prof*. Below we present the XSTM specification for that association *work*.

```
<one2N split="true">
  <type>work</type>
  <members1N>
    <one>
      <topicAssoc>Department</topicAssoc>
      <role>employs</role>
    </one>
    <N>
      <topicAssoc>Professor</topicAssoc>
      <role>works-at</role>
    </N>
  </members1N>
</one2N>
```

4.5.3 Relationships M to N

This kind of relationship denotes *many to many* associations among elements.

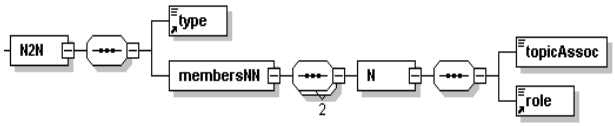


Figure 8: The M2N element.

The *M2N* element is a sequence of an association type and the members of this association, as can be seen in the figure 8.

The *membersMN* element (MN means 'M to N') is a sequence of two *N* elements.

The *M2N* element has an attribute, named *split*, that specifies the format of this relationship. This attribute has the same meaning as the one described in the previous relationship. If the value of this attribute is `true`, it means that will be created an association for each pair of topic occurrences (first or second *N* member) found in the input XML document (resulting MxN pairs). Otherwise, if the value is `false`, it means that will be created only one association grouping all the pair occurrences found for all the topics.

For instance, an association between the topic types *Professor* and *Course* illustrates a *M2N* relationship because actually it relates one element (*prof*) with the element *course*. Below we present the XSTM specification for the association *education*.

```
<M2N split="true">
  <type>education</type>
  <membersMN>
    <N>
      <topicAssoc>Professor</topicAssoc>
      <role>teaches</role>
    </N>
    <N>
      <topicAssoc>Course</topicAssoc>
      <role>is-taught-by</role>
    </N>
  </membersMN>
</M2N>
```

5. XSTM-P: AN XSTM PROCESSOR

The XSTM language, defined in the previous section, specifies the TM Builder process, enabling the systematic codification (in XSLT) of the extraction task.

In that circumstances we understood that it was possible to generate automatically the Extractor developing another XSL processor to translate an XSTM specification into the TM Builder code.

The XSTM processor (XSTM-P for short) is the TM Builder generator; it is one of the main pieces in our architecture, as can be seen in Figure 9. It takes a TM specification (an XML instance, written according to the XSTM language), and generates an XSL stylesheet that will process an input XML document to extract the Topic Map.

Both XSL stylesheets (the generator and the extractor) are processed by a standard XSL processor like Saxon[6] or Xalan[7], what in our opinion is one of the benefits of the proposal.

The main algorithm of the XSTM-P is now:

```
> Define the KEY tables, to create
  associations from cross references.
> During the tree traversal:
  * for each topic type: create xstm:topic;
  * for each occurrence role: create
```

[6] http://saxon.sourceforge.net/
[7] http://xml.apache.org/xalan-j/

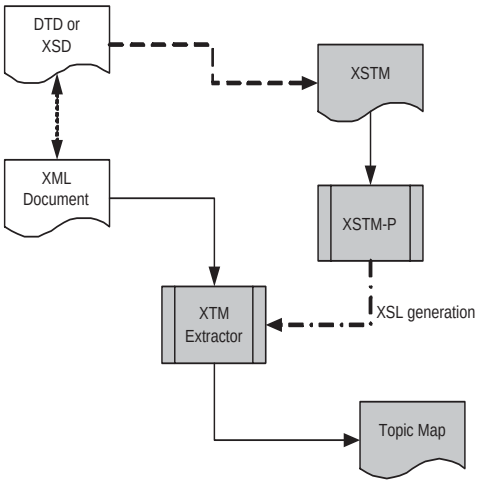


Figure 9: The XSTM architecture.

```
    xstm:topic;
  * create the occurrence types;
  * for each association type: create
    xstm:topic, which includes the members
    of this association type;
> For each topic defined in XSTM file:
  * create a xstm:for-each to each element
    with the path to each one;
> For each association defined in XSTM file:
  * create the association among all members:
    - defined in one2one;
    - defined in one2N;
    - defined in M2N;
```

After processing the complete specification for the case-study under work, a XSTM description with 194 lines, we produced a TM Builder that is a file with 528 XSLT lines. The Topic Map extracted from a source file with 342 lines is a XTM file with 4205 lines[8] with 98 topics and 173 associations.

6. CONCLUSION

Looking at a TM we can think of it as having two distinct parts: an ontology and an object catalog. The ontology is defined by what we have been designating as topic type, association type, and occurrence role type. The catalog is composed by a set of information objects that are present in information resources (one object can have multiples occurrences in the information resource) and that are linked to the ontology. Figure 10 gives a schematic representation of this vision.

In XSTM, the ontology definition takes the same effort to specify as in XTM, we have to specify every single topic type, association type, and occurrence role type. However, in XTM everything is a topic. XSTM further classifies those topics, giving them a more concrete semantics, naming them topic type, association type, or occurrence role type. So, for the ontology part the gain is achieved through a more precise semantics.

For the catalog, the situation is completely different. In our topic and association specifications we use XPath expressions that act like queries. This way the gain we obtain is equal to the number of occurrences retrieved by the query expression.

In the case of the associations the gain is even higher: N for the 1:N relations and MxN for the M:N relations.

[8]The files can be found in http://alfa.di.uminho.pt/ grl/xstm

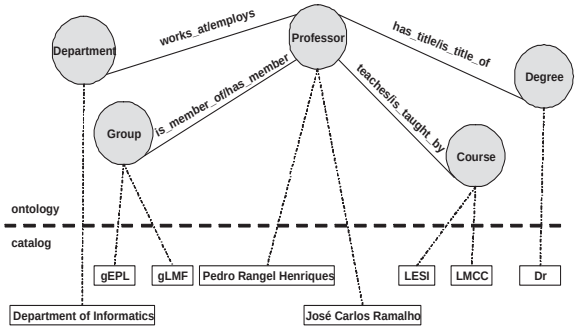


Figure 10: The case-study's ontology and catalog.

7. REFERENCES

[1] K. Ahmed, D. Ayers, M. Birbeck, J. Cousins, D. Dodds, J. Lubell, M. Nic, D. Rivers-Moore, A. Watt, R. Worden, and A. Wrightson. *Professional XML Meta Data*. Wrox Programmer to Programmer Series, 2001.

[2] M. Biezunski and S. R. Newcomb. Topic Maps Frequently Asked Questions. www.infoloom.com, September, 1999.

[3] N. Bradley. *The XML Companion*. Addison-Wesley, 3rd edition, 2002.

[4] J. Duckett, O. Griffin, S. Mohr, F. Norton, N. Ozu, I. Stokes-Rees, J. Tennison, K. Williams, and K. Cagle. *Professional XML Schemas*. Wrox Press, 2001.

[5] C. F. Goldfarb and P. Prescod. *XML Handbook*. Prentice Hall, 4th edition, 2001.

[6] E. Harold and W. Means. *XML in a Nutshell*. O'Reilly & Associates, 2001.

[7] S. Holzner. *Inside XML*. New Riders Publishing, 1st edition, 2000.

[8] J. Park and S. Hunting. *XML Topic Maps: Creating and Using Topic Maps for the Web*. Prentice Hall, 2003.

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

HaQuery: Uma Biblioteca de Combinadores para Interrogar XML

David Costa, António Faria and João Saraiva

Departamento de Informática,
Universidade do Minho
Portugal

Resumo Este documento apresenta *HaQuery*: uma biblioteca de combinadores definidos na linguagem *Haskell* para interrogar documentos *Xml*. A própria biblioteca *HaQuery* está construída utilizando uma abordagem puramente combinatorial, recorrendo para tal a combinadores de parsing e *Xml*, ambos *standard* em *Haskell*. Neste documento apresentamos uma breve descrição da construção da biblioteca, e alguns exemplos da sua utilização.
Uma especificação formal de *HaQuery* foi ainda escrita no formalismo de gramáticas de atributos e uma versão interactiva e incremental desta biblioteca foi automaticamente construída pelo sistema LRC. Neste documento apresentamos um exemplo onde se embebeu o *HaQuery* de modo a construir uma poderosa linguagem de interrogação para uma folha de cálculo previamente definida.

1 Introdução

Este documento apresenta *HaQuery*: uma biblioteca de combinadores definidos na linguagem *Haskell* para interrogar documentos *Xml*. O desenho da linguagem *HaQuery* baseia-se bastante em *XQuery*, uma linguagem tipada e funcional para interrogar *Xml*, que está presentemente a ser desenvolvida pelo *Xml Query Working Group* do *World-Wide Web Consortium* (W3C) [Dra02,Wad02].

Para introduzir *HaQuery*, vamos utilizar alguns exemplos de interrogações efectuadas sobre um documento *Xml* apropriado. Mais concretamente, consideramos um documento que contém informação que descreve um curso universitário. Isto é, o documento descreve o nome do curso, o ano em que ocorre no seu programa, a informação referente aos professores e ainda informação sobre os alunos inscritos para frequentar o curso. Para listar os alunos registados no curso, teremos de escrever a seguinte frase (*i.e.*, interrogação) em *HaQuery*:

```
Q: /curso/cadeira/inscritos ?
```

Se pretendermos listar os estudantes que frequentam o terceiro ano (ou superior) ou ainda os estudantes cujo último nome é *Atento* e estão registados como *TE* (trabalhadores estudantes), então poderemos escrever em *HaQuery*:

```
Q: //aluno[ @ano .>=. 3
          | (./nome/@sobrenome='Atento' & ~(@estatuto = 'TE'))
          ] ?
```

HaQuery permite ainda formular interrogações mais complexas, como por exemplo, o aninhamento de interrogações e efectuar interrogações sobre um conjunto de documentos *Xml* e não sobre um documento apenas (como nos exemplos acima que usa um documento por defeito. Este documento, bem como o resultado desta interrogação estão apresentados no apêndice A e na secção 2, repectivamente).

A biblioteca *HaQuery* está construída utilizando uma abordagem puramente combinatorial: a análise sintáctica de *HaQuery* (*i.e.*, a análise sintactica da sua notação concreta) está definida através de combinadores de parsing *standard* (tal como definidos em [Sar02b]), enquanto que para a sua interpretação semântica recorremos à biblioteca *HaXml*: os combinadores de *Xml* embedidos em *Haskell* [WR99]. Embora utilizando um paradígma de programação diferente, a construção deste processador de *HaQuery* utiliza a arquitectura clássica de um processador/interpretador de linguagem: primeiro um parser (baseado em combinadores) valida a entrada e constrói uma árvore que a representa abstractamente. Posteriormente, esta árvore é atravessada (*i.e.*, decorada) de modo a determinar o seu significado: o resultado da interrogação, neste caso. Uma das vantagens de usarmos *HaXml* é o facto de esta biblioteca possuir já todo um conjunto de funções genéricas para manipular *Xml*, *e.g.*, efectuar o parsing the *Xml* e DTDs, filtrar e transformar *Xml*, *pretty printing*, etc, que são as fundações da nossa biblioteca de interrogação. Como resultado desta abordagem, o esforço para desenvolver *HaQuery* (e agora a sua manutenção e actualização) foi bastante simples e conciso: todas as funções necessárias para definir a nossa linguagem de interrogação recorrem aos combinadores pre-definidos em *HaXml*. Assim, não foi necessário desenvolver funções inductivas (complexas) que atravessam a árvore abstracto que representa o documento *Xml*. Como resultado, a biblioteca completa está escrita em apenas 300 linhas de código *Haskell*.

As gramática de atributos são um formalismo declarativo para definir computações em árvores. Mais recentemente, as gramáticas de atributos são vistas como um paradígma de programação com a expressividade de uma linguagem funcional *lazy* (*e.g.*, *Haskell*). As gramáticas de atributos, porém, tem várias vantagens quando comparadas com programação *lazy*, nomeadamente: a detecção estática de circularidades e a possibilidade de executar essas especificações utilizando diferentes modelos de computação (por exemplo, cálculo estrito, lazy e incremental). Para tirar partido destas vantagens das gramáticas de atributos, a biblioteca *HaQuery* está também definida através de uma gramática de atributos definida no sistema Lrc. O sistema Lrc [KS98] é um gerador de ferramentas interactivas e incrementais baseado no formalismo de gramáticas de atributos. Utilizando o Lrc obtemos assim um sistema interactivo e incremental para interrogar documentos *Xml*: o sistema *LrcQuery* [FCS]. Por último, uma vez que o sistema Lrc produz calculadores de atributos nas linguagens C (incremental

ou não), *Haskell* (lazy, estrito e deflorestado) e OCaml, uma versão de *HaQuery* foi também produzida para estas seis representações.

As bibliotecas *HaQuery* e *LrcQuery* foram construídas e estão desponíveis para serem utilizadas pela comunidade científica.

Este documento está organizado da seguinte forma: na secção 2 apresentamos a linguagem *HaQuery*, primeiro começamos por definir alguns exemplos (secção 2.1) e posteriormente mostramos um exemplo da utilização do interpretador desenvolvido (seccão 2.5). Na secção 2.6 apresentamos os combinadores de parsing para *HaQuery*. Na secção 2.7 descrevemos os combinadores de *Xml* para *Haskell*, enquanto que na secção 2.8 apresentamos os combinadores de *HaQuery*. Na secção 3 apresenta-se sucintamente o sistema *LrcQuery*. A secção 4 contém as conclusões e em apêndice inclui-se o documento *Xml* usado como exemplo.

2 *HaQuery*: Combinadores *Haskell* para Interrogar *Xml*

A linguagem *HaQuery* baseia-se bastante em *XQuery*, uma linguagem de interrogação *Xml*. A linguagem *XQuery* está a ser desenvolvida pelo *Xml Query Working Group* com a intensão de ser uma linguagem pequena, simples de processar e em que as interrogações são concisas e de fácil compreensão. As mesmas ideais estão presentes no desenvolvimento de *HaQuery*. De facto, *HaQuery* pretende ser uma primeira modelação de *XQuery* em *Haskell* recorrendo para tal a técnicas avançadas de construção de processadores de linguagens no paradígma de programação funcional.

O *HaQuery* destina-se a ser utilizado como uma linguagem de extracção de informação de um documento *Xml*. Deste modo, todas as interrogações produzem como resultado um documento *Xml*. Antes de descrevermos como foi construída esta biblioteca, vamos primeiro introduzir a linguagem, recorrendo para tal a exemplos de possíveis interrogações.

2.1 *HaQuery* via Exemplos

Considere de novo o documento *Xml* com informação sobre um curso universitário (ver apêndice) e que pretendemos listar todos os alunos inscritos no curso aí definido. Para efectuarmos esta interrogação, tem de ser-nos possível referir um qualquer ponto de um documento *Xml*. Assim, *HaQuery* possuí um mecanismo para referenciar estruturas hierarquicas, em tudo idêntico à bem conhecida notação de referenciação hierarquica do sistema de ficheiros de Unix. Porém, em vez de serem nome de directorias a indicar o caminho, usamos em *HaQuery* o nome das etiquetas dos elementos contidos no documento *Xml*. O separador / indica uma relação de parentesco entre nodo pai e nodos filhos. Assim, para efectuar esta interrogação teremos de especificar todo o caminho da raíz do documento até ao fragmento que contém a informação sobre inscrições. Em *HaQuery* as interrogações iniciam-se pelas letras `Q:` e são terminadas pelo sinal de interrogação. Esta interrogação em *HaQuery* escreve-se do seguinte modo:

```
Q: /curso/cadeira/inscritos ?
```

Esta interrogação selecciona os elementos `inscritos` dentro dos elementos `cadeira` que instanciaram dentro do elemento raiz `curso`. O asterisco * pode ser utilizado como "joker" para instanciar com qualquer elemento etiquetado, daí que:

```
Q: /curso/*/inscritos ?
```

selecciona todos os elementos `inscritos` que estejam dentro de um qualquer elemento que seja descendente directo de `curso` . O asterisco pode ser utilizado também como sufixo e prefixo para instanciar um conjunto de elementos anotados[1]

```
Q: /curso/c* ?
```

Esta interrogação selecciona todos os elementos descendentes de `curso` cujo nome se inicia por **'c'**. Uma dupla barra `"//"` permite uma busca recursiva por todos os elementos anotados. Assim:

```
Q: /curso//inscritos ?
```

seleciona todos os elementos `inscritos` descendentes directos ou indirectos, a qualquer profundidade do elemento raíz `curso`. Para selecionar o conteúdo textual entre um elemento anotado utiliza-se o *ífen* `-`:

```
Q: /curso/nome/- ?
```

que instância com conteúdo textual dentro do elemento anotado `nome`. Utilizando o *asterisco*, `*` em conjunto com os operadores `//` e *ífen*, `-`, podemos escrever:

```
Q: /*//- ?
```

que selecciona o conteúdo textual de todos os elementos anotados contidos no documento *Xml*. Na prática o que se obtém é o documento *Xml* desprovido de todas as etiquetas, obtendo-se assim apenas o texto contido entre as anotações.

A união de duas interrogações é expressa através do operador + que pode obrigar ao uso de parenteses:

```
Q: /curso/cadeira/(docentes + inscritos) ?
```

que seleciona os elementos `docentes` e `inscritos`, ambos contidos no elemento `cadeira`. Finalmente,

```
Q: /curso//aluno/@estatuto ?
```

devolve o valor do atributo `estatuto` do(s) elemento(s) `aluno` selecionado(s), isto se o atributo existir.

[1] De notar que não estão implementadas todas as funcionalidades das expressões regulares habitualmente associadas ao "joker"asterisco.

2.2 Predicados

Existe a noção de predicado num elemento que é dada pela utilização dos parênteses rectos. Assim a interrogação:

```
Q: /curso/cadeira[./tps] ?
```

seleciona todos os elementos `cadeira` que contenham pelo menos um elemento `tps`. Querendo atestar da existência de um atributo em particular fazemos, por exemplo:

```
Q: /*//aluno[@estatuto] ?
```

que vai seleccionar todos os elementos `aluno` que tenham o atributo `estatuto` definido. Existe também a possibilidade de comparar valores de atributos com os operadores =, ! =, <, <=, > e >= que comparam dois valores do tipo alfa-numérico. Uma comparação só pode ser efectuada entre dois atributos ou entre o valor um atributo e uma `string`, no entanto quando efectuamos uma comparação entre o valor de um atributo e uma "string"essa tem de aparecer do lado direito do operador.

```
Q: /*/cadeira[./tps/@natureza='obrigatoria'] ?
```

Esta interrogação selecciona todos os elementos `cadeira` que possuam o argumento `natureza` definido com o valor `obrigatoria`. Se a comparacao alfa-numérica for inapropriada, é possivel fazer a comparacao numérica[2]. Os operadores numéricos são os mesmos que os operadores de comparação alfa-numerica só que têm um ponto antes e depois do operador, .=. , .! =. , .<. , .<=. , .>. , .>=., por exemplo:

```
Q: //cadeira[./avaliacao[@aprovados .>. @reprovados]] ?
Q: //cadeira[./avaliacao[@aprovados .>. @reprovados]]/designacao/- ?
```

na primeira interrogação obtemos todos os elementos `cadeira` que possuam como descendente directo pelo menos um elemento `cadeira` que por sua vez tenha os atributos aprovados e reprovados definidos e que o valor numérico do atributo aprovados seja maior que o valor numérico do atributo reprovados. Na segunda interrogação obtemos apenas o conteúdo textual do elemento `designação` descendente directo de cada um dos elementos `cadeira` obtidos na interrogação anterior.

Para além de se poder comparar o valor dos atributos também é possivel comparar o conteúdo textual de um elemento. Por exemplo, a interrogação seguinte selecciona todos os elementos `aluno` cujo o conteudo textual do elemento `nome` seja igual a `Zezinho`.

```
Q: //aluno[./nome/- = 'Zezinho'] ?
```

[2] Esta característica menos elegante do *HaQuery* é devida a uma limitação séria do *Xml*. De facto, só é possível determinar o tipo de um atributo (por exemplo, string ou inteiro) após a validação do documento com um *Xml* Schema. Para não limitarmos o uso de *HaQuery* a interrogar documentos munidos do seu Schema apenas, optamos por considerar operadores diferentes.

2.3 Selecção Posicional

Uma outra funcionalidade que a ferramenta *HaQuery* oferece é que a notação dos parêntesis rectos permite também fazer a selecção de elementos através da sua posição, por exemplo:

```
Q: //aluno[3] ?
```

que vai selecionar o terceiro elemento `aluno` da lista de aqueles que foram selecionados.

É possivel também ter uma lista de posições separadas por vírgulas ou mesmo intervalos em que o limite superior e inferior é separardo por um *ífen*, `-`. O simbolo *dólar*, **$** deve ser usado quando se pretende referir a úlima posição. Por exemplo:

```
Q: //aluno[0,2-\$,1] ?
```

De todos os elementos `aluno` do documento irá selecionar o que surge em primeiro, de seguida aprecerá a lista do terceiro até ao último e por fim o segundo, retornando-os por esta mesma ordem.

2.4 Combinação de Predicados

Os predicados podem ser combinados usando os mais comuns operadores booleanos **&**, | e $\sim$ e os parêntesis podem ser usados para desambiguar. Por exemplo, a interrogação que se apresenta a lista todos os estudantes que frequentam o terceiro ano (ou superior) ou ainda os estudantes cujo último nome é *Atento* e estão registados como *TE*.

```
Q: //aluno[ @ano .>=. 3
          | (./nome/@sobrenome='Atento' & ~(@estatuto = 'TE'))
          ] ?
```

2.5 *HaQuery* a Trabalhar

Nas secções anteriores foram apresentadas várias formas de efectuar interrogações em *HaQuery*. Antes de iniciarmos a descrição da construção do processador/interpretador de *HaQuery*, vamos primeiro apresentar um exemplo de utilização deste processador.

Para efectuarmos uma interrogação num documento, foi desenvolvida uma função *Haskell*, chamada `query`, que recebe dois argumentos: a interrogação (uma string) e o documento *Xml* a interrogar (ou mais precisamente o ficheiro onde este se encontra). O resultado da interrogação é enviado para o *output* como um documento *Xml pretty-printed*. A seguir apresentamos a execução da interrogação anterior a ser executada no interpretador da *Haskell Hugs*.

```
Main> query "Q: //aluno[@ano.>=.3|(./nome/@sobrenome='Atento' & ~(@estatuto='TE'))] ?"
            "teste.xml"
<HaQuery query="Q: //aluno[@ano.>=.3|(./nome/@sobrenome='Atento' & ~(@estatuto='TE'))] ?"
         file="teste.xml"
  ><aluno n="40000" ano="3" matriculas="5" estatuto="ORD"
    ><nome sobrenome="Grossa"
      >Nelita Coxa</nome></aluno
  ><aluno n="44000" ano="3" matriculas="4" estatuto="ORD"
    ><nome sobrenome="Muito"
      >Mane andaquia</nome></aluno
  ><aluno n="55556" ano="2" matriculas="2" estatuto="ORD"
    ><nome sobrenome="Atento"
      >Zezinho</nome></aluno>
</HaQuery>
```

2.6 *HaQuery*: Parsing com Combinadores e Gramática Abstracta

Para construirmos o processador/interpretador de *HaQuery*, *i.e.*, a função *query*, utilizamos a arquitectura clássica de um compilador. Mais concretamente, construimos uma parser para a notação concreta de *HaQuery*, que constrói (virtualmente) uma árvore de sintaxe abstracta. Esta é posteriormente decorada de modo a ser interpretada a interrogação.

Para a construção do parser utilizamos os combinadores de parsing apresentados em [Sar02b]. De seguida, apresentamos um fragmento deste parser. Recorde que uma interrogação em *HaQuery* começa pelos caracteres `Q:` (modelado pelo combinador `token "Q:"`), seguido da interrogação propriamente dita (sub-parser *aquery*) e, por fim, pelo caracter `?` (modelado pelo combinador `symbol '?'`). O combinador `<*>` modela a sequenciação, enquanto o combinador `<|>` modela o operador *ou* em BNF. Tipicamente, um parser executa acções semânticas durante o processamento de entrada. O combinador `<$>` modela a aplicação de uma função semântica.

```
query = prodquery <$> token "Q:" <*> aquery <*> symbol '?'
  where
     prodquery _ b _ = ProdQuery b

aquery =  currentcontext      <$> token "./" <*> tquery
      <|> rootcontext         <$> symbol '/' <*> tquery
      <|> deepcontextfromroot <$> token "//" <*> tquery
     where
       currentcontext      _ b = CurrentContext      b
       rootcontext         _ b = RootContext         b
       deepcontextfromroot _ b = DeepContextFromRoot b
```

Observe que pelo facto dos combinadores de parser terem nomes semelhantes à notação BNF que se pretende modelar, o programa *Haskell* "é" também a gramática.

As acções semânticas do parser constroem uma representação abstracta da interrogação. A estrutura (ou sintaxe) abstracta da linguagem *HaQuery* (ou mais genericamente de qualquer linguagem) é facilmente definida através de tipos de dados algébricos em *Haskell*. A seguir apresentam-se três tipos de dados *Haskell*: *Query*, *AQuery* e *TQuery*, que definem parte da estrutura abstracta de *HaQuery*. Onde $ProdQuery$, $CurrentContext$ e $ProdBracketTQ$ são funções que constroem valores desses tipos.

```
data Query  = ProdQuery AQuery
data AQuery = CurrentContext TQuery
            | RootContext TQuery
            | DeepContextFromRoot TQuery
data TQuery = ProdBracketTQ TQuery XQuery
            | ProdTag Tag XQuery
```

Observe que as acções semânticas do parser apresentado anteriormente limitam-se a chamar as funções construtoras da estrututa abstracta de *HaQuery*. De facto, o resultado de um parser é uma árvore sintáctica[3].

Após efectuarmos o parsing the *HaQuery*, é necessário interpretar a interrogação num (ou mais) documento(s) *Xml*. Isto significa que este intepretador (*i.e.*, a função *query*, apresentada anteriormente) tem de manipular documentos *Xml*. Uma vez que existem já bibliotecas que manipulam *Xml* em *Haskell*, *e.g.HaXml*, todo o processo de interpretação de interrogações vai ser feito utilizando essa biblioteca. O *HaXml* apresenta-se de seguida.

2.7 Combinadores de *Xml*: *HaXml*

O *HaXml* é uma biblioteca de combinadores que permite manipular *Xml* em *Haskell*. Esta biblioteca inclui funções que permitem:

- um parser de *Xml*,
- um parser error-correcting para *Html*,
- validadar *Xml*,
- pretty-printers para *Xml*, etc

As linguagens funcionais são conhecidas por serem bastante eficientes para especificar estruturas em forma de árvore e definir as funções que as manipulam. Por exemplo, a estrutura (em árvore) de um documento *Xml* é definida em *Haskell* pelo tipo de dados *Document*. Este tipo está definido na biblioteca *HaXml*:

[3] Em processadores de linguagens simples a construção desta árvore pode não ser necessário, obtendo-se assim um processador mais eficiente. No caso do processador de *HaQuery*, isso é obtido automaticamente através do uso de técnicas de deflorestação existentes no compilador de *Haskell* `ghc` e no sistema de gramáticas de atributos LRC.

```
data Document  = Document Prolog (SymTab EntityDef) Element
...
data Element   = Elem Name [Attribute] [Content]
data Content   = CElem Element
               | CString String
```

A biblioteca *HaXml* é constituída por um vasto conjunto de funções/combinadores que manipulam termos do tipo *Document*. Basicamente, estas funções filtram documentos *Xml*. Por exemplo, existem filtros que permitem seleccionar elementos dada a sua posição, ou devolver os descendents de um elemento, etc. Todos estes filtros partilham o mesmo tipo, isto é, recebem como argumento o conteúdo (tipo *Content*) de um documento *Xml* (que pode ser um pedaço de texto ou um elemento anotado completo) e filtram esse documento, produzindo como resultado uma lista de conteúdos (tipo [*Content*]) que contém os fragmentos que obedecem ao filtro definido (a lista vazia modela o facto de nenhum fragmento ser filtrado). O tipo destes filtros é definido em *HaXml* pelo tipo $CFilter$:

```
type CFilter = Content → [Content]   — definido em XmlCombinators.hs
```

Estas funções podem ainda ser vistas como *predicados*, possibilitando fazer decisões sobre o conteúdo dos seus argumentos de entrada. A seguir apresentam-se o tipo de alguns destes filtros.

```
position :: Int → CFilter → CFilter          — seleciona o elemento duma dada posição
tagWith  :: (String → Bool) → CFilter        — aplica um predicado ao nome de um elemento
deepest  :: CFilter → CFilter                — busca recursiva (em profundidade)
children :: CFilter                          — Retorna os descendentes de um elemento
o        :: CFilter → CFilter → CFilter      — permite composição de dois filtros
txt      :: CFilter                          — predicado que testa se um elemento tem
                                             — conteúdo textual. Retorna esse mesmo conteúdo
```

Para demonstrar como se podem escrever facilmente processadores de *Xml* em *HaXml*, vamos construir uma função que utiliza alguns destes combinadores. A função *processXMLwith* aplica um filtro que recebe como parâmetro (parâmetro *filter*) a um documento *Xml* pre-definido *teste.xml* (para simplicar a apresentação não o passamos como argumento). Esta função faz o seguinte: primeiro lê o documento *Xml* do ficheiro e efectua o seu parsing, utilizando a função de parser incluída na biblioteca *xmlParse*[4]. O parser de *Xml* devolve um termo do tipo *Document*. Uma vez que os filtros manipulam termos do tipo *Content*, a função extrai o conteúdo do documento, e aplica-lhe o filtro recebido como argumento. Por fim, envia para a saída uma versão *pretty print* do resultado de aplicar o filtro (funções *render* e *vcat*).

```
getElem (Document _ _ e) = CElem e
```

[4] Parser este também definido com combinadores de parsing.

```
processXMLwith :: CFilter → IO ()
processXMLwith filter =
 do
   xmlInput ←readFile "teste.xml"          — read de Xml source file
   let xml   = xmlParse xmlFile xmlInput   — parsing the Xml file to Xml type
   let cont  = getElem xml                 — select Content
   let res   = filter cont
   putStr    $  render . vcat . map content $ res
```

De seguida mostram-se dois exemplos de utilização desta função. O primeiro exemplo selecciona o elemento na posição 0 do documento. O segundo exemplo selecciona o texto dos filhos a maior profundidade.

```
CombTeste>  processXMLwith (position 0 children)
<nome>Matemática e Ciências da Computação</nome>

CombTeste> processXMLwith (deepest (txt 'o' children)
Matemática e Ciências da Computação
Álgebra Linear
Lurdes
lurdes@math.di.uminho.pt
Lurdes
...
```

2.8 *HaQuery*: Combinadores

Na secção anterior, vimos como é possível utilizar os combinadores da biblioteca *HaQuery* quer para manipular directamente *Xml*, quer para construirmos as nossas próprias funções (*e.g.*, *processXMLwith*). Vamos agora ver como podemos utilizar estes mesmos combinadores para modelar o interpretador de interrogações *HaQuery*.

Antes de definirmos as funções que modelam este interpretador, vamos definir primeiro qual o tipo do filtro que estas interrogações exprimem. Como vimos na secção2.1, numa interrogação podemos seleccionar um ponto qualquer no documento *Xml*. Assim, os filtros de interrogação manipulam não só todo o conteúdo do documento, mas também o conteúdo do ponto seleccionado. Logo, estes filtros recebem esses dois conteúdos como argumento. Este tipo de filtro $DFilter$ está definido a seguir.

```
type DFilter = Content → Content → [Content]   — definido em HaQuery.hs
```

O interpretador de *HaQuery* é definido como uma função inductiva sobre o tipo de dados *Query*. Esta função escreve-se elegantemente utilizando concordância de padrões em *Haskell*. Para cada constructor do tipo de dados *Query* (e tipos que este deriva) introduzimos uma definição alternativa da função. Parte destas funções apresentam-se de seguida.

```
haQueryProc :: Query → DFilter
haQueryProc (ProdQuery aq) = aqProcessor (global keep) aq
aqProcessor :: DFilter → AQuery → DFilter
aqProcessor localise (CurrentContext tq) = tqProcessor [(// >>) localise] tq
aqProcessor _ (RootContext tq) = tqProcessor [oglobo id] tq
aqProcessor _ (DeepContextFromRoot tq) = tqProcessor [oglobo deep] tq
```

em que as funções (*i.e.*,combinadores) *global*, *keep*, *oglobo*, *deep* e // >> são combinadores *HaXml* (ou pequenas adaptações desses combinadores para manipularem o tipo $DFilter$ e não apenas o tipo $CFilter$). Neste documento omitimos a definição destes combinadores uma vez que nos levaria a uma discussão muito especifica da linguagem *Haskell*, o que está fora do contexto deste documento.

A função *query* define-se facilmente utilizando as funções apresentadas anteriormente.

```
query :: [Char] → [Char] − → IO ()
query q xmlFile =
 do
    let query = haQueryParse q
    let xml = xmlParse xmlFile xmlInput
    let xmlCont = getElem xml
    let result = haQueryProc query xmlCont xmlCont
    (hPutStrLn stdout . render . vcat . map content) result
```

3 LrcQuery: Um sistem Interactivo e Incremental para Interrogar Linguagens

As gramáticas de atributos, tal como a programação funcional, são um formalismo declarativo bastante indicado para especificar computações em estruturas em árvore. Embora no início as gramáticas de atributos fossem usadas para especificar e contruir compiladores. Compiladores estes que executavam em modo *batch*. Recentemente, as gramáticas de atributos são usadas não só para construir compiladores, mas também para construir outras ferramentas, tais como editores estruturados [RT89], ambientes de programação [KS98], linguagens visuais [KS02], algoritmos de *pretty-printing* complexos [SAS99], animação de programas [MLAŽ02,Sar02a], etc.

Mais recentemente ainda as gramáticas de atributos são vistas como um formalismo indicado para escrever programas funcionais *lazy* [dMPJvW99,dMBS00]. As gramáticas de atributos têm várias vantagens quando comparadas com programação *lazy* [Sar99], nomeadamente na detecção estática de circularidades e na possibilidade de executar essas especificações utilizando diferentes modelos de computação (por exemplo, cálculo estrito, lazy e incremental). Uma outra característica importante é o facto de se poderem derivar ambientes de programação

a partir de uma gramática de atributos. Este ambientes permitem ao utilizador poderosas formas de interacção com a ferramenta.

Para tirarmos partido de todos estas vantagens das gramáticas de atributos, apresentamos um breve exemplo do sistema *LrcQuery*: a modelação de *HaQuery* no sistema de gramáticas de atributos Lrc.

O sistema *LrcQuery* está especificado como uma gramática de atributos genérica. Isto é, *LrcQuery* está definido por uma gramática de atributos que pode ser embebida em qualquer outra gramática de atributos que especifica formalmente uma qualquer linguagem formal. Por exemplo, pode ser embebida numa gramática de atributos que especifica uma base de dados bibliográfica (*e.g.*, bibTeX) para, por exemplo, interrogarmos a base de dados sobre quais os livros escritos por um autor. Ou pode ser embebida numa folha de cálculo para interrogar sobre quais os registos em que um dado campo é maior que um dado valor, etc

Para embebermos *HaQuery* no paradígma de gramáticas de atributos, seguimos os seguintes passos:

- Primeiro, a estrutura abstracta de *HaQuery*, foi modelada em gramáticas de atributos na linguagem SSL [RT89]: a linguagem utilizada pelo sistema Lrc. A semântica (estática) de *HaQuery* foi especificada através da definição de atributos e suas equações, tal como é feito tradicionalmente neste paradígma de programação. Deste modo, a gramática de atributos resultante é uma componente de software genérica para interrogação no contexto de gramáticas de atributos.
- Segundo, re-utilizamos os combinadores de *Xml* pre-definidos em Lrc, para mapear uma qualquer gramática abstracta para a gramática abstracta do *Xml*. Por outras, palavras, mapeamos a árvore abstracta de linguagem em consideração para uma árvore *Xml* (ver tipo `Document` apresentado a seguir).
- Definimos um atributos de ordem superior para modelar a cola de componentes no contexto de gramáticas de atributos, tal como definido em [Sar02a].
- Por último, definimos atributos e suas regras de cálculo para a construção de um interface interactivo avançado para efectuar as interrogações. Por exemplo, este interface apresenta ao utilizador uma visão da interrogação bem formatada e alindada, bem como guia o utilizador a formular interrogações correctas. Da facto, o ambiente construído, para além de proporcionar ao utilizador um editor tradicional, possuí um editor estruturado que tendo conhecimento das regras semânticas de *HaQuery* auxilia o utilizador a efectuar interrogações correctas.

A apresentação desta gramática de atributo está fora do contexto deste documento. Porém, para mostrar a semelhança entre o formalismo de gramática de atributos e (os tipos de dados em) *Haskell*, apresentamos de seguida o fragmento de Lrc que define a estrutura abstracta de um documento *Xml*:

```
Document    : CDocument (Prolog Miscs Element);
Element     : CElem    (Name LstAttribute LstContent);
```

```
Content     : CElement (Element)
            | CString  (CharData);
```

De notar as semelhanças deste fragmento da gramática de atributos e a sua definição em *Haskell* apresentado anteriormente.

3.1 Interrogando a Folha de Cálculo de Estudantes

Utilizando as técnicas anteriores, apresentamos um ambiente de programação que modela uma folha de cálculo de estudantes, totalmente especificada por uma gramática de atributos de ordem superior. O ambiente foi gerado pelo sistema LRC. Basicamente, a folha de cálculo consiste numa fórmula que é aplicada a um conjunto de estudantes que estão inscrito num curso.

Esta folha de cálculo apresenta-se na figura 1. O utilizar pode interactuar com o sistema através de selecção de menus, primir um botão, ou pura e simplesmente editando texto via o teclado. Por exemplo, o utilizador pode editar a fórmula na *frame* superior, ou pode ainda editar o registo de estudantes (*frame* inferior). Para cada uma desta interacções, o ambiente produz respostas em tempo real. Obviamente, que no contexto de folhas de cálculo um modelo de computação incremental é fundamental para se obter um bom tempo de resposta. Esta computação incremental é automaticamente obtida utilizando o sistema LRC, uma vez que este produz (também) calculadores de atributos incrementais. Geralmente, as folhas de cálculo são apresentadas em forma de tabela. Assim, neste ambiente embebeu-se uma componente de gramática de atributos existente no LRC para formatação de tabelas (mostrado na janela *Students ascii*).

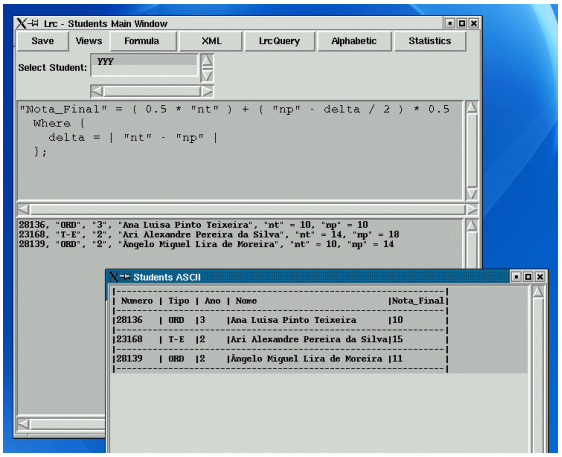


Figura 1. Um ambiente para modelar uma folha de cálculo de estudantes.

Observe que a coluna `Nota_Final` contém as notas que os alunos obtiveram em função da fórmula que está a ser considerada. Esta fórmula é (incremental-

mente) aplicada a cada um dos registos da base de dados de estudantes. Caso o utilizador altere a fórmula (ou a nota do aluno), então a folha de cálculo é automatica e incrementalmente actualizada.

O sistema Lrc permite disponibilizar diferentes vistas sobre a mesma entrada. Na figura 2 o utilizador pode também ver e interactuar com uma vista *Xml* (verbosa) da folha de cálculo[5].

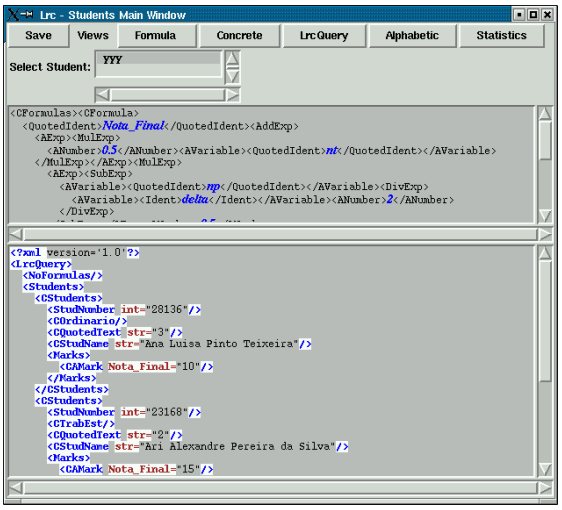


Figura 2. Uma vista *Xml* da folha de cálculo de estudantes.

Após aplicarmos a fórmula aos alunos, será conveniente interrogarmos a folha de cálculo para sabermos, por exemplo, quais os alunos que passaram ou reprovaram no curso. Para obter este comportamento podemos, obviamente, embeber a componente de gramática de atributos *LrcQuery* na nossa folha de cálculo. Na figura 3 apresenta-se uma possível frase em *LrcQuery* para a interrogação anterior.

Outras características do sistema *LrcQuery*:

- O sistema *LrcQuery* é um sistema incremental para interrogar linguagens/documentos. Isto é, o utilizador pode durante a execução do sistema alterar a formulação da interrogação, que o sistema imediata e incrementalmente produz a resposta.
- O ambiente de interrogação permite ao utilizar seleccionar diferentes vistas/notações para a linguagem de interrogação e compete ao utilizador seleccionar a que mais lhe agrada. Na figura 3 apresenta-se a vista por defeito

[5] A aplicação da fórmula à base de dados é feita em Lrc via atributos de ordem superior. Será interessante comparar como poderia ser modelado esta aplicação de uma função "dinâmica" utilizando os vários dialectos de *Xml*.

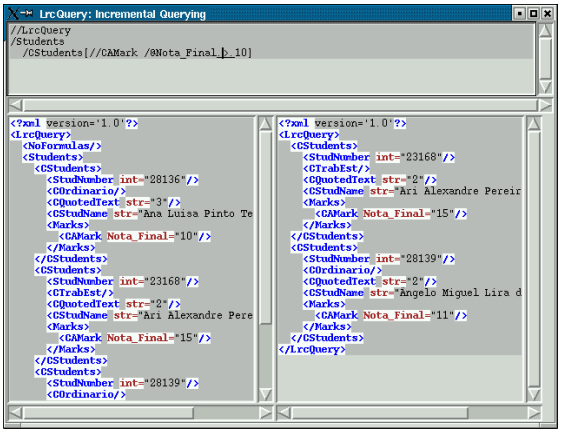


Figura 3. A Interrogação a folha de fálculo. Na *frame* superior encontra-se a interrogação (que o utilizador pode alterar). Na *frame* do lado esquero encontra-se a vista *Xml* da base de dados de alunos, enquanto na *frame* do lado direito encontra-se a resposta à interrogação.

que corresponde à notação concreta definida em *HaQuery*. Existe ainda uma vista em *Xml*, bem como uma representação gráfica da árvore *Xml* (incluída na biblioteca de combinadores *Xml* do LRC).

– Uma das características de sistema LRC é o facto de permitir *animar* a execução dos calculadores de atributos que produz (em modo incremental ou não). Deste modo, é possível animar a execução do sistema *LrcQuery* e visualizar graficamente a execução de uma interrogação.

4 Conclusões

Neste documento apresentamos a linguagem de interrogação *HaQuery*. O processador de *HaQuery* foi construído em dois paradígmas de programação diferentes: programação funcional (lazy) e gramáticas de atributos. Mais concretamente, a linguagem de domínio específico *HaQuery* foi embebida na linguagem *Haskell* e em gramáticas de atributos. No primeiro caso tirou-se partido dos combinadores de *Xml* do *Haskell* para embeber *HaQuery*. No segundo caso utilizaram-se técnicas avançadas de gramáticas de atributos, nomeadamente o uso de ordem superior para embeber a linguagem de domínio específico e, ainda, o uso de interfaces interactivo e incrementais para mais facil e eficientemente se fazerem as interrogações.

Por último gostariamos de referir que estes dois projectos foram propostos aos alunos como um projecto de uma disciplina de último ano do curso de licenciatura. Dois alunos, co-autores deste documento, aceitaram o desafio e realizaram sem grandes dificuldades o projecto. Estamos convencidos que com uma abordagem mais tradicional ao processamento de linguagens a biblioteca *HaQuery*

não teria sido construída dentro dos prazos estabelecidos, uma vez que exigiria um esforço muito maior na sua construção.

Referências

[dMBS00] Oege de Moor, Kevin Backhouse, and Doaitse Swierstra. First-Class Attribute Grammars. In D. Parigot and M. Mernik, editors, *Third Workshop on Attribute Grammars and their Applications, WAGA'99*, pages 1–20, Ponte de Lima, Portugal, July 2000. INRIA Rocquencourt.

[dMPJvW99] Oege de Moor, Simon Peyton-Jones, and Eric van Wyk. Aspect-Oriented Compilers. In *Proceedings of the First International Symposium on Generative and Component-Based Software Engineering (GCSE '99)*, volume 1799 of *LNCS*. Springer-Verlag, September 1999.

[Dra02] W3C Working Draft. *XQuery 1.0: An XML Query Language*, April 2002.

[FCS] António Faria, David Costa, and João Saraiva. LrcQuery: a Multiple View, Interactive and Incremental Querying System for XML. Technical report, Department of Computer Science, University of Minho , Portugal. (in preparation).

[KS98] Matthijs Kuiper and João Saraiva. Lrc - A Generator for Incremental Language-Oriented Tools. In Kay Koskimies, editor, *7th International Conference on Compiler Construction, CC/ETAPS'98*, volume 1383 of *LNCS*, pages 298–301. Springer-Verlag, April 1998.

[KS02] Uwe Kastens and Carsten Schmidt. Vl-eli: A generator for visual languages. In Mark van den Brand and Ralf Laemmel, editors, *Electronic Notes in Theoretical Computer Science*, volume 65. Elsevier Science Publishers, 2002.

[MLAŽ02] Marjan Mernik, M. Lenič, E. Avdičaušević, and V. Žumer. Lisa: An interactive environment for programming language development. In Nigel Horspool, editor, *International Conference on Compiler Construction, CC/ETAPS'02*, volume 2304 of *LNCS*, pages 1–4. Springer-Verlag, April 2002.

[RT89] T. Reps and T. Teitelbaum. *The Synthesizer Generator*. Springer, 1989.

[Sar99] João Saraiva. *Purely Functional Implementation of Attribute Grammars*. PhD thesis, Department of Computer Science, Utrecht University, The Netherlands, December 1999. `ftp://ftp.cs.uu.nl/pub/RUU/CS/phdtheses/Saraiva/`.

[Sar02a] João Saraiva. Component-based Programming for Higher-Order Attribute Grammars. In Don Batory, Charles Consel, and Walid Taha, editors, *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, GCSE 2002, Held as Part of the Confederation of Conferences on Principles, Logics, and Implementations of High-Level Programming Languages, PLI 2002, Pittsburgh, PA, USA, October 3-8, 2002*, volume 2487 of *LNCS*, pages 268–282. Springer-Verlag, October 2002.

[Sar02b] João Saraiva. *Language Processing (with a functional flavour)*. 2002. (in preparation).

[SAS99] Doaitse Swierstra, Pablo Azero, and João Saraiva. Designing and Implementing Combinator Languages. In Doaitse Swierstra, Pedro Henriques,

and José Oliveira, editors, *Third Summer School on Advanced Functional Programming*, volume 1608 of *LNCS*, pages 150–206. Springer-Verlag, September 1999.

[Wad02] Philip Wadler. Xquery: a typed functional language for querying xml. In *Fourth Summer School on Advanced Functional Programming, Oxford*, August 2002.

[WR99] Malcolm Wallace and Colin Runciman. Haskell and xml: Generic combinators or type-based translation? In *International Conference on Functional Programming*, pages 1–12. ACM, 1999.

A Exemplo: Documento *Xml*

```
<curso tipo='licenciatura' anos='5'>
  <nome>Matemática e Ciências da Computação</nome>

  <cadeira ano='1' tipo='semestral' semestre='1' creditos='5.0'>
    <designacao abrev='AL'>Álgebra Linear</designacao>

    <docentes dpt='dm'>
      <docente aula='T'>
        <nome sobrenome='Teixeira'>Lurdes</nome>
        <email>lurdes@math.di.uminho.pt</email>
      </docente>
      <docente aula='TP'>
        <nome sobrenome='Teixeira'>Lurdes</nome>
        <email>lurdes@math.di.uminho.pt</email>
      </docente>
    </docentes>

    <inscritos n='3'>
      <aluno n='55555' ano='1' matriculas='3' estatuto='TE'>
        <nome sobrenome='Atento'>Joaozinho Sempre</nome>
      </aluno>
      <aluno n='55556' ano='1' matriculas='1' estatuto='ORD'>
        <nome sobrenome='Teixeira'>Mafalda</nome>
      </aluno>
      <aluno n='40000' ano='3' matriculas='5' estatuto='ORD'>
        <nome sobrenome='Grossa'>Nelita Coxa</nome>
      </aluno>
    </inscritos>
    <avaliacao aprovados='1' reprovados='2'/>
  </cadeira>

  <cadeira ano='1' tipo='semestral' semestre='1' creditos='3.5'>
    <designacao abrev='PPI'>Paradigmas da Programacao I</designacao>

    <docentes dpt='di'>
      <docente aula='T'>
        <nome sobrenome='Valença'>José Manuel</nome>
```

```
        <email>jmv@di.uminho.pt</email>
        <url>www.di.uminho,pt/~jmv</url>
      </docente>
      <docente aula='TP'>
        <nome sobrenome='Barros'>José Bernardo</nome>
        <email>jbb@di.uminho.pt</email>
        <url>www.di.uminho.pt/~jbb</url>
      </docente>
    </docentes>

    <inscritos n='4'>
      <aluno n='55555' ano='1' matriculas='3' estatuto='TE'>
        <nome sobrenome='Atento'>Joaozinho Sempre</nome>
      </aluno>
      <aluno n='55556' ano='1' matriculas='1' estatuto='ORD'>
        <nome sobrenome='Teixeira'>Mafalda</nome>
      </aluno>
      <aluno n='44000' ano='3' matriculas='4' estatuto='ORD'>
        <nome sobrenome='Muito'>Mane andaquia</nome>
      </aluno>
      <aluno n='55556' ano='2' matriculas='2' estatuto='ORD'>
        <nome sobrenome='Atento'>Zezinho</nome>
      </aluno>
    </inscritos>
  </cadeira>
</curso>
```

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

14 Fev.	S7(14.30h) Publicação Electrónica de Conteúdos	[*FrameDoCMS _ um sistema de gestão de conteúdos para documentação de frameworks baseado em XML e WikiWikiWeb*] - **Ademar Aguiar**, FEUP - UP; **Gabriel David**, FEUP - UP; [*XML na Modelação de Sistemas Hipermédia*] - **Luís Carriço**, DI-FC-UL; **Rui Lopes**, DI-FC-UL; **Miguel Rodrigues**, DI-FC-UL; **Amadeu Dias**, DI-FC-UL; [*Geração de Conteúdos para Plataformas Móveis*] - **Filipe Figueiredo Correia**, ParadigmaXis; **Joel Varanda**, ParadigmaXis; **João Correia Lopes**, FEUP-UP; **Alexandre Valente de Sousa**, ISMAI;

FRAMEDOCMS – um sistema de gestão de conteúdos para documentação de frameworks baseado em XML e WikiWikiWeb

Ademar Aguiar, Gabriel David

Faculdade de Engenharia da Universidade do Porto / (INESC Porto?)
Rua Dr. Roberto Frias nº 378, 4200-465 Porto, Portugal
{aaguiar, gtd}@fe.up.pt

Resumo. A qualidade da documentação das frameworks aplicacionais orientadas por objectos constitui um factor crucial para a sua reutilização eficaz. Mas produzir boa documentação para uma framework é difícil, desagradável e dispendioso, sobretudo devido à diversidade de informação que é necessário escrever, consolidar e interligar. Este artigo apresenta um sistema de gestão de conteúdos desenvolvido com recurso a tecnologia XML e WikiWikiWeb, com o objectivo de simplificar o processo de documentação de frameworks, contribuindo assim para a redução dos seus custos de produção.

1 Introdução

As frameworks aplicacionais orientadas por objectos constituem uma poderosa técnica para reutilização de software em larga-escala cujo sucesso depende fortemente da qualidade da documentação que as acompanha [9][5].

Uma *framework* (infraestrutura aplicacional) pode ser definida como uma aplicação reutilizável, semi-acabada, fácil de estender, que foi especialmente concebida para servir de base à produção de aplicações concretas para um determinado domínio de problemas. Através da reutilização simultânea de código e desenho, as frameworks permitem aumentar a produtividade de desenvolvimento, reduzir tempos e melhorar a qualidade das aplicações **Error! Reference source not found.**[10][11].

Tal como outras vantagens da reutilização de software em geral, os benefícios da reutilização de frameworks manifestam-se apenas ao longo do tempo e requerem investimentos iniciais. Por exemplo, até que uma equipa de desenvolvimento fique apta a reutilizar uma framework de forma efectiva, é necessário dedicar tempo e esforço considerável na aprendizagem e compreensão dos seus detalhes. As frameworks constituem um dos tipos de produtos de software mais complexos, e como tal são também normalmente dificeis de aprender a usar e de compreender de ínicio.

Contudo, quando uma framework é acompanhada de uma boa documentação ela torna-se mais fácil de aprender e utilizar. Mas por outro lado, definir e produzir boa documentação para uma framework é normalmente bastante complexo e dispendioso.

Neste artigo apresenta-se o FrameDocMS (Framework Documentation Content Management System), um sistema de gestão de conteúdos em desenvolvimento especificamente concebido para apoiar e automatizar, na medida do possível, tarefas típicas de documentação de frameworks, que vão desde a criação, organização e transformação de conteúdos, até à sua apresentação e utilização final em diferentes formatos. Este trabalho é parte integrante de uma nova abordagem para a produção e utilização de documentação de frameworks [1], a qual tem como objectivo fundamental facilitar a reutilização de frameworks aplicacionais orientadas por objectos. Adicionalmente às ferramentas e utilitários referidos neste artigo, a abordagem define ainda um modelo de conteúdos, um processo genérico para documentação de frameworks e um visualizador de conteúdos configurado de acordo com a teoria de instrução minimalista [7].

2 Documentação de Frameworks

Documentação de qualidade favorece qualquer projecto de software, mas para as frameworks uma boa documentação é um factor crucial de sucesso. Sem uma documentação técnica adequada que seja clara, completa e precisa, capaz de explicitar os detalhes mais importantes de arquitectura e desenho, e orientar os utilizadores na sua configuração, as frameworks podem revelar-se particularmente difíceis de compreender por novos utlizadores, requerendo longos períodos de aprendizagem e comprometendo assim algumas das suas principais vantagens de reutilização [4][6].

Uma boa documentação efectivamente facilita a aprendizagem e compreensão de uma framework, mas a sua produção e manutenção é normalmente uma tarefa dificil, dispendiosa e desagradável para a equipa de desenvolvimento. Dependendo do aspecto concreto a documentar, do tipo de leitor em causa e das preferências do produtor da documentação, pode ser conveniente recorrer a diferentes tipos de documentação.

2.1 Diferentes Tipos de Documentos

A documentação típica de uma framework inclui informação que a permite descrever sob diferentes perspectivas (vistas externa, interna, estática e dinâmica), a diferentes níveis de abstracção e detalhe (arquitectura, desenho, código), e para tal agrega normalmente diversos tipos de documentos, os quais se descrevem a seguir.

Visão Global. A visão global permite definir o contexto da framework relativamente ao domínio coberto, constituindo assim um primeiro passo decisivo para a sua eventual selecção e reutilização. Neste texto introdutório, tenta-se apresentar o vocabulário fundamental do domínio do problema e explicita-se de forma muito clara

o âmbito da framework, as funcionalidades que são suportadas e as que não são, as que são configuráveis e as que são rígidas. Pode ainda ser apresentada uma aplicação concreta muito simples, bem como enumerada a restante documentação fornecida.

Exemplos. Os exemplos de aplicações construídas com base na framework são frequentemente o primeiro, e muitas vezes o único, tipo de documentação disponibilizado aos utilizadores da framework. Por forma a facilitar a sua compreensão imediata por parte de novos utilizadores, a documentação deve incluir um conjunto de exemplos em código-fonte que gradualmente vai apresentando as formas possíveis de reutilização da framework, desde as mais simples até às mais sofisticadas, terminando com exemplos de aplicações completas.

Guiões. Os guiões de configuração (*recipes* e *cookbooks*) descrevem informalmente como reutilizar uma determinada funcionalidade da framework, usando para tal linguagem natural, figuras e, por vezes também, código-fonte. Embora informalmente, cada guião segue uma estrutura que inclui objectivos, sequência de passos a realizar, eventuais referências para outros guiões ou documentos e código-fonte de exemplos. Os diversos guiões devem ser organizados em forma de espiral, em que os guiões que descrevem as formas mais frequentes de reutilização são apresentados primeiro, e os que contém detalhes e conceitos são deixados para o fim. Diversas frameworks recorreram com sucesso a este tipo documentação, das quais se destacam a MVC (Model-View-Controller), a MacApp, e a CommonPoint da Taligent [14]**Error! Reference source not found.**[12].

Casos de Utilização. Os casos de utilização permitem documentar as funcionalidades suportadas pela framework, constituindo assim um excelente recurso para se identificar com detalhe o domínio aplicacional coberto, e as funcionalidades que requerem e permitem adaptação.

Padrões de Desenho. Os padrões de desenho descrevem uma solução genérica para um problema de desenho que é recorrente num determinado contexto [11][3][2]. Os padrões constituem um nível de abstracção intermédio entre o nível de aplicação e o nível de classes, ao especificar as relações entre classes e objectos envolvidos num determinado problema de desenho. A descrição de um padrão normalmente explica o problema, o contexto, a solução, e uma discussão das consequências da sua aplicação, utilizando uma forma textual, eventualmente ilustrada com figuras, modelos e código-fonte de exemplos concretos.

Manual de Desenho. O manual de desenho, por vezes também referido como manual de produto, é o tipo de documento que colecciona toda a informação relacionada com o desenvolvimento da framework, incluindo informação sobre o domínio do problema, requisitos, especificações, arquitectura, componentes, desenho, código, e evolução histórica. De crucial importância é a inclusão da análise dos compromissos de desenho encontrados durante o desenvolvimento e as respectivas justificações para as decisões tomadas.

Manual de Referência. O manual de referência duma framework orientada por objectos, contém a descrição de todos os módulos, classes, interfaces, métodos, variáveis e tipos, referindo o papel e responsabilidades assumidas no sistema e realçando os pontos e formas de configuração suportadas. Os conteúdos dos manuais de referência são na sua grande maioria deriváveis directamente do código-fonte e respectiva documentação de baixo-nível. Embora úteis a utilizadores experientes como documento de referência, não são muito úteis na fase de aprendizagem.

2.2 Diferentes Notações para os Conteúdos

Da análise dos diversos tipos de documentos utilizados na documentação de frameworks pode-se constatar que os seus contéudos são originalmente representados em diferentes notações:

- **Código-fonte**, correspondendo ao código escrito para a implementação da framework e exemplos numa determinada linguagem de programação.
- **Modelos**, correspondendo a informação estruturada, textual ou visual, que permite a descrição formal (ou quase) da framework ou partes dela. Em frameworks orientadas por objectos, os modelos são normalmente descritos na notação UML[1].
- **Textos estruturados**, correspondendo a conteúdos textuais que podem e devem ser organizados de acordo com uma estrutura bem definida, embora possivelmente flexível, tais como os guiões e os padrões de desenho.
- **Texto livre**, correspondendo a conteúdos textuais sem estrutura bem definida, ou sem grandes necessidades de estruturação.
- **Outros formatos**, correspondendo a imagens, figuras, blocos de texto, etc.

Esta classificação das notações utilizadas não pretende ser exaustiva, mas apenas servir para realçar as diferenças em termos de representação dos diversos conteúdos.

2.3 Diferentes Audiências

Adicionalmente à diversidade de tipos de documentos e de notações utilizadas para representar os conteúdos, a documentação de frameworks deve ainda satisfazer as necessidades de diferentes audiências.

As frameworks podem ser reutilizadas por diferentes tipos de utilizadores com diferentes necessidades em termos da informação que procuram na documentação de uma framework.

Programadores de Aplicações. Os programadores de aplicações numa primeira fase de selecção de frameworks procuram informação sobre o domínio do problema coberto e os pontos de configuração, para poder avaliar se determinada framework é

[1] UML — Unified Modeling Language, uma norma da Object Management Group para a representação de modelos de objectos.

ou não útil para o desenvolvimento das aplicações que têm em vista. Após a fase de selecção, os programadores pretendem saber como estender e configurar a framework para produzir a aplicação em vista. Eles precisam de identificar os pontos de configuração relevantes e aprender a configurá-los de acordo com os requisitos das suas aplicações concretas.

Programadores da Framework. Os programadores da própria framework precisam conhecer desde o domínio de aplicação, e desenho interno da framework, até aos mais pequenos detalhes de implementação de algumas das suas partes.

Programadores de Outras Frameworks. Os programadores de outras frameworks normalmente estudam diversas frameworks, mesmo de outros domínios, com o objectivo de encontrar ideias e padrões de desenho que lhes possam ser úteis para o desenho das frameworks em vista.

Destes tipos de utilizadores, destacam-se os programadores das aplicações baseadas na framework, por constituírem a maior audiência, e os programadores da própria framework, por terem um papel muito importante tanto na autoria como na utilização intensiva da documentação.

2.4 Fluxo de Conteúdos Típico

Por forma a satisfazer toda esta diversidade de requisitos, têm assim de ser produzidos, integrados e consolidados num repositório comum um volume considerável de conteúdos. Por forma a permitir uma boa navegabilidade, os conteúdos normalmente contêm inúmeras referências cruzadas, umas definidas implicitamente pelo contexto e outras definidas explicitamente pelos autores.

O fluxo típico de produção de documentação para uma framework começa, naturalmente, com a criação de diversos tipos de conteúdos, entre os quais código, modelos, documentos, imagens, etc. Idealmente, a criação de todos estes tipos de conteúdos pode decorrer em paralelo, pelo que a consistência entre eles deve ser sempre assegurada para que seja fácil disponibilizar a todos os autores e leitores as suas versões mais actuais. Um aspecto importante desta fase consiste em suportar a criação e actualização de referências cruzadas entre conteúdos. Durante a criação é ainda comum utilizar-se documentos-tipo e seguir normas de codificação e documentação. Na figura 1. apresenta-se de forma simplificada este fluxo de conteúdos.

Depois de criados, os diferentes tipos de conteúdos são normalizados, integrados e armazenados. Por último, são formatados e seleccionados de acordo com as necessidades específicas dos seus leitores, para serem posteriormente publicados e apresentados.

Diversas abordagens para documentação de frameworks provaram ser eficazes na redução da típica longa curva de aprendizagem, mas a melhor combinação de tipos de documentos, técnicas de escrita e estilos de apresentação não é única, e depende muito do contexto, de condicionantes económicas, especificidades que variam de

projecto para projecto, e até das próprias características psicológicas e condicionalismos específicos de cada programador/utilizador.

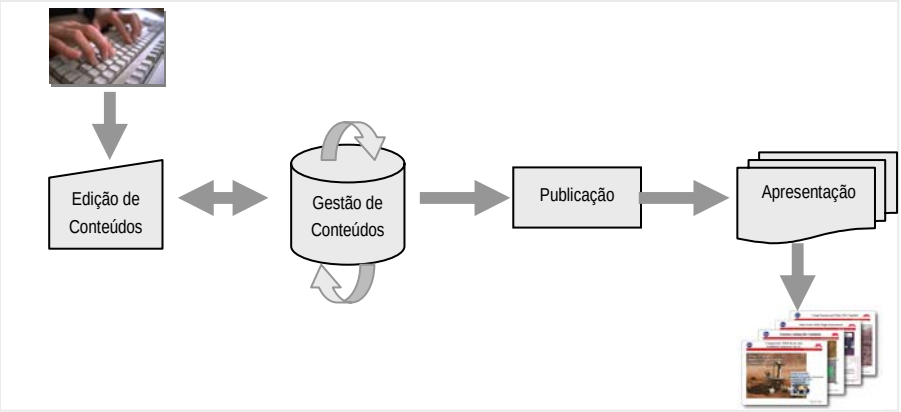


Fig. 1. Fluxo de contéudos típico.

Assim, uma boa abordagem para documentação frameworks tem de conciliar diversos requisitos, entre os quais se destacam os seguintes:

- ser fácil de usar pelos programadores, para que a actividade de documentação não constitua um obstáculo à actividade de programação, mas antes pelo contrário a apoie;
- ser flexível para se adaptar às particularidades de cada ambiente de projecto;
- ser capaz de interligar de forma simples os diferentes tipos de conteúdos produzidos;
- ser económico para possibilitar reduzir os altos custos normalmente associados à produção de documentação.

Por tudo isto, é crucial uma gestão integrada de conteúdos, facilmente adaptável e integrável no ambiente de desenvolvimento, e que seja capaz de suportar desde as fases de criação e integração, até às fases de publicação e apresentação de conteúdos orientada às diferentes audiências.

3 WikiWikiWeb

Um WikiWikiWeb, ou simplesmente um Wiki, pode ser definido como uma plataforma Web que suporta a edição cooperativa de documentos. Um Wiki é um sítio Web escrito pelos seus utilizadores. Qualquer utilizador pode modificar e criar páginas usando um vulgar navegador Web.

O conceito Wiki é uma invenção de Ward Cunningham e criou o primeiro Wiki em [17]. A palavra Wiki-Wiki (uí-ki-uí-ki) significa rápido em hawaiano. Actualmente,

existem inúmeras implementações diferentes deste conceito, desde umas muito simples até outras de gestão e instalação mais complexa mas que adicionam novas funcionalidades úteis, tais como: armazenamento em bases de dados, controlo de acessos, notificações por email e gestão de utilizadores.

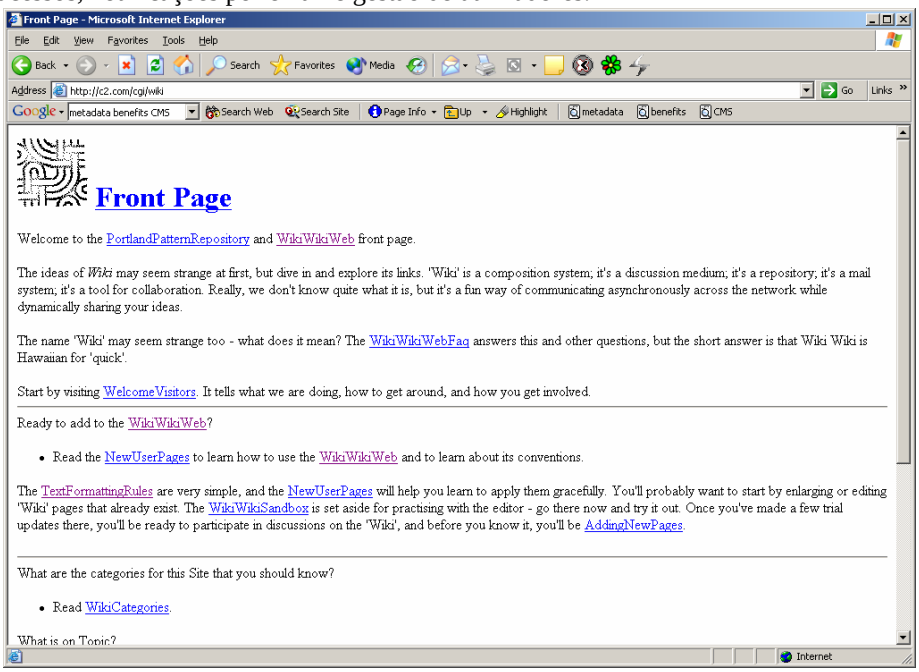


Fig. 2. A página de entrada no primeiro WikiWikiWeb em http://c2.com/.

Num Wiki, cada página corresponde a um tópico diferente. As páginas podem ser prontamente editadas através dos botões "Edit" vulgarmente disponíveis no topo ou no fundo da página. Depois de editar o tópico através do navegador Web, as alterações podem ser gravadas, ficando imediatamente disponíveis na página desse tópico.

O Wiki tem uma linguagem de anotação própria muito simples, que para além de formatações de texto elementares, permite também a ligação automática entre tópicos através de palavras com formatos convencionados, dos quais os nomes Wiki são o exemplo mais importante. Um nome Wiki é uma concatenação de duas ou mais palavras iniciadas por letra maiúscula, tais como os nomes: *WikiName*, *TopicoWiki*, ou *AindaNomesWikiGrandesComoEste*. Diversas implementações do conceito Wiki permitem estender facilmente a linguagem de anotação para suportar, por exemplo, novas convenções para a definição automática de ligações.

Tópicos novos são automaticamente criados sempre que, através de um nome Wiki, se acede a um tópico que ainda não tem página criada. A eliminação de tópicos é suportada através da escrita de palavras reservadas no fim da página, por exemplo.

Sendo a edição completamente livre, pode-se pensar que o risco dos conteúdos se perderem é grande, bastando para tal a utilização indevida de um dos seus utilizadores. E de facto assim é, caso o utilizador Wiki não siga as regras de "boa conduta Wiki". De qualquer modo, para obviar o problema, a grande maioria das

implementações Wiki possui um sistema de controlo de versões que permite registar todas as alterações efectuadas, e assim repôr versões anteriores em caso de necessidade.

Em suma, um Wiki é uma ferramenta de colaboração bastante inovadora porque, através de uma interface e regras muito simples, permite a edição e organização dos conteúdos, por qualquer utilizador, técnico ou não, o que promove não só uma utilização participada e crítica mas também uma evolução gradual dos conteúdos. Outras características típicas de um Wiki são: pesquisa em texto integral, formatação de texto, possibilidade de anexar ficheiros a páginas.

4 Componentes do FrameDocMS

O FrameDocMS é uma infraestrutura baseada em XML e WikiWikiWeb destinada a apoiar nas tarefas de criação, integração, transformação e apresentação dos diversos tipos de conteúdos tipicamente utilizados na documentação de frameworks aplicacionais orientadas por objectos.

O sistema é composto por um Wiki e um conjunto de diversos documentos-tipo, linguagens de anotação, conversores de conteúdos para representações em XML, e transformadores de conteúdos XML para formatos de apresentação (HTML[2] e PDF[3]).

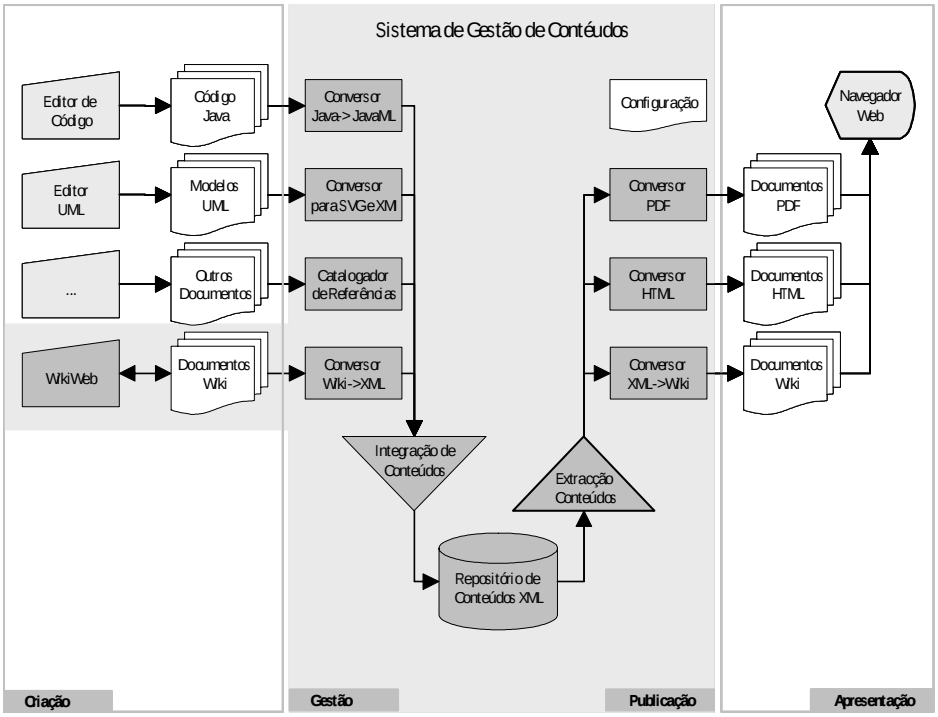


Fig. 3. Componentes do FrameDocMS e sua interligação ao longo do fluxo de conteúdos.

Na figura 2. representam-se estes componentes bem como a sua interligação ao longo do fluxo de conteúdos, desde a fase de criação até à fase de apresentação.

4.1 Wiki Adaptado.

O componente principal do sistema é um Wiki adaptado para satisfazer os requisitos da produção de documentação para frameworks. Para cada tipo de documento a suportar (tutoriais, exemplos, guiões, padrões de desenho, etc), o Wiki é dotado de um documento-tipo e de capacidades para: mapeamento desse tipo de documento para XML, e reconhecimento de novos tipos de ligações automáticas para esse tipo de documentos. Adicionalmente, ao Wiki são ainda acrescentados diversos tópicos reservados através dos quais é possível efectuar configurações, consultas e processamentos úteis para a documentação de frameworks.

Com base num Wiki assim configurado é possível criar e organizar os conteúdos dos vários tipos de documentos utilizados na documentação de uma framework, cooperativamente por toda a equipa de desenvolvimento, com tempos de publicação instântaneos, e recorrendo simplesmente a um vulgar navegador Web.

Anotação de Documentos.

Para facilitar a edição de conteúdos de documentos de estrutura menos simples, é possível disponibilizar documentos-tipo estruturados, isto é, documentos que devem obedecer a uma estrutura pré-definida, normalmente definida em esquemas XML. Por exemplo, a edição de um documento do tipo padrão de desenho, pode ter associada uma hipotética linguagem de anotação PATTERNML (*Pattern Markup Language*), ou DOCBOOK, as quais ditam a forma de descrição dos seus conteúdos, normalizando-os.

Para além da utilidade para a criação e validação de conteúdos, a possibilidade de edição de documentos estruturados é também muito importante para uma boa gestão de conteúdos, ao permitir a anotação (manual ou automática) dos conteúdos com meta-informação (descrições, autoria, datas, palavras-chave, etc.), um valioso recurso para a pesquisa, classificação, navegação e formatação de conteúdos.

Ligações Automáticas para Documentos, Código e Modelos.

Uma página Wiki pode referenciar outros tópicos Wiki, através de ligações internas, ou referenciar outras páginas Web quaisquer, através de ligações externas.

As ligações internas podem ser definidas implicitamente no texto, através da utilização de esquemas de nomes convencionados, tal como descritos nas regras de formatação de texto de um Wiki. As regras podem variar com a implementação de um Wiki, mas apenas no sentido de as estender e não de as alterar. As ligações externas

---

[2] HTML ─ Hypertext Markup Language

[3] PDF ─ Adobe Portable Document Format

para outros documentos devem ser explicitamente definidas através dos respectivos endereços (URL). Por exemplo:

- o texto *StartingPoints* permite definir uma ligação interna para um tópico com o nome *StartingPoints*;
- o texto *c2:StartingPoints* permite definir uma ligação para o tópico *StartingPoints* do Wiki em http://c2.com/cgi/wiki/.
- o texto http://c2.com/cgi/wiki/ define uma ligação para o URL que identifica.

Pela sua importância especial no contexto da documentação de frameworks e software em geral, as ligações entre páginas Wiki e código-fonte ou modelos, são suportadas de forma especial através de extensões às regras base de um Wiki. Dependendo do Wiki concreto, as extensões podem ser definidas de várias formas.

Por exemplo:

- o texto *javaLink:HelloWorld* poderia ser interpretado como uma ligação para a classe HelloWorld localizada no primeiro dos caminhos definidos para a pesquisa de ficheiros de código-fonte em linguagem Java;
- o texto *javaInline:hello.HelloWorld* permitiria incluir nesta página o código-fonte da classe *HelloWorld* pertencente ao pacote *hello*;
- o texto *umlLink:classDiagram:HelloWorld* poderia ser interpretado como uma ligação para a figura com o diagrama de classes de nome *HelloWorld* localizado de acordo com os caminhos definidos para a pesquisa de diagramas UML.

De forma idêntica, podem também ser definidas convenções para o estabelecimento de ligações entre documentos de vários tipos, mas nestes casos novas regras são dispensáveis, bastando simplesmente utilizar nomes Wiki.

4.2 Conversores para XML

Em consequência da utilização intensiva do Wiki, a grande maioria dos textos dos documentos tende a residir em páginas Wiki. Os restantes conteúdos importantes são o código-fonte, actualmente suportado apenas em linguagem Java, e modelos UML, tanto a sua componente vectorial como a sua componente estrutural.

Para cada um destes tipos de conteúdos são utilizados transformadores baseados em folhas de estilo XSL (*Extensible StyleSheet Language*) que os convertem para uma representação equivalente em XML, respectivamente para as linguagens WikiML (*Wiki Markup Language*), JavaML (*Java Markup Language*), SVG (*Scalable Vector Graphics*) e XMI (*XML Metadata Interchange*). Dependendo do implementação concreta do Wiki utilizado, a conversão de páginas Wiki para XML pode ainda requerer a escrita de um analisador sintático específico e só então é que a folha de estilos pode ser aplicada.

4.3 Repositório de Conteúdos

Atendendo a que os conteúdos serão na sua grande maioria representáveis em XML, o repositório pode corresponder a uma área no sistema de ficheiros, onde para além dos

ficheiros XML são também guardadas referências ou cópias dos restantes documentos não representados em XML. Embora inicialmente mais dificil de implementar, e mais lento em termos computacionais, o repositório pode até dispensar o armazenamento de todos os ficheiros XML, uma vez que os pode obter dinamicamente através dos conteúdos nos formatos originais e programas específicamente desenvolvidos para os processar. Dependendo do Wiki utilizado, o repositório pode também assumir a forma de uma base de dados relacional.

4.4 Conversores para Apresentação

A apresentação dos conteúdos é suportada em dois formatos principais: o formato HTML, para visualização através de um navegador Web; e o formato PDF, para impressão em papel.

Para a formatação dos conteúdos em cada um destes formatos serão utilizados dois grupos de conversores repectivos, utilizando folhas de estilo XSL. Atendendo a que a transformação dos conteúdos para o formato de apresentação na Web é feita pelo próprio Wiki, para os conteúdos em páginas Wiki, apenas é necessário transformar os conteúdos armazenados em código-fonte e em modelos UML.

Quando se pretende adequar os conteúdos a um tipo de audiência específico, pode-se revelar necessário utilizar uma organização dos conteúdos distinta da original, o que pode obrigar a reordenações, filtragens e composições visuais dos conteúdos diferentes das originais.

Relativamente ao formato para impressão, as transformações necessárias são mais consideráveis, uma vez que todos os conteúdos têm que ser sequenciados em fluxos de texto, ser devidamente interligados através de referências cruzadas, e exaustivamente indexados por forma a facilitar simultaneamente a sua leitura sequencial, em hipertexto, e as suas consultas por referência. A indexação será efectuada com base na meta-informação entretanto recolhida durante a criação dos conteúdos, a sua qualidade dependerá da quantidade e qualidade dessa meta-informação.

5 Tecnologias de Integração: WikiWikiWeb e XML

Para a implementação do sistema de gestão de conteúdos é necessária a integração dos seus vários componentes, quer ao nível da interoperabilidade de funcionamento dos vários programas e aplicações envolvidas, quer ao nível da informação trocada entre eles.

5.1 Integração Funcional

A integração dos vários componentes ao nível funcional é realizada principalmente pelo Wiki e pelas extensões desenvolvidas para a sua adaptação. Atendendo a que o

Wiki utilizado se encontra implementado em linguagem Java e utiliza *Servlets* e JSP[4], a integração é conseguida através de Wiki, Java e JSP.

5.2 Integração de Informação

Em termos de informação, a integração entre os vários componentes é conseguida através do recurso a XML, e a uma linguagem de anotação intermédia, a FRAMEDOCML, que se destina a integrar numa única linguagem de anotação os diversos tipos de conteúdos representáveis em XML.

Sempre que necessário para efeitos de processamento, os conteúdos originais são transformados para conteúdos nesta linguagem intermédia, e posteriormente para formatos apropriados para apresentação.

Outros conteúdos sem necessidade de representação em XML poderão residir em ficheiros, tais como ficheiros PDF, imagens, ou outros. Nestes casos, a navegação suportada para localizações internas aos seus conteúdos, fica dependente de como tal navegação é suportada pelo navegador Web em utilização, não sendo implementado nenhum processamento adicional.

A integração ao nível da informação veiculada entre os componentes é assim conseguida através de dados XML, os quais são processados por utilitários e programas Java, utilizando também folhas de estilo em XSL e analisadores sintáticos do tipo SAX[5].

5.3 Integração em Ambientes de Desenvolvimento Integrado

A integração do sistema de gestão de conteúdos com outras ferramentas de desenvolvimento e aplicações externas ao sistema FrameDocMS, é realizada através da informação produzida pelos editores de conteúdos (código-fonte, modelos, documentos, etc).

No entanto, a integração do sistema com ferramentas de desenvolvimento integrados, tais como o JBuilder ou o VisualAge for Java permite ir mais além, ao possibilitar a integração dos editores de código e de documentação Wiki. Melhor ainda, a integração do Wiki numa ferramenta de desenvolvimento como o Together, Eclipse e Rational XDE, que permitem a edição simultânea de código e modelos UML, permite a integração completa de todos os editores necessários para produzir os conteúdos típicos de documentação de frameworks. Como os Wiki apenas requerem um navegador Web, a sua integração num ambiente de desenvolvimento deste género é regra geral extremamente fácil de realizar, o que possibila a edição sincronizada de todos os tipos de conteúdos numa mesma ferramenta, com as consequentes vantagens em termos de aumento de produtividade, tanto na actividade de desenho, como na programação e documentação de uma framework.

---

[4] JSP — *Java Server Pages*

[5] SAX — *Simple API for XML*

Prevê-se a experimentação deste tipo de integração numa fase posterior do presente trabalho.

6 Conclusões

Uma boa documentação é crucial para o sucesso de uma framework mas a sua produção é também difícil e dispendiosa. Uma das dificuldades reside na integração dos diversos tipos de documentos utilizados, cujos conteúdos se encontram intimamente relacionados, embora representados em diferentes notações (texto, modelos, código-fonte, imagens, etc). Uma outra dificuldade reside na necessidade de adequação dos diversos conteúdos disponíveis ao tipo do leitor, variando o âmbito e o nível de abstracção da informação apresentada.

Neste artigo, foi apresentado um sistema de gestão de conteúdos especificamente desenvolvido com o objectivo de melhorar o processo de documentação de frameworks. Os requisitos mais importantes para um sistema destes são a simplicidade, baixo custo, e a facilidade de utilização pelos diversos elementos da equipa de desenvolvimento, em especial, pelos programadores da framework, os maiores detentores de conhecimento importante àcerca da framework a documentar.

Entre as diversas alternativas possíveis para implementação de tal sistema foi escolhido o conceito de WikiWikiWeb, por se basear em ideias muito simples e atractivas, mas simultaneamente bastante inovador e poderoso em termos de edição cooperativa de documentos na Web.

O FrameDocMS é um sistema em fase de desenvolvimento que recorre à recorre à utilização conjunta das tecnologias XML e WikiWikiWeb. Conclui-se que esta combinação de tecnologias permite diminuir o esforço que é habitualmente necessário dispender para a documentação de frameworks orientadas por objectos, ao conseguir aliar a simplicidade, facilidade e versatilidade de edição de documentos num Wiki, às vantagens que a tecnologia XML oferece em termos de facilidades de integração, processamento e apresentação de informação.

Claro que um Wiki adaptado para editar documentos estruturados XML pode ser considerado mais difícil de usar do que um Wiki típico, ou que um Wiki pode ser considerado um editor de conteúdos XML menos bom do que um editor típico de XML. De qualquer modo, a combinação de ambas as tecnologias resulta numa ferramenta com características extremamente atractivas, capaz de afastar da mente de quem desenvolve o “fantasma” da documentação. Destas características realçam-se as seguintes:

- fácil de integrar num ambiente de desenvolvimento de software;
- fácil de usar por qualquer elemento da equipa de desenvolvimento, técnico ou não técnico: gestor, analista, arquitectos, programadores, verificadores, etc;
- promove a participação de todos os intervenientes no processo de documentação;
- melhora a comunicação entre os elementos da equipa de desenvolvimento;
- facilita o acesso, revisão e evolução gradual da documentação;

- permite a integração de conteúdos de forma controlada e estruturada, sem impôr grandes formalismos aos autores e preservando os conteúdos num formato universal como é o XML.

Em desenvolvimentos futuros, será avaliada a possibilidade de utilização de outras implementações de Wiki, com o objectivo de providenciar uma ferramenta mais completa do que o VeryQuickWiki em termos de facilidades de desenvolvimento em equipa.

Referências

[1] Aguiar, A. (2000). A minimalist approach to framework documentation. In Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum), pages 143–144. ACM Press.

[2] Alexander, C., Ishikawa, S., and Silverstein, M. (1977). A Pattern Language. Oxford University Press.

[3] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). Pattern Oriented Software Architecture — a System of Patterns. John Wiley & Sons.

[4] Butler, G. (1997). A reuse case perspective on documenting frameworks. http://www.cs.concordia.ca/faculty/gregb.

[5] Butler, G., Dénomné, P. Documenting frameworks. In Building Application Frameworks: Object-Oriented Foundations of Framework Design (New York, 1999), pp. 495-504. John Wiley and Sons.

[6] Butler, G., Keller, R. K., and Mili, H. (1998). A framework for framework documentation. http://www.cs.concordia.ca/faculty/gregb.

[7] Carroll, J. M. The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill, MIT Press, 1990.

[8] Cotter, S. and Potel, M. (1995). Inside Taligent Technology. Addison-Wesley.

[9] Fayad, M. and Schmidt, D. C. (1997a). Object-oriented application frameworks. Communications of the ACM, 40(10):32–38. …

[10] Fayad, M. E., Schmidt, D. C., and Johnson, R. E. (1999). Building Application Frameworks — Object-Oriented Foundations of Framework Design. John Wiley & Sons.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns — Elements of reusable object-oriented software. Addison-Wesley.

[12] Goldberg, A. (1984). Smalltalk-80: The Interactive Programming Environment. Addison-Wesley.

[13] Johnson, R. (1992). Documenting frameworks using patterns. In Paepcke, A., editor, OOPSLA'92 Conference Proceedings, pages 63–76. ACM Press.

[14] Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. Journal of Object-Oriented Programming, 1(3):27–49.

[15] Schmucker, K. J. (1986). Object-Oriented Programming for the Macintosh. Hayden Book Company.

[16] Taligent Inc. (1994). Building Object-Oriented Frameworks. Taligent Inc.

[17] The Original Wiki, http://c2.com/cgi/wiki.

XML na Modelação de Sistemas Hipermédia

Luís Carriço, Rui Lopes, Miguel Rodrigues e Amadeu Dias
Departamento de Informática,
Faculdade de Ciências da Universidade de Lisboa
Bloco C5, Piso 1, Campo Grande, Lisboa

Neste artigo apresenta-se uma metodologia de construção de aplicações hipermédia, com base nas aproximações sistemáticas de concepção destes sistemas. Destas aproximações que genericamente introduzem modelos a três níveis de abstracção (conceptual, de navegação e de apresentação), optou-se por recorrer às variantes orientadas a objectos, em particular à UWE. Esta propõe extensões à UML, sob a forma de estereótipos, que estabelecem interligações entre os modelos. O trabalho aqui descrito apresenta uma solução para a tradução integrada e congruente destes modelos para especificações XML, que, por sua vez, permitem com recurso ao XSLT e a CSS a geração de aplicações hipermédia. Estas aplicações assim geradas e fortemente baseadas nos modelos de concepção sustentam, não só uma clara separação entre estrutura de informação e apresentação, como também apresentam fortes características de coerência ao nível da interface com o utilizador. Finalmente é apresentado um caso de estudo em que a metodologia foi aplicada dando origem a um protótipo de um sistema hipermédia.

1. Introdução

A criação de sistemas hipermédia e, em particular, a sua concretização na Web tem, na sua grande maioria, seguido uma aproximação desregrada com consequências frequentemente desastrosas, em termos de coerência, quer a nível da estrutura, da navegação ou mesmo da apresentação, que se reflecte naturalmente na sua usabilidade.

A utilização de metodologias de concepção, com ênfase nos aspectos de modelação, tem, tal como noutras áreas, emergido como forma de minimizar o problema. Das várias propostas oferecidas, desde o HDM [11][10], ao OOHDM[22][23], passando pelo RMM [14][15] e outras[17], surgiu recentemente a UWE [3][12][18]. A UWE (UML-based Web Engineering approach) tenta integrar as ideias específicas dessas metodologias, orientadas para os sistemas hipermédia, nas capacidades da UML [4]. Esta linguagem, fortemente divulgada, é suficientemente genérica e extensível para que possa abarcar essa integração, cobrindo as diversas frentes do processo de modelação que devem ser suportadas na concepção dos sistemas hipermédia.

Genericamente pode dizer-se que as metodologias de concepção de sistemas hipermédia seguem de perto as aproximações sistemáticas e iterativas dos modelos de processos de desenvolvimento de software (e.g. Processo Unificado [16]) com particular ênfase na iteração, tendo em conta a riqueza, flexibilidade e requisitos de usabilidade inerentes ao conceito "hipermédia". Todavia, nas fases de análise e concepção ("design") alargam e organizam as aproximações genéricas, sistematizando-as na modelação: (1) da estrutura de informação subjacente (modelo conceptual); (2) da informação e estrutura de navegação (modelos de navegação); (3) e da apresentação (modelo de apresentação), esta última com variantes ao nível estrutural e dinâmico nas metodologias orientadas para objectos (e.g. UWE e OODHM). Também nestas é comum a utilização de Casos de Uso como meio de fundamentar os modelos, particularmente as perspectivas de navegação e, posteriormente, as de apresentação.

Ao nível da estrutura de conceitos, e usando como base a UWE [13] e a sua relação com a UML, as extensões fundamentais colocam-se nos modelos de navegação e apresentação. Nos primeiros introduzem-se abstracções (na forma de estereótipos) de contextos, índices, menus, visitas guiadas e interrogações [3] que são usados em diagramas de classes UML, focados nos aspectos de navegação. Nos segundos são

definidos elementos abstractos de interacção, usados igualmente em variantes (extensões) de diagramas de classes [12]. Os aspectos dinâmicos da apresentação traduzem-se por diagramas de estados e os modelos conceptuais são modelos de classes UML .

Uma das questões que se coloca nestas metodologias é, sem dúvida, a forma como se concretizam os sistemas hipermédia a partir dos modelos de alto nível, mantendo as noções semânticas da metodologia e a coerência que daí resulta na sua instanciação. Obviamente não se deve sobrecarregar o processo de codificação com repetições ou reformulações desnecessárias dos padrões definidos, quer ao nível dos modelos, quer ao nível do meta-modelo definido pela estrutura de conceitos e pela sua articulação no âmbito da metodologia.

Nesse sentido, o trabalho relatado neste artigo, propõe a utilização de XML [9] na construção de um suporte lógico para a concretização de sistemas hipermédia, concebidos de acordo com a metodologia UWE. Para esse fim, foram definidos os tipos de documentos XML necessários à materialização da estrutura de conceitos usada nos três conjuntos de modelos propostos (conceptual, de navegação e de apresentação). Definiram-se os elementos base dos diagramas de classe UML (e.g. classes e associações), bem como os elementos correspondentes aos estereótipos de navegação e apresentação definidos na UWE.

Com base nesses elementos fez-se uma especificação de um caso de estudo, a saber um sistema hipermédia sobre museus. Cada um dos modelos elaborados aquando da concepção do sistema foi então especificado em XML, desde o modelo conceptual ao de apresentação. Por fim a concretização da estrutura de apresentação (abstracta no modelo) foi traduzida em HTML recorrendo ao XSLT e CSS. Naturalmente esta especificação define os padrões de apresentação e interacção com o sistema (e.g. como é apresentado uma obra ou um artista, independentemente de qual é ou quem é) e não a sua instanciação para dados particulares (e.g. Mona Lisa, Guernica, Picasso). Finalmente as especificações, enquanto padrões de modelação e apresentação, foram reutilizados na instanciação dum protótipo do sistema propriamente dito.

O artigo está organizado do seguinte modo: na secção seguinte apresenta-se um panorama das aproximações existentes para a modelação de sistemas hipermédia, dando especial relevância as características da UWE enquanto metodologia adoptada neste trabalho; na secção 3 discutem-se alguns aspectos e aplicações da interligação da UML com a XML, elaborando um pouco mais sobre trabalhos e projectos em curso, que fazem essa ligação no sentido da geração de aplicações Web a partir de metodologias de concepção hipermédia e em particular da UWE; na secção seguinte descrevem-se as características das estruturas e do processo que nortearam o trabalho relatado neste artigo; na secção 5 apresenta-se o caso de estudo em que o processo foi aplicado discutindo-se sucintamente os modelos, especificações e protótipo desenvolvido; finalmente conclui-se e delineiam-se direcções de trabalho futuro.

2. Modelação de Sistemas Hipermédia

A aproximação sistemática à modelação de sistemas hipermédia (em que se inclui, mas que não se esgotam na Web) é uma disciplina recente. O seu ponto de visibilidade estabeleceu-se essencialmente com a publicação de um conjunto de artigos datados de 1995 em que metodologias como o HDM [10] o RMM [14] e o OOHDM [21] (na sua fase embrionária) foram apresentadas. Desde aí vêm surgindo evoluções [8][15] que, particularmente no caso do OOHDM [23] se têm aproximado das linguagens e metodologias de concepção de software genéricas, inclusive recorrendo à UML [4] nas fases iniciais do processo de desenvolvimento [22]. A UWE [13][3] propõe uma aproximação metodológica semelhante ao OOHDM, com a diferença de usar a UML ao longo de todo o processo de modulação. Nas suas evoluções mais recentes [18][12] introduz mesmo especificações mais rigorosas de transição entre modelos e, particularmente, de inter-ligação entre os conceitos (ou estereótipos) introduzidos em cada modelo (de facto meta-modelo).

Aspectos metodológicos

As metodologias acima apresentam-se num conjunto de fases que dão origem a conjunto de modelos implícita ou explicitamente interligados (Figura 1). No caso das duas últimas os Casos de Uso tomam um papel relevante nas fases iniciais do levantamento de requisitos e guiam, tal como propostos em [16], todas as fases do processo de desenvolvimento. O seu papel é estrutural mas a sua relevância em termos da geração do XML ou de um qualquer sistema hipermédia esbate-se nos restantes modelos.

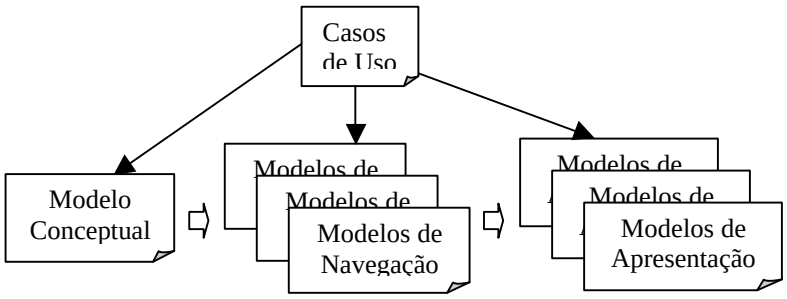


Figura 1. Modelos de concepção de um Sistema Hipermédia

Modelo conceptual

Quanto ao modelo conceptual este reflecte a estrutura da informação em termos de classes (ou entidades) e, no caso de o sistema se materializar numa base de dados, representará, grossomodo, o seu esquema conceptual. Também neste aspecto da modelação, como nos Casos de Uso, não são introduzidas extensões significativas às aproximações genéricas de modelação sejam elas a UML (OOHM e UWE) ou os esquemas Entidade Associação (RMM). São identificadas, para além das classes, os atributos e as relações passando pelas suas variantes (associações, generalizações, etc.)

Modelos de navegação

Relativamente aos modelos de navegação apresentam-se apenas os conceitos relativos à aproximação UWE, embora se possam encontrar conceitos equivalentes nos restantes. Assim sendo a UWE introduz duas fases de enriquecimento do modelo de navegação: (1) o modelo do espaço de navegação (2) o modelo de estrutura de navegação.

O primeiro introduz o conceito de classe de navegação, um estereótipo definido sobre classes genéricas UML. Cada uma dessas classes representa informação acessível numa determinada perspectiva de navegação, eventualmente associada a um Caso de Uso específico. De uma forma simplificada e traçando um paralelo com as Bases de Dados pode dizer-se que as classes de navegação representam "vistas" das classes conceptuais. No modelo do espaço de navegação, por conseguinte, é necessário estabelecer a correspondência entre classes de navegação e classes conceptuais e mesmo entre atributos das primeiras e atributos das segundas. A representação visual das classes de navegação ilustra-se na Figura 2.

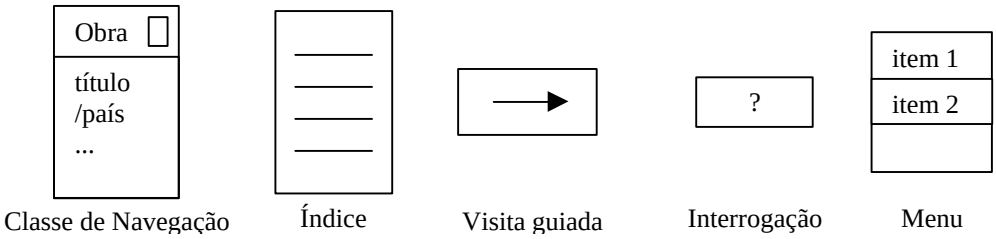


Figura 2. Simplificação das representações visuais dos estereótipos dos modelos de navegação

No modelo da estrutura de navegação são propostos 4 estereótipos fundamentais (ver acima): os índices, as visitas guiadas, as interrogações e os menus. Os primeiros representam listas, normalmente computadas, de ligações (âncoras) para instâncias da classe a que estão ligadas à saída (uma relação dirigida do índice para uma classe de destino), com base em parâmetros definidos pela classe de entrada (uma relação dirigida do índice para uma classe de destino). A informação providenciada pelas visitas guiadas é semelhante à anterior com a diferença que a navegação se pretende passo a passo, i.e. de cada instância da classe de saída apresentada neste contexto é possível aceder à seguinte e à anterior. As interrogações representam formas livres mais de pesquisa (que normalmente resultam em índices), ao contrário dos menus que tipicamente são enumerados finitos e pré determinados de ligações. Estes 4 conceitos, para além das ligações simples, são usados no estabelecimento de relações entre classes de navegação. O resultado final é de facto um modelo de navegação[16][4], tipicamente decomposto em dois diagramas (do espaço e da estrutura).

Modelos de apresentação

Na modelação da apresentação, no âmbito da UWE [12], introduzem-se um conjunto de estereótipos UML, cujo objectivo é a construção de representações esquemáticas (esboços tipificados) da interface com o utilizador. Essas representações constituem os modelos de apresentação (estáticos). Esses estereótipos incluem conceitos relativamente comuns, a saber: enquadramento (*frame*), colecção, formulário, texto, imagem, etc. Naturalmente, a metodologia não propõe estes arquétipos isoladamente, introduz também (recentemente) mecanismos (ainda em evolução) para a interligação destes com as classes e estruturas de navegação definidas nos modelos anteriores. É importante referir que, genericamente, as instâncias das classes de navegação (e.g. "Dali" e "Picasso" da classe "Pintor") serão apresentadas numa estrutura coerente entre instâncias da mesma classe, definida por um modelo de apresentação (e.g. "Texto" no canto superior esquerdo com o nome do artista, qualquer que ele seja). Note-se finalmente que o mesmo tipo de raciocínio se deve tecer relativamente às instâncias do modelo conceptual e os modelos de navegação e apresentação.

3. Especificação e Geração de XML

As propostas de ligação entre a XML [9] a UML [4] têm surgido sob as mais diversas formas. Entre elas deve mencionar-se o XMI (XML Metadata Interchange) [20] que, embora com um espectro mais abrangente que a UML, tem-se consolidado formato por excelência para a partilha de objectos, por exemplo, entre aplicações e ferramentas que trabalham sobre UML. A especificação XMI define um conjunto de regras rigorosas para a geração de: (1) DTDs (XML), e mais recentemente esquemas (XML Schema), a partir de modelos de classes; e (2) documentos XML a partir de modelos de objectos (instâncias dos anteriores). A Figura 3 ilustra a utilização das regras XMI na produção de XML a partir de modelos UML.

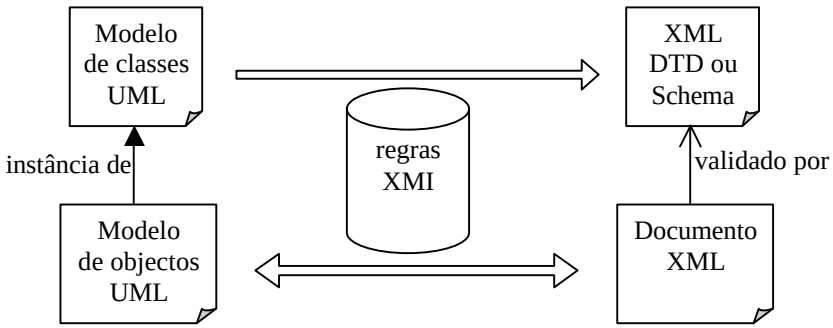


Figura 3. Geração de XML a partir de modelos UML

De uma forma simplificada dir-se-ia que classes e atributos UML originam a definição de elementos XML (ou tipos complexos) e respectivos atributos, sendo igualmente incluídas regras para a especificação de herança, estereótipos, etc. (veja-se [7][20] para uma descrição mais detalhada de regras).

A aplicação directa desta correspondência entre UML e XML através do XMI é a utilização dos esquemas, DTDs e documentos XML como formatos intermédios, que, sendo exportados de ferramentas de modelação que recorrem a diagramas UML, possam ser importados por outras ferramentas, regenerando os modelos, ou por aplicações que vêem assim a sua tarefa de interpretação da UML simplificada - já que podem partir da leitura de XML com todas as vantagens de suporte que daí advém (acesso a interpretadores, DOM , etc.). Segundo Carlson [7] é igualmente útil a capacidade que a correspondência acima mencionada providência para analisar vocabulários XML existentes ("reverse engineering"), à luz de uma aproximação mais adequada (de mais alto nível, a UML). Esta ligação UML/XML oferece também (e essencialmente) uma linha coerente para a modelação de aplicações XML a partir de modelos UML mais adequados nas fases primárias do desenvolvimento de software.

Sistemas hipermédia e XML

É no sentido anterior, do desenvolvimento de aplicações, que surgem as propostas (e projectos) da geração de sistemas hipermédia, em particular para ambientes Web [1][2][8][19]. Estas distinguem-se das anteriores, em especial pela utilização de uma metodologia em que se inclui o conjunto de modelos acima

referido (conceptual, navegação e apresentação). A dificuldade acrescida imposta por estas aproximações refere-se essencialmente:

- à inter-relação existente entre os modelos (e.g. atributos relacionados, classes de um modelo que referem as de outro, etc.);
- à inter-relação existente entre os conceitos da própria aproximação (meta-classes) e as regras e procedimentos que pressupõem na transição entre os modelos (e.g. um índice pressupõe uma interrogação à classe de navegação de destino com base em informação da de origem – os "pintores" de uma determina "época");
- à proximidade crescente que os modelos de navegação e apresentação têm com a sua concretização (a interface com o utilizador) no sistema hipermédia.

De entre as propostas de base refira-se as ferramentas ou bancadas de produção que utilizam o RMM [1] ou o HDM [8] como metodologia de concepção, embora usando o HTML como linguagem de destino - com recurso a folhas de estilo (CSS) e acesso a bases de dados. Mais recentemente, e na linha do trabalho da metodologia aqui usada como exemplo (a UWE), Kraus e Koch [19] propõem uma aproximação à publicação em XML, para a geração de aplicações Web, a partir de modelos concebidos de acordo com a aproximação UWE.

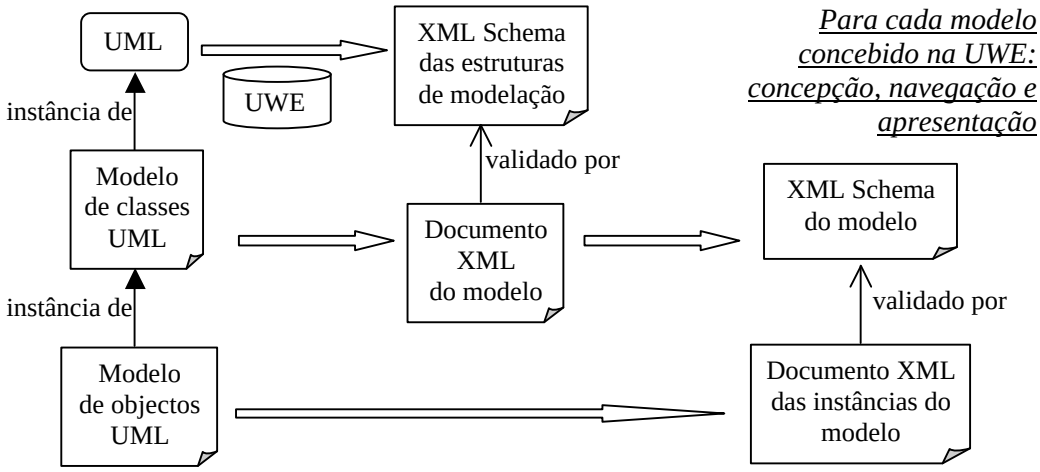


Figura 4. Documentos gerados na ligação da UWE à publicação XML

De acordo com os autores [18][19] a aproximação proposta integra ferramentas de especificação de modelos UML e de publicação XML, induzindo os modelos UML produzidos nas primeiras, por traduções sucessivas nas bancadas de publicação. Ao contrário do processo anteriormente apresentado na Figura 3 são gerados documentos XML (e os respectivos esquemas), quer para a descrição dos modelos quer para a descrição das instâncias dos modelos (os dados concretos). Na Figura 4 ilustra-se esta característica relevando o facto de que mesmo para os modelos de navegação e apresentação são gerados (especificados) quer os documentos XML do modelo, quer os documentos XML das instâncias. Para além das ferramentas acima mencionadas propõe-se a utilização de componentes Java do lado do servidor e, por exemplo, XSLT e folhas de estilo, introduzidas na bancada de publicação adoptada, para a concretização da aplicação Web.

4. A Tradução de Modelos em Aplicações XML

A construção de aplicações hipermédia em XML a partir de metodologias de concepção especificamente definidas para estes sistemas tem como objectivos basilares a criação de aplicações que:

- sigam uma separação clara entre a informação e os mecanismos de apresentação e interacção, quer para efeitos de modularidade, quer de reutilização, quer de adaptação a situações ou características dos dispositivos ou utilizadores ;
- adoptem as características de coer1ência na apresentação e interacção com a informação que nortearam originalmente a definição das metodologias de concepção com o intuito de promover melhorias na usabilidade destes sistemas.

Neste sentido é particularmente importante que, tendo seleccionado o XML para a representação das aplicações, as especificações nesta linguagem reforcem as regras impostas pelas metodologias, e tirem partido das características de validação e modularidade suportados nas arquitecturas XML. Assim a metodologia proposta neste artigo, define o processo e identifica os elementos ilustrados na Figura 5.

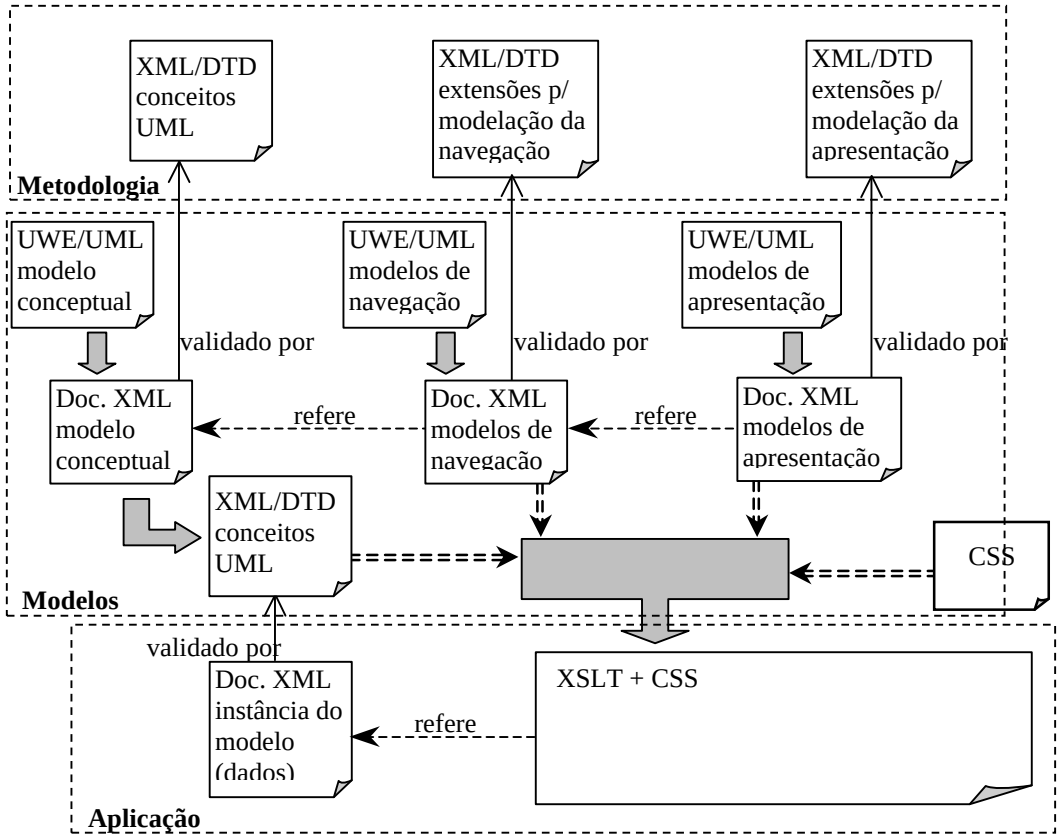


Figura 5. Processo de construção de aplicações XML a partir de modelos e metodologias

Os elementos identificados no topo (sob a designação Metodologia) constituem DTDs e introduzem respectivamente os constructos base: (1) da UML, na linha do XMI; (2) da UML estendida com os estereótipos correspondentes aos modelos de navegação da UWE; (3) da UML estendida com os estereótipos correspondentes aos modelos de apresentação da UWE. Estes DTDs não serão normalmente refeitos em sistemas hipermédia distintos e dependem exclusivamente da metodologia de concepção adoptada. No estado actual de desenvolvimento são especificados como DTDs e reflectem a aproximação UWE - deverão porventura transformar-se em esquemas XML tirando assim partido das características que estes oferecem.

No central rectângulo a traço interrompido apresentam-se os documentos que resultam do processo de concepção do sistema hipermédia. Do ponto de vista de quem concebe apenas as especificações UWE/UML e a especificação dos CSS são requeridos, sendo as restantes geradas automaticamente. Note-se que ao contrário da aproximação proposta por Kraus e Koch [19], afigura-se perfeitamente dispensável nesta arquitectura a geração de DTD ou esquemas XML para a criação de documentos XML que instanciam os modelos de navegação e apresentação. Estes funcionam apenas como padrões sendo as especificação das instâncias (dados da aplicação) exclusivamente definidas de acordo com o modelo conceptual.

Finalmente na componente de aplicação, o XSLT gerado a partir dos modelos extrai a informação de acordo com a instanciação do modelo conceptual apresentando-a coerente com os padrões de navegação, apresentação e interface (os CSS) especificados.

5. Um Caso de Estudo

Como caso de estudo concebeu-se um sistema hipermédia, sob o tema "Museu de Arte Interactivo". A concepção passou pelos passos determinados pela aproximação UWE, sendo posteriormente aplicado o processo de geração aqui proposto. O protótipo obtido em XML/XSLT/CSS executa-se sobre o Internet Explorer 6.

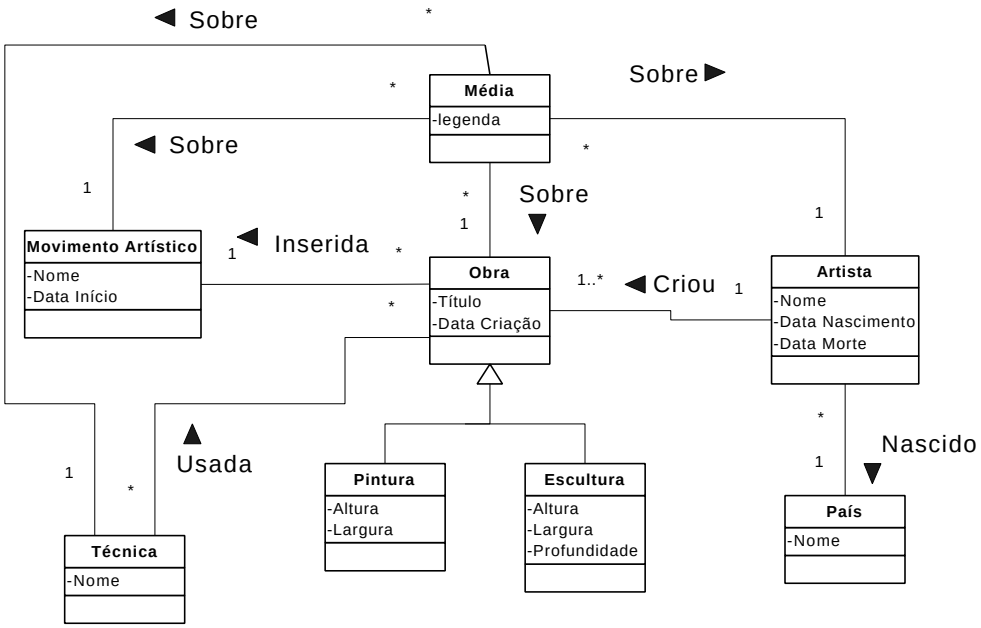


Figura 6. Excerto do Modelo Conceptual.

Nas Figura 6 ilustra-se o modelo conceptual da aplicação em estudo. Os elementos usados nesta versão simplificada incluem os conceitos de classe e atributo, associação e generalização da UML.

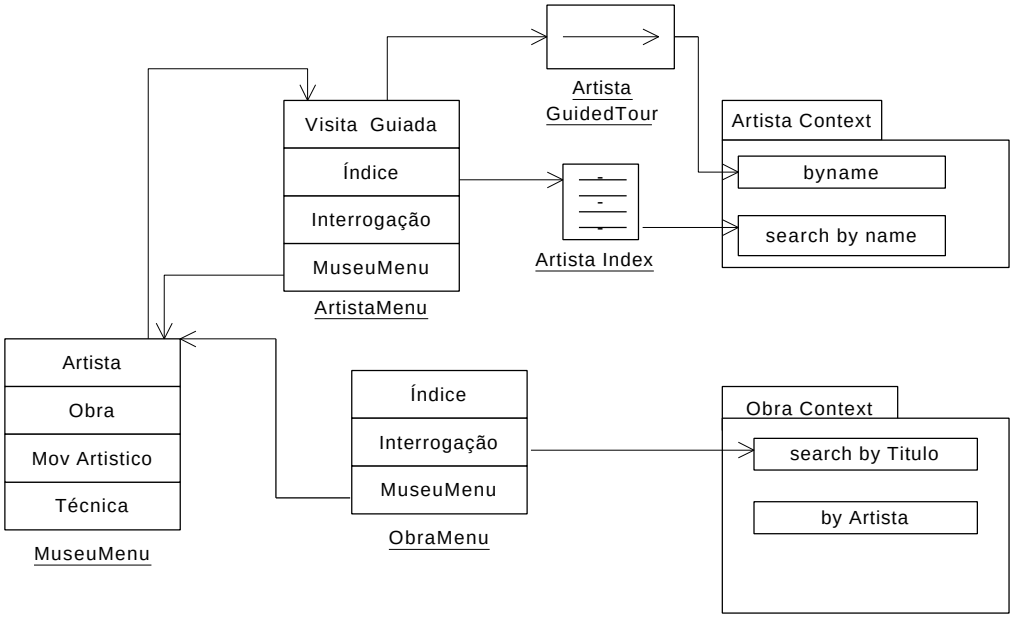


Figura 7. Excerto do modelos de Estrutura de Navegação

Uma vez definido este modelo concebeu-se ainda um modelo do espaço de navegação em que se omitem classes como "País" integrando-as em atributos da classe de navegação "Artista" – correspondente

à homónima de concepção mostrada na figura. Posteriormente foram introduzidas as estruturas de navegação, a saber menus, índices e visitas guiadas obtendo um modelo semelhante ao apresentado na Figura 7.

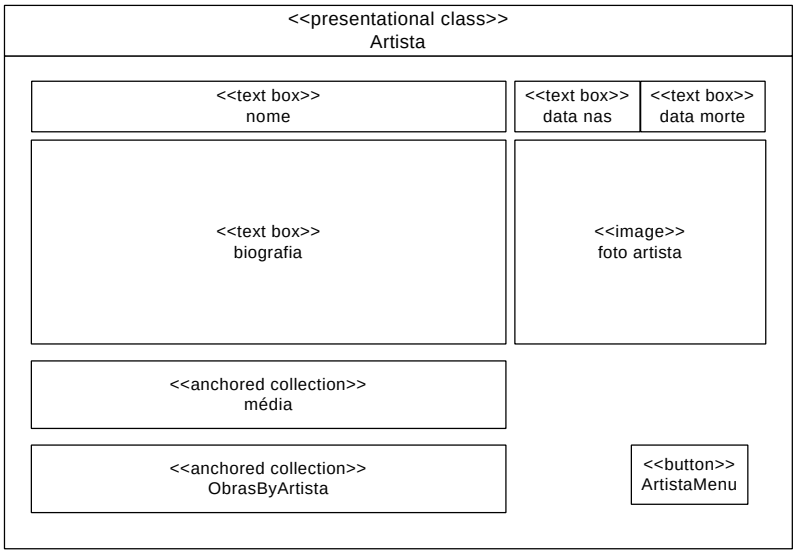


Figura 8. Excerto de um dos modelos de apresentação.

Na Figura 8 apresenta-se um dos modelos de apresentação desenvolvidos. Uma vez processados e combinados com uma especificação de CSS que concretiza os elementos de apresentação, estes modelos deram origem a um sistema de que se apresenta uma perspectiva na Figura 9.

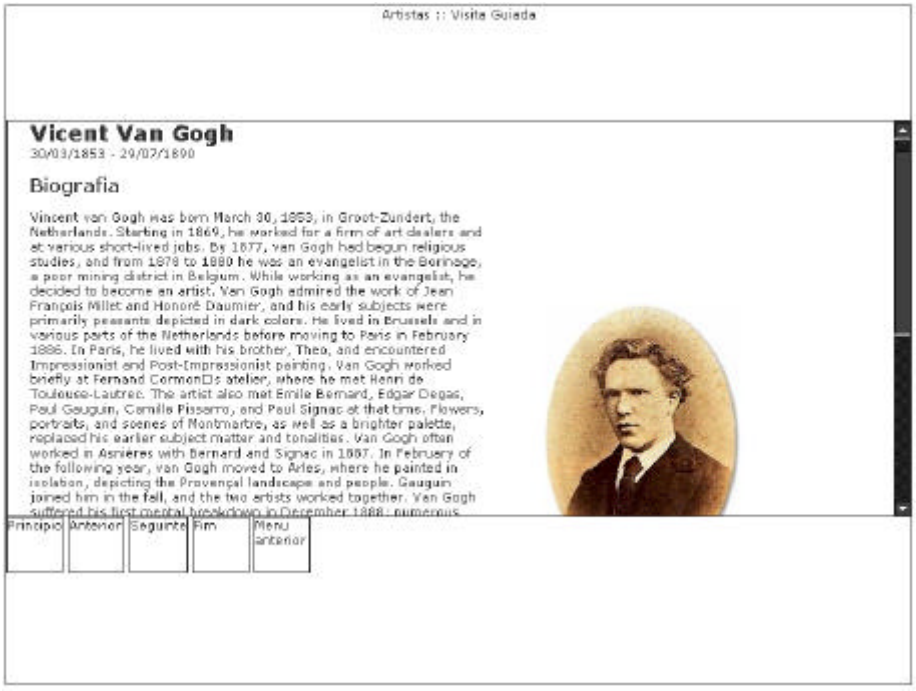


Figura 9. Visita guiada sobre Artistas.

Note-se que na figura se apresenta uma instanciação de um Artista, através do padrão definido pelo modelo de apresentação da figura anterior enquadrado por um outro padrão de apresentação associado às visitas guiadas.

6. Conclusões e Trabalho Futuro

O trabalho desenvolvido permitiu identificar os padrões e procedimentos necessários à construção de uma plataforma, sobre XML, que concretiza os conceitos propostos nas metodologias de sistemas hipermédia, tirando partido nos mecanismos de validação e verificação na garantia da coerência indispensável à criação destes sistemas. Por outro lado, levantou questões importantes relativas aos três níveis de suporte requeridos (meta-modelação, modelação e instanciação) e à sua concretização numa plataforma sobre XML.

Como trabalho futuro mencione-se a introdução dos aspectos dinâmicos da modelação dos sistemas, na apresentação, não nos sentidos propostos em [12][3], mas numa perspectiva de meios (media) activos de base temporal. Nesse sentido a inclusão de diagramas de sequência estendidos, ao nível da modelação será complementada pela adopção de dialectos baseados no tempo (SMIL [24][5][6]), para suprir as necessidades multimédia das aplicações em foco.

7. Bibliografia

[1] V. Balasubramanian, Alf Bashian and Daniel Porcher. “A large-scale hypermedia application using document management and Web technologies”. Proceedings of the eighth ACM conference on Hypertext, pages 134-145, Southampton United Kingdom, April 6 - 11, 1997.

[2] Denilson Barbosa. “XML and Hypermedia Applications” - CSC2524S 2000 Project

[3] Hubert Baumeister, Nora Koch & Luis Mandel. "Towards a UML extension for hypermedia design". In UML99 The Unified Modeling Language - Beyond the Standard, LNCS 1723, Fort Collins,USA, Springer, October 1999.

[4] Grady Booch, James Rumbaugh & Ivar Jacobson. “The Unified Modelling Language User Guide”. Addison-Wesley, 1999.

[5] Dick C. A. Bulterman. "SMIL 2.0 Part 2: Overview, Concepts, and Structure", IEEE MultiMedia, vol. 8, n.º 4, Oct-Dec, 2001.

[6] Dick C. A. Bulterman. "SMIL 2.0 Part 1: Examples and Comparisons", IEEE MultiMedia, vol. 9, n.º 1, Jan-Mar, 2002.

[7] David Carlson. “Modeling XML Applications with UML: Practical E-Business Applications”. Boston: Addison-Wesley, 2001.

[8] Piero Fraternali & Paolo Paolini. "Model-Driven Development of Web Applications: The Autoweb System", ACM Transactions on Information Systems, vol. 28, nº 4, Oct 2000.

[9] Michael Morrison, Frank Boumphrey & David Brownell (1999). “XML Unleashed”. Sams Publishing.

[10] Franca Garzotto, Luca Mainetti & Paolo Paolini. " Hypermedia Design, Analysis and Evaluation Issues", Communication of the ACM, vol. 38, nº 8, Aug 1995.

[11] Franca Garzotto, Paolo Paolini & Daniel Swabe. "HDM - A Model based Approach to Hypertext Application Design", ACM Transactions on Information Systems, vol.11, n.º 1, Jan 1993.

[12] Rolf Hennicker & Nora Koch. “Modeling the User Interface of Web Applications with UML”. In Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, Workshop of the pUML-Group at the UML 2001, October 2001.

[13] Rolf Hennicker & Nora Koch. "A UML-based Methodology for Hypermedia Design". In A. Evans, S. Stuart, and B. Selic, editors, UML'2000 - The Unified Modeling Language - Advancing the Standard, vol 1939 of Lecture Notes in Computer Science, York, England, Oct 2000. Springer Verlag.

[14] Tomás Isakowitz, Edward A. Stohr & P. Balasubramanian, "RMM - A Methodology for Structured Hypermedia Design", Communication of the ACM,, vol. 38, nº 8, Aug 1995.

[15] Tomás Isakowitz,, Arnold Kamis & Marius Koufaris. "The Extended RMM Methodology for Web Publishing", Working Paper IS98 -18, Center for Research on Information Systems, 1998.

[16] Ivar Jackobson, Grady Booch & James Rumbaugh. “The Unified Software Development Process”, Addison-Wesley, 1998.

[17] Nora Koch. "A Comparative Study of Methods for Hypermedia Development", Technical Report 9905, LudwigMaximilians - Universitt Munchen, November 1999.

[18] Nora Koch & Andreas Kraus. The expressive Power of UML-based Web Engineering. In Second International Workshop on Web-oriented Software Technology. CYTED, D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, editors, June 2002.
[19] Andreas Kraus & Nora Koch. “Generation of Web Applications from UML Models using an XML Publishing Framework”. In 6th World Conference on Integrated Design and Process Technology (IDPT), June 2002.
[20] OMG. “XML Metadata Interchange (XMI)”, OMG Document ad/2001-06-12, June 2001.
[21] Daniel Schwabe & Gustavo Rossi. “The Object Oriented Hypermedia Design Model”, Communication of the ACM, vol. 38, nº 8, Aug 1995
[22] Daniel Schwabe & Gustavo Rossi. "Developing Hypermedia Applications Using OOHDM", Workshop on Hypermedia Development, Pittsburgh, USA, Jun 1998.
[23] Daniel Schwabe, Gustavo Rossi & Simone D. J. Barbosa. "Systematic Application Design with OOHDM", ACM Hypertext' 96, USA, 1996.
[24] W3C. Synchronized Multimedia Integration Language (SMIL 2.0), Aug, 2001.

Geração de Conteúdos para Plataformas Móveis

Filipe Correia[1], Joel Varanda[1], João Correia Lopes[2], Alexandre Sousa[3]

[1] ParadigmaXis, Av. da Boavista, 1043, 4100-129 Porto.
http://www.paradigmaxis.pt/
{filipe.correia,joel.varanda}@paradigmaxis.pt
[2] Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto.
http://www.fe.up.pt/
jlopes@fe.up.pt
[3] Instituto Superior da Maia, Av. Carlos Oliveira Campos, 4475-690 Castelo da Maia.
http://www.ismai.pt/
avs@ismai.pt

Resumo Recentemente têm aparecido novos serviços na Web que disponibilizam informação personalizada eventualmente em dispositivos móveis, que vão muito para além dos tradicionais conteúdos devolvidos em HTML. A construção e manutenção de sistemas de média e grande dimensão que disponibilizem esses serviços, implica custos elevados. Este artigo descreve o trabalho levado a cabo no sentido de facilitar a construção e manutenção de aplicações Web, suportando a apresentação de conteúdos em múltiplas plataformas: PC, PocketPC e WAP. A arquitectura do sistema desenvolvido baseia-se na utilização de uma cadeia de filtros SAX que aplica modificações sucessivas aos conteúdos recolhidos de diversas fontes, sendo algumas destas modificações, transformações XSL. Os dados das diversas fontes são agrupados numa representação intermédia TipML, um dialecto de XML definido no âmbito deste trabalho.

1 Introdução

A *World Wide Web* (Web) tem, tradicionalmente, fornecido informação e serviços sobre HTML [5], tendo este vindo a ser o formato de representação por excelência neste meio. Uma nova geração de serviços de informação personalizada e serviços móveis, entre outros, têm vindo recentemente a disponibilizar conteúdos de modo bem aceite pelo público. No entanto, a construção de tais sistemas implica custos elevados de desenvolvimento e manutenção, nomeadamente quando possuem média ou grande dimensão.

Um problema encontrado frequentemente por quem desenvolve aplicações com uma interface Web, centra-se na necessidade de a tornar acessível ao maior número possível de utilizadores. Esta necessidade passa, muitas vezes, pelo suporte oferecido a múltiplas plataformas de modo a aumentar a abrangência de difusão.

Hoje em dia existem já variadíssimas plataformas com a capacidade de consultar informação disponível na Web e, com o aumento dessa variedade, maior se torna a complexidade de desenvolvimento de aplicações, de modo a que todas estas plataformas sejam suportadas; em última análise, o desenvolvimento poderá mesmo passar pela elaboração de sítios Web distintos, apropriados a cada uma das plataformas. Como resultado, os custos de desenvolvimento e manutenção tornam-se maiores do que seria de esperar, uma vez que se multiplicam por cada plataforma suportada.

Um outro problema consiste na dificuldade em separar os dados e o modo como estes são apresentados. Quando é conseguida eficazmente, esta separação torna-se bastante útil, uma vez que permite que cada uma das componentes da aplicação (dados e apresentação) possa ser alterada, facilmente, sem que a outra seja muito influenciada.

Este trabalho tem por objectivo facilitar a construção e manutenção de aplicações Web, suportando a apresentação de conteúdos em múltiplas plataformas: PC, PocketPC e WAP.

Para tal, foi projectado e construído um sistema, seguindo uma metodologia de programação literária, que permite o desenvolvimento de aplicações Web separando em camadas aplicacionais diferentes, a representação dos dados e o modo como estes são apresentados. Para além de HTML e JSP, foram utilizadas sobretudo tecnologias ligadas ao XML (*eXtensible Markup Language*): XSL, WML, HTML, SAX, DOM.

Na construção e manutenção dos programas desenvolvidos, foi utilizada a ferramenta *Open Source* de suporte à aplicação da programação literária, *dotNoweb* [12].

2 Representação intermédia de um documento

Inicialmente o uso de HTML tinha o objectivo de marcar informação de modo a definir um documento genérico, a ser apresentado na Web. Deveria ser dada ênfase aos elementos que definem o documento (como títulos, parágrafos, etc) e deveria ser dada pouca importância à formatação que essa informação teria junto do utilizador (como cores, tipos de letra, etc). Contudo, face à necessidade que existia, e existe, de especificar o modo como a informação é apresentada ao utilizador, a apresentação de HTML passou a ser rígida, centrada apenas na formatação de informação e não na sua estruturação como documento, independentemente do modo como é apresentado.

Existem várias razões que impedem que um determinado documento HTML, publicado na Web, seja disponibilizado a qualquer plataforma. Factores como capacidade de processamento, área de ecrã disponível e largura de banda, limitam as capacidades dos dispositivos a partir dos quais seria interessante aceder ao documento em questão.

Na tentativa de suportar o crescente número de plataformas existentes com a capacidade de acesso à Web, surge um aumento da complexidade de desenvol-

vimento de aplicações que tenham por objectivo ser visualizadas independentemente do dispositivo utilizado [9].

Se o objectivo inicial do HTML fosse mantido, existiria uma linguagem mais generalista, que podia ser interpretada correctamente por qualquer plataforma. Deste modo, determinado documento poderia ser consultado em qualquer das plataformas pretendidas sem que, para isso, fosse necessário construir e manter uma versão desse documento para cada uma delas. Contudo, segundo esta abordagem, não existiria qualquer tipo de informação relativa ao modo como a informação seria apresentada.

Na tentativa de estabelecer um compromisso entre a definição de um documento e a definição do modo como é apresentada determinada informação, foi construído um sistema de desenvolvimento de aplicações Web. Este sistema separa, em camadas aplicacionais distintas, a estruturação da informação de um documento, do modo como este será apresentado em cada plataforma. Assim, é permitido a um utilizador do sistema configurar informação textual e gráfica de modo a que esta possa ser visualizada e que permita a interacção em várias plataformas.

Por forma a obter uma representação de um documento independente da plataforma onde este irá ser apresentado, foi definida a linguagem intermédia TipML (*Tip Markup Language*). Esta representação intermédia é baseada num dialecto de XML, que define o posicionamento e ordem de apresentação de fracções de conteúdos, sem incluir informação relativa ao modo como esses conteúdos deverão ser apresentados. Esta linguagem genérica, nesta altura, tem por objectivo suportar as linguagens alvo HTML e WML [10].

Tomando como ponto de partida um documento escrito em TipML, é possível adaptá-lo às necessidades particulares de cada dispositivo. A linguagem é suficientemente rica para ser adaptada a um formato de documento estruturalmente complexo (como uma página HTML), mas também suficientemente simples, para ser adaptada a um documento com muito pouca informação estrutural (como uma página WML).

Esta abordagem permite construir apenas um documento, disponibilizando-o a qualquer uma das plataformas suportadas. Assim, a manutenção de uma dada aplicação multiplataforma é bastante simplificada; basta ser modificado apenas o documento TipML, em vez de ser necessário modificar todas as versões (para todas as plataformas) do documento. Uma segunda vantagem é a separação entre o conteúdo do documento e o modo como este é apresentado. Assim, a adaptação do documento a cada formato suportado, pode ser alterada conforme seja desejado.

3 Negociação e geração de conteúdos

Para que um dado documento Web possa ser visualizado na plataforma que lhe acede em dado momento, é necessário conhecer de que dispositivo se trata exactamente para, deste modo, a informação ser devolvida num formato que o dispositivo possa tratar adequadamente [3].

O protocolo HTTP (*Hyper Text Transfer Protocol*) fornece alguns mecanismos de apoio à negociação de conteúdo. Em cada pedido HTTP, efectuado por um cliente Web, são enviados, no cabeçalho desse mesmo pedido, os tipos de conteúdo (através de MIME — *Multi-Purpose Internet Mail Extensions*) que esse navegador Web aceita [14].

A partir desta informação é seleccionado o tipo de conteúdo a ser apresentado, de acordo com o dispositivo alvo correspondente. Assim, a selecção do tipo de conteúdo a apresentar é feita automaticamente conforme o tipo de dispositivo que aceda ao documento. Por cada dispositivo suportado existe uma folha de estilo XSL [1], preparada para adaptar um documento escrito em TipML, num documento específico ao dispositivo alvo.

Cada plataforma possui especificidades muito concretas. As características, das plataformas consideradas, que constituem maiores limitações à adaptação dos conteúdos a cada plataforma, são apresentadas na Tabela 1.

A representação em HTML para ser interpretada por um PC, é a que mais se aproxima de um documento escrito em TipML. Assim, torna-se mais simples fazer a adaptação do conteúdo para esta plataforma, tornando, no entanto, mais complexa a adaptação para outras plataformas.

No caso de se procurar uma apresentação em PocketPC, poderá também ser usado HTML mas, neste caso, é necessário obter uma representação diferente da anterior, por forma a não comprometer a usabilidade da aplicação construída. A dificuldade de maior relevância, é a representação da mesma quantidade de informação num espaço de ecrã de menores dimensões.

Quando se pretende que a aplicação construída seja acessível via WAP, a adaptação do conteúdo torna-se um pouco mais complexa, do que no caso anterior. Não só é necessário ter em atenção que a área de ecrã disponível é bastante limitada, como também é necessário ter em consideração as diferenças do formato de codificação da informação textual (WML) e gráfica (WBMP).

	PC	**PocketPC**	**WAP**
Formatos	HTML	HTML	WML
Imagens	gif, jpeg	gif, jpeg	WBMP
Dimensões ecrã	800x600, 1024x768	240x268	96x45, 101x65

Tabela 1. Capacidades de cada plataforma

4 Arquitectura Lógica do sistema

A arquitectura lógica do sistema de construção de aplicações Web, desenvolvido por forma a atingir os objectivos enunciados, é constituída por três camadas: a camada de dados, a camada de integração e a camada de apresentação.

A arquitectura do sistema baseia-se na utilização de uma cadeia de filtros SAX que aplica modificações sucessivas aos conteúdos recolhidos de diversas fontes [11], sendo algumas destas modificações transformações XSL.

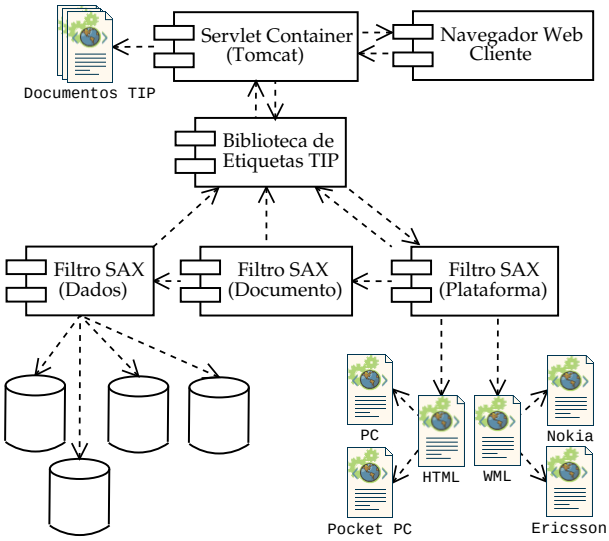


Figura 1. Diagrama de Componentes do Sistema

Os conteúdos podem ser obtidos de uma ou mais bases de dados relacionais e são depois convertidos para XML pela camada de dados sendo, neste formato, introduzidos na cadeia de filtros SAX.

A camada seguinte de integração, trata de compor os dados das diversas fontes construindo um documento que constitui uma representação intermédia em TipML.

A camada de apresentação transforma o documento TipML de modo a que este fique de acordo com as capacidades do dispositivo cliente. Após a negociação de conteúdo, está identificada a plataforma de apresentação e basta, por isso, incluir uma folha de estilos XSL para o formato pretendido.

Após todas as modificações, os conteúdos estão adaptados ao dispositivo a que se destinam. Deste modo, o sistema desenvolvido tem a capacidade de efectuar a saída para vários formatos de dados, de acordo com as características de cada uma das plataformas suportadas.

5 Arquitectura Física do sistema

Por forma a ilustrar a arquitectura física do sistema desenvolvido, apresenta-se, usando a linguagem de modelação UML, o Diagrama de Componentes e o Diagrama de Distribuição, respectivamente na Figura 1 e na Figura 2.

O problema em questão não sugeriria, à partida, a utilização de um *parser* de XML uma vez que o objectivo consistia em ler a informação de uma base de dados, para além de ela poder ser obtida de ficheiros XML. Contudo, é possível

emular a leitura de dados XML como se de um *parser* (que suporte DOM ou SAX) se tratasse, sendo, na realidade, esses dados lidos de outra fonte.

O modo de funcionamento de um *parser* SAX, baseado em eventos, favorece a sua adaptação de modo a servir os propósitos pretendidos. Um factor favorável ao uso do SAX neste âmbito, é o processamento sequencial da informação e a possibilidade de modularização de diversas componentes pelo recurso a uma cadeia de filtros. Pelo uso de filtros SAX os dados são alterados em sequência, imediatamente após a leitura e análise gramatical de determinada secção de informação.

O DOM revela-se ideal para armazenar informação quando o modelo de dados pretendido é já uma árvore. Assim, com o seu uso, é possibilitado o acesso aleatório aos dados, mas para este caso não é vantajosa a sua utilização, uma vez que os dados poderão ser acedidos pela sua ordem original. Assim, o tempo e recursos computacionais adicionais necessários para a construção em memória da árvore DOM não seriam, de modo algum, compensados pela possibilidade de acesso aleatório aos dados.

Foi, assim, criado um filtro SAX que se encarrega de fazer a interacção com as bases de dados, obtendo a informação lá contida, como se da leitura de um ficheiro XML se tratasse, tornando-se este processo completamente transparente para o resto da aplicação. A este *leitor* de informação podem ser adicionados os filtros que se pretendam de modo a modificar, de algum modo, os dados em XML dele obtidos. O acesso às base de dados é efectuado por intermédio de uma *API* disponibilizada por um *Driver* JDBC. Será este *driver* que efectuará a interacção propriamente dita com as bases de dados.

Face aos propósitos da aplicação foram utilizados três filtros (ver Figura 1 [6]), sendo o primeiro, como já referido, um filtro de leitura de conteúdos da base de dados. Este filtro, uma vez tendo iniciado a aquisição de informação, irá lançar eventos correspondentes a uma estruturação em XML dos dados obtidos. Tais eventos são tratados pelo filtro seguinte na cadeia.

O segundo filtro, tem o papel de combinar os conteúdos recebidos (provenientes de base(s) de dados relacional(ais)) com a definição do posicionamento de componentes no documento (proveniente do documento *TIP* a ser processado). A combinação referida é feita por meio de uma folha de estilos, já que o filtro SAX não é mais que um transformador XSL. O resultado final da transformação consiste num documento escrito em TipML.

O passo seguinte é dado pelo terceiro e último filtro e consiste na adaptação do documento, agora escrito em TipML, para a plataforma pretendida. Para tal, os eventos do segundo filtro são capturados e, sendo este também um filtro transformador de XSL, é-lhes aplicada uma segunda folha de estilo, específica ao formato de saída em questão (HTML ou WML). A cada folha de estilo que é possível aplicar, são associadas ainda configurações específicas à plataforma em que o formato em questão será utilizado.

O documento *TIP* referido na Figura 1 é uma página JSP adaptada para servir alguns propósitos específicos, como será referido mais à frente. No entanto,

este facto permite que a cadeia de filtros já apresentada seja gerida por uma biblioteca de etiquetas de JSP [13].

Estas etiquetas são usadas na definição de certos blocos de informação num documento *TIP* e a elas está associada a lógica que cria cada filtro e faz com que estes funcionem em conjunto. Todos os blocos de informação por elas marcados num documento *TIP* constituem, de algum modo, informação necessária ao funcionamento da cadeia de filtros SAX.

Pelo uso de uma biblioteca de etiquetas JSP é excluída do documento *TIP* qualquer lógica relacionada com a cadeia de transformação e o facto de um documento *TIP* se tratar, no fundo, de uma página JSP possibilita ainda que os conteúdos das etiquetas sejam gerados dinamicamente por meio de *Scriptlets*, inclusive a partir de parâmetros externos (uma vez que o conteúdo das etiquetas acabará por ser código XML, muitos dos princípios apresentados em [8] são aplicáveis).

O funcionamento do sistema é desencadeado por um pedido por parte de um navegador Web. Esse pedido será dirigido ao *Tomcat*, que servirá simultaneamente como servidor Web e *Servlet Container*. Como tal, o *Tomcat* obtém o documento *TIP* especificado no pedido, lê dele os dados necessários e inicializa a cadeia de filtros SAX, para que o documento seja transformado no formato apropriado.

Quando construída uma aplicação em *TIP*, os componente executáveis de todo o sistema são distribuídos como apresentado na Figura 2.

O *Tomcat* encontra-se a correr numa máquina específica para o efeito onde, num caso limite, poderiam estar também as bases de dados a utilizar. Contudo, esta não é a configuração mais vulgar, uma vez que a possibilidade, oferecida pelo sistema, de serem utilizados dados de várias fontes distintas, permite que as bases de dados se encontrem distribuídas. Assim, todos os nós estão ligados à *Internet*, por intermédio da qual são efectuadas as passagens de informação entre os nós. Os dispositivos cliente existentes constituem nós de *hardware* distintos e, cada um deles, poderá requisitar documentos *TIP* ao servidor Web. Os dispositivos permitidos, como foi já referido, poderão ser de natureza variada (dispositivos WAP, PC e PocketPC), correndo, em cada um deles, um navegador Web apropriado, com a capacidade de requisição de documentos ao servidor.

6 Detalhes de Implementação

Partindo da definição da arquitectura apresentada anteriormente, foi necessário desenvolver todos os componentes necessários ao funcionamento do sistema. As tecnologias utilizadas foram JSP, XML, XSL e a plataforma Java [7].

A lógica de negociação e adaptação de conteúdo foi implementada com recurso a uma biblioteca de etiquetas JSP, que irá residir no *Tomcat Web container*.

No âmbito da adaptação de conteúdo foram criadas duas etiquetas centrais: a etiqueta <`tip:data`> e a etiqueta <`tip:process`>.

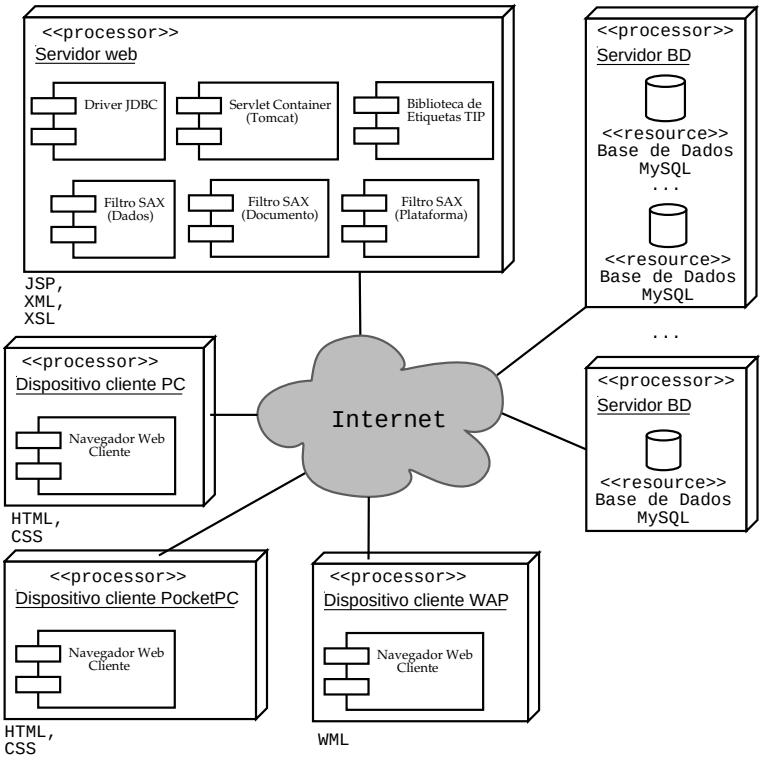


Figura 2. Diagrama de Distribuição do Sistema

Para acesso às bases de dados é utilizada a informação do corpo da etiqueta <tip:data>. Tal informação está estruturada segundo um dialecto de XML bem definido, como apresentado na Figura 3.

A etiqueta <tip:process> define a lógica que instancia toda a cadeia de filtros SAX com vista a gerar a versão do documento no formato pretendido: WML ou HTML com folhas de estilos CSS externas [2]. No corpo desta etiqueta é gerada (por JSP) uma folha de estilo XSL que é por este meio passada aos filtros SAX, intervindo nas transformações efectuadas.

Posteriormente é aplicada uma transformação XSL específica à plataforma de visualização alvo; existe por exemplo, uma para HTML num PC e outra para HTML num PocketPC.

7 Teste e exploração

Por forma a avaliar a viabilidade da arquitectura proposta, a adequação da linguagem intermédia desenvolvida e os resultados das apresentações obtidas em HTML e em WML foram construídas duas aplicações de teste. Estas aplicações

```
[...]
<tip:data>
   <server>
      <name>localhost</name>
      <database>
         <name>BD</name>
         <query record="{}noticia"{}>
            select * from noticias order by date desc
         </query>
      </database>
   </server>
</tip:data>
[...]
```

Figura 3. Excerto de um documento TIP

baseiam-se em aplicações existentes em funcionamento na Web permitindo, deste modo, a validação das funcionalidades disponibilizadas e a detecção de aspectos que podem ser melhorados em desenvolvimentos futuros.

A primeira aplicação baseia-se em `http://www.digito.pt`, que se destina à divulgação de notícias sobre ciência e tecnologia, de onde foram obtidas as notícias inseridas numa base de dados construída para o efeito. A notícia a apresentar é passada como parâmetro a uma consulta à base de dados, sendo aplicadas de seguida as cadeias de filtros SAX por forma a obter uma representação em HTML para PC, uma representação em HTML para PocketPC e, finalmente, uma representação em WML que foi visionada num emulador. O conteúdo foi apresentado com resultados bastante satisfatórios nas várias plataforma alvo.

A segunda aplicação baseia-se em `http://www.newportex.pt`, que se destina à divulgação de notícias sobre têxteis, e permitiu concluir que os resultados são melhores, nomeadamente em termos de navegação, quando os conteúdos são pensados directamente em TipML ao invés de serem adaptados de páginas existentes, como foi o caso nos dois exemplos descritos. Tal como com a aplicação anterior, esta aplicação de teste também expôs diversos melhoramentos que foram, ou estão em vias de ser, incorporados no TipML.

A título de exemplo, apresentam-se nas Figuras 4, 6 e 5 ilustrações, retiradas do funcionamento do sistema desenvolvido, em uso para a segunda aplicação referida.

8 Conclusões e Trabalho Futuro

Os resultados obtidos permitem-nos concluir que a arquitectura proposta e implementada satisfaz globalmente os objectivos, possibilitando a publicação, em formatos apropriados a vários dispositivos, de informação proveniente de uma ou mais fontes, nomeadamente bases de dados relacionais.

Por forma a melhorar a eficiência, as folhas de estilo XSL estáticas podem ser compiladas com JAXP [4] e as folhas de estilo XSL geradas dinamicamente a partir de JSP poderiam ser mantidas numa cache.

Figura 4. Interface do Newportex em HTML

Em termos de imagens, a apresentação pode ser muito melhorada através de redimensionamento, uma vez que nesta altura apenas é decidido se são incluídas ou não no documento final, ou mesmo através de conversão de formatos, por exemplo de GIF para WBMP para WAP.

Outros melhoramentos em vista incluem o suporte para sistemas legados, envolvendo a tradução para TipML de páginas existentes em HTML, e a extensão a novas plataformas através da construção de novas folhas de estilo.

Referências

1. Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo G utentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. Extensible Stylesheet Language (XSL) version 1.0. Technical report, W3C, October 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/.
2. Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. Cascading style sheets, level 2 - CSS2 specification. Technical report, W3C, May 1998. http://www.w3.org/TR/1998/REC-CSS2-19980512.
3. K. H. Britton, R. Case, A. Citron, R. Floyd, Y. Li, C. Seekamp, B. Topol, and K. Tracey. Transcoding: Extending e-business to new environments. *IBM Systems Journal*, 40(1), 2001.
4. Eric M. Burke. *Java and XSLT*. O'Reilly & Associates, Inc., September 2001.
5. James Clark and Steve DeRose. HTML 4.01 specification. Technical report, W3C, December 1999. http://www.w3.org/TR/html401/.
6. Jim Conallen. *Building Web Applications with UML*. Addison-Wesley, December 1999.
7. Elliotte Rusty Harold. *Processing XML with Java*. Addison-Wesley, 2001.
8. Marshall Lamb. Generate dynamic XML using JavaServer Pages technology. Technical report, ibm.com/developerworks, December 2000.
9. Benoît Marchal. *Applied XML Solutions*. Sams Publishing, August 2000.

Figura 5. Interface do Newportex em WML

10. OMA. Wireless markup language specification version 1.3. Technical report, WAP Forum, Ltd, February 2000. http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf.
11. O'Reilly. *SAX2*. O'Reilly & Associates, Inc., January 2002.
12. Alexandre Valente Sousa. Literate programming applied to software maintenance and teaching computer science. *Perspectivas XXI*, 4(8):101–120, November 2001.
13. Sun. JavaServer pages, 2002. http://java.sun.com/products/jsp/.
14. W3C. HTTP request fields, May 1994. http://www.w3.org/Protocols/HTTP/HTRQ_-Headers.html.

Figura 6. Interface do Newportex em HTML para PocketPC

XATA 2003

XML: Aplicações e Tecnologias Associadas

Braga – 13 e 14 de fevereiro

Programa da Workshop

14 Fev.	S8(16.30h) Metainformação	[*Catalogação e distribuição de Meta-Informação Geográfica em XML*] - **Jorge Gustavo Rocha**, DI-UM; **André Nuno Faria**, DI-UM; **Paulo Brito**, DI-UM; [*Partilha de Conteúdos e Semântica Multimédia*] - **José Manuel Torres**, INESC-Porto; [*Desafios e contribuições resultantes da utilização da XML na norma ISO/MPEG-7 e suas aplicações*] - **Rui J. Lopes**, Comp.Dep.-Lancaster University; **Adam T. Lindsay**, Comp.Dep.-Lancaster University; **David Hutchison**, Comp.Dep.-Lancaster University; [*Omnipaper- Descrição de recursos de noticias digitais em RDF*] - **Teresa Susana Mendes Pereira**, DSI-UM; **Ana Alice Baptista**, DSI-UM;

Catalogação e Distribuição de Meta-Informação Geográfica em XML

Jorge Gustavo Rocha and Nuno Faria and Paulo Brito

Departamento de Informática
Universidade do Minho

Resumo Neste trabalho apresentamos todo o ciclo de vida do desenvolvimento de uma aplicação de catalogação de meta-informação geográfica, com base na norma ISSO 19115. Dado que a interpretação das normas propostas pela ISO são especificações UML, e que desses modelos se pode derivar automaticamente (usando o Relational ROSE) Schemas XML, optamos por usar o próprio schema da norma como estrutura fundamental da própria aplicação de catalogação. Com base nesta filosofia, entendíamos que seria fácil reestruturar a aplicação sempre que surgissem alterações à norma (e consequentemente ao schema da norma). A aplicação de catalogação está desenvolvida em JAVA, e é criada uma estrutura de navegação em árvore, de acordo com a estrutura do schema da norma. O programador escolhe que ramos da árvore são visíveis, e a que ramos estão associados os formulários. São discutidos ainda, neste artigo, as questões do armazenamento e disponibilização da meta-informação, recorrendo a modelos híbridos de dados (SGBDR com documentos XML) e a serviços XML.

1 Introdução

1.1 Meta-Informação

A meta-informação surge muito naturalmente quando há necessidade da mesma: ou por uma questão de dimensão, ou porque a informação constitui um recurso crítico. No caso dos SIG, a informação geográfica ainda é um recurso crítico, com um peso muito significativo nos custos de uma aplicação. Por exemplo, em [Hoh98], é dito que, como regra geral, 50% dos custos de uma aplicação SIG estão relacionados com a aquisição e preparação dos dados.

Interessa, por isso, concentrar a informação sobre todos os recursos conhecidos, relacionados com o domínio, num único catálogo. A informação sobre esses recursos, a que chamamos meta-informação, contemplará características como o contacto do fornecedor, o preço e condições de aquisição, descrição textual da informação, suportes disponíveis, etc.

1.2 Normalização

Havendo mais do que uma iniciativa para catalogar a informação geográfica, e havendo muitos produtores espalhados pelo mundo, surge de imediato a questão

da normalização. A normalização começou por ser tratada a nível dos mercados mais dinâmicos, os Estados Unidos e Canadá, e também ao nível a Comunidade Europeia.

Nos Estados Unidos, existe desde os anos 60 um comité criado pela administração central, *The Federal Geographic Data Committee* (FGDC), com a participação de muitas instituições governamentais, para coordenar a promoção e o uso da informação geográfica. Este organismo foi responsável por muitas iniciativas relacionadas com a definição das facetas a catalogar e com a disponibilização da meta-informação geográfica.

Na Comunidade Europeia, as iniciativas relacionadas com a catalogação e disponibilização de meta-informação, para além dos esforços de alguns países membros, foram conduzidas pelo CEN/TC 287 - Comité Técnico de Informação Geográfica do Comité Europeu de Normalização.

Hoje em dia, o âmbito dos esforços de normalização é mundial e, por isso, tentam-se reunir esforços no sentido de criar um consenso sobre a catalogação de informação geográfica. No entanto, ainda estamos perante três grandes iniciativas:

Normalização Oficial		Normalização de Facto
Âmbito Europeu	Âmbito mundial	
Comité Técnico CEN/TC 287 (adormecido)	Comité Técnico ISO/TC 211	Open GIS Consortium

O trabalho apresentado foi desenvolvido com base na norma especificada pelo Comité ISO. Este decisão foi assumida pelo Instituto Geográfico Português (IGP), que é o organismo nacional responsável pelo acompanhamento da produção de informação geográfica.

O standard a que nos referimos é o ISO 19115, que define a meta-informação genérica para informação geográfica. Outros standards relacionados, que definem tipos de dados geográficos específicos, determinadas extensões, etc, são definidos em standards da série 19100. Actualmente (início de 2003) muitos destes standards relacionados ainda não foram publicados oficialmente, e carecem de descrições textuais que ajudem à sua interpretação.

2 A norma ISO/TC 211

As especificações produzidas pelo Comité ISO/TC 211 são escritas em *Unified Modeling Language* (UML).

O recurso a uma linguagem de especificação como o UML tem duas grandes vantagens: a primeira tem a ver com a utilização de uma linguagem amplamente aceite no ensino e na indústria, independente dos fabricantes e, por outro lado, permite trabalhar a um nível conceptual adequado a uma tarefa que envolve muitos autores de muitos países diferentes.

A norma é uma estrutura abstracta para descrever informação geográfica, através da definição dos elementos da metadata e do estabelecimento de uma terminologia, um conjunto de definições e um conjunto de mecanismos de extensão

comuns. Esta forma abstracta da norma pode levar a que haja interpretações um pouco diferentes de produtor para produtor.

2.1 Conversão de UML para XML

No caso de estudo apresentado, optamos por usar o mapeamento da especificação UML para XML desenvolvido pelo *Digital Geographic information Working Group* (DGIWG). O DGIWG foi formado para desenvolver standards para suportar a troca e partilha de informação geográfica em formato digital no âmbito dos países subscritores do tratado do Atlântico Norte (OTAN). Esta passagem de UML pra XML apresenta algumas dificuldades, já não o comité ainda não teve tempo para publicar a documentação que ajuda a interpretar a norma.

Ao adoptar esta interpretação da DGIWG, apostamos numa interpretação comum aos países da OTAN, que provavelmente prevalecerá sobre outras interpretações.

Como a norma em causa, ISO 19115, Geographic Information - Metadata, se socorre de muitas outras normas relacionadas, o resultado não é uma único XML Schema, mas sim uma colecção com vários schemas.

3 Ferramenta de Catalogação

O trabalho de catalogação é sempre fastidioso e por isso, toda a ajuda que uma aplicação de catalogação possa proporcionar, é bem necessária. Por isso, foi desenvolvida uma ferramenta de catalogação, em que o desenho dos formulários de entrada de dados foi muito cuidado. Todo este trabalho do desenho dos formulários foi feito manualmente, não sendo possível sistematizar a conversão de um schema (com cerca de 30 páginas) para um conjunto de formulários. O desenho da estrutura e interligação dos formulários acabou por ser, até agora, a tarefa mais morosa. O desenho complica-se porque, por exemplo, alguns dos formulários recorrem à visualização de um mapa, que obriga a ter funcionalidades como zoom e pan, para seleccionar áreas a que a informação geográfica se refere ou não.

O desenho desta interface divide a tarefa segundo cinco grandes facetas diferentes. Essas são:

1. Identificação,
2. Metadados,
3. Representação Espacial,
4. Qualidade e
5. Distribuição.

Cada um delas, é por sua vez dividida em facetas mais pequenas, até um ponto em que já não se divide mais, e aparece um formulário com diversos campos que caracterizam essa faceta. Só as facetas que já não se dividem mais (as

folhas desta árvore) têm um formulário associado. Esta forma hierárquica visualizada graficamente permite navegar mais facilmente sobre um grande número de formulários, sem perder nunca o contexto dos formulários que vão surgindo.

Por esta razão, a ambiente de catalogação consiste numa área dividida verticalmente em duas partes: à esquerda apresenta-se a árvore de navegação e à direita apresentam-se os formulários associados às folhas.

3.1 Imposição de restrições

Está a ser utilizada uma cor, quer visível na hierarquia, quer nos formulários, para distinguir as facetas com campos obrigatórios e esses mesmos campos, de forma a permitir uma catalogação mínima. Como exemplificado (cf. Figura 8), sempre que o utilizador selecciona um campo do formulário, aparece uma descrição do mesmo. O elevado número de campos obriga-nos a apresentar esta descrição ao utilizador. A hierarquia das facetas de catalogação também permite algumas funcionalidades relacionadas com campos repetitivos. Permite adicionar repetições, que ficam imediatamente disponíveis para preenchimento, e permite remover qualquer uma destas repetições.

3.2 Requisitos

3.3 Tecnologia utilizada

JAVA O programa de apoio à catalogação foi escrito em JAVA, dado que os seus executáveis correm numa grande variedade de plataformas, contemplando os produtores de Informação Geográfica (IG) que têm estações de trabalho Unix e os que têm postos baseados em PCs ou Macs. A plataforma JAVA, dispondo de muitas classes para o processamento de XML, também se mostrou muito adequada a essa tarefa, embora a performance no carregamento dos formulários deixe muito a desejar. No que diz respeito ao desenho da interface, esta linguagem não é tão adequada, pois é muito moroso o desenho das dezenas de formulários necessários.

Processamento do XML Todo o processamento do XML (parsing, XPath, XSL) é feito com base do XDK da Oracle [2].

O XDK da Oracle, entre outras funcionalidades, apresenta:

XML Parser for Java
XSLT Processor for Java
XML Schema Processor for Java

3.4 Soluções e problemas

Árvore de Navegação Havendo uma estrutura associada à norma de catalogação que é dada por um schema, desde logo nos pareceu que seria lógico usar essa própria estrutura para guiar a introdução dos metadados. No entanto, não é directa a forma de utilizar a árvore do DOM como árvore de interacção.

Onde pendurar os formulários É preciso decidir que ramos da árvore viram formulários (e esses ramos deixam de poder ser subdivididos).

Como tratar os campos repetitivos e opcionais Há formulários que se podem repetir, que correspondem a sub-árvores que tem uma cardinalidade variável. Também há formulários, por corresponderem a facetas opcionais, podem desaparecer do próprio DOM, e por isso desaparecem da própria interface.

Etiquetas contextualizadas na árvore de interacção À medida que se preenchem os formulários, é necessário visualizar a árvore de interacção com etiquetas dependentes da catalogação efectuada. Por exemplo, relacionado com o problema anterior, uma lista repetitiva de contactos pode ser visualizada com o nome do contacto na árvore de interacção, tornando mais fácil a tarefa de edição. No nosso caso específico também ocorrem sub-árvores associadas à mesma entidade do schema, mas cujo tipo varia e, por isso, dá origem a facetas diferentes de catalogação. Se a árvore fosse baseada apenas nas etiquetas, o utilizador ficaria sem possibilidade de distinguir estas sub-árvores, bem diferentes.

Tipos enumerados Como é natural numa norma, existem dezenas de atributos que assumem valores de tipos enumerados. Todos estes tipos enumerados estão descritos num documento XML próprio que é processado pela aplicação. A partir desse documento, são preenchidas todas os elementos da interface que permitem a escolha num domínio finito (combo boxes, list boxes, etc).

Ajuda A ajuda associada a cada campo do formulário está descrita num documento XML que associa a cada tag a respectiva descrição. Esse documento é também processado pela aplicação para mostrar essa ajuda ao utilizador.

Catalogações modelo Por ser uma ferramenta de catalogação baseada em XML, e porque a própria estrutura de navegação se constrói a partir do XML, optamos por criar um conjunto de documentos de base que servem de modelo à criação de novos documentos. Como é trabalhoso criar um novo documento a partir do zero, fica ao critério do próprio utilizador ou da instituição a que pertence, o desenvolvimento de uma conjunto de documentos padrão que permitam acelerar o processo de catalogação.

4 Sistema Distribuído de Catalogação

Embora a reunião e disponibilização dos catálogos de informação geográfica ainda não esteja disponível, é referida a arquitectura adoptada para o sistema distribuído de catalogação que envolve as instituições produtoras e consumidores de IG. Esta arquitectura corresponde a um primeiro protótipo que foi desenvolvida no âmbito do estágio do aluno finalista Paulo Monteiro.

Arquitectura Na solução proposta, cada produtor de IG possui uma aplicação que lhe permite editar e gravar os seus respectivos metadados, em documentos XML, como foi já referido.

Cada instituição produtora ou detentora de informação geográfica, é convidada a enviar esses documentos para o SNIG, por forma a integrarem o catálogo de informação geográfica nacional, e consequentemente tornarem-se pesquisáveis por todos os potenciais utilizadores/clientes.

Para armazenar o catálogo que reúne toda a meta-informação, optamos por utilizar um SGBD Oracle onde, para além de armazenar todos os catálogos sob a forma de documentos XML, guarda também alguma meta-informação sobre esses catálogos, necessária à gestão dos mesmos.

A arquitectura adoptada é constituída por:

Aplicação de catalogação Disponibilizada a todos os produtores de IG, que permite o carregamento dos metadados atendendo ao ISO 19115;

Gestor de Versões Faz a sincronização dos metadados catalogados em determinada instituição com a informação existente no SNIG;

SGBD Oracle Centraliza os metadados catalogados segundo a norma ISO. Os catálogos, que são documentos XML, estão a ser guardados em campos BLOB do Oracle;

Interface WWW Para consulta dos catálogos privilegia-se a Internet. Para isso, é necessário uma interface de pesquisa que permita encontrar os recursos que se procuram (por desenvolver);

Interface XML (Web Services) Para consulta dos catálogos por software. Este serviço é disponibilizado para permitir que outras aplicações possam consultar os catálogos de uma forma automática (em desenvolvimento).

Base de Dados Híbrida: SGBD com documentos XML A base de dados é, neste momento, constituída por apenas quatro tabelas, a saber:

- Instituições: Nesta tabela são guardadas as informações respeitantes às instituições produtoras de IG.
- Utilizadores: Aqui são guardadas as informações relativas aos utilizadores que podem aceder ao sistema para inserção/alteração de IG, sendo esses utilizadores pertencentes às diversas instituições registadas na BD.
- XML: Esta tabela armazena todos os documentos XML gerados pela aplicação de catalogação. A tabela é actualizada pelo Gestor de Versões.
- Ficheiros: Serão guardados nesta tabela todos os ficheiros referidos nos documentos XML como, por exemplo, os que contêm imagens de amostras nos formatos JPG, GIF, etc.

5 Conclusão

Ao descrever de forma tão resumida o desenvolvimento de uma aplicação de catalogação baseada em XML, queremos destacar alguns dos especificidades do problema.

A primeira, advém do facto de trabalharmos com um schema gerado automaticamente a partir do UML. Sendo uma transformação sistemática, não contém optimizações nem interpretações que conduzissem a uma formulação mais simples e mais adequada à manipulação. Ou seja, o schema base é complexo. Além disso, não é fácil encontrar ferramentas que processem integralmente o schema gerado.

Em segundo lugar, a aplicação acabou por constituir um trabalho muito moroso pela dimensão do problema: existem dezenas de formulários, cada qual com diversos campos, o que resulta em centenas de campos. A informação a preencher em cada campo não é, nalguns casos, por falta de guiões publicados pelo ISO, facilmente inferida. Isso consumiu muito tempo de análise. Para essa análise, muito contribuiu o trabalho do extinto CNIG.

O aspecto mais positivo a destacar, no resultado final, é a facilidade de parametrização da própria aplicação. A mesma permite suporta a alteração do schema de base, já que a interface é estruturada com base numa instância. Permite também melhorar permanentemente os textos de ajuda sem interferir com a funcionalidade e permite ajustar os tipos de dados enumerados (usados para preencher as combo boxes, por exemplo), também sem interferir com a funcionalidade da aplicação.

5.1 Trabalho Futuro

Depois de uma primeira arquitectura para um sistema distribuído de catalogação, como trabalho futuro (já em desenvolvimento, no âmbito do mestrado do Ricardo Martins), está em preparação a criação de uma Infra-estrutura de Dados Espaciais (IDE). Esta infra-estrutura responde não só a identificação de uma IG existente, mas a todas as funcionalidades requeridas por um ambiente comercial e heterogéneo de informação geográfica e dos serviços baseados na localização.

Referências

[Hoh98] Pat Hohl, editor. *GIS Data Conversion: Strategies, Techniques, and Management.* OnWord Press, 1998.

[San01] Maribel Yasmina Campos Alves Santos. *PADRÃO Um Sistema de Descoberta de Conhecimento em Bases de Dados Geo-referenciadas.* PhD thesis, Universidade do Minho, 2001.

Referências na Web

1. Marie-Claire FRANON. Cen/tc 287 geographic information. http://forum.afnor.fr/afnor/WORK/AFNOR/GPN2/Z13C/index.htm. Informação relacionada com os trabalhos do comité europeu encarregado da norma CEN 287.

2. Oracle. Oracle xml developer's kit for java. http://technet.oracle.com/tech/xml/xdk_java/content.html. Oracle XML Developer's Kit (XDK) contains XML component libraries and utilities that you can use to XML-enable applications and Web sites.
3. Bjørnhild Sæterøy. Iso/tc 211 geographic information/geomatics. http://www.isotc211.org/. Toda a informação relacionada com os trabalhos do comité. O acesso a determinados arquivos é reservado.

Partilha de Conteúdos e Semântica Multimédia

José Torres

Inesc-Porto, Campus da FEUP, Rua Dr. Roberto Frias, 378
4200-465 Porto – Portugal
jtorres@inescporto.pt
http://www.inescporto.pt

Resumo. A utilidade dos conteúdos multimédia depende largamente da facilidade com que se pode chegar até eles. Neste artigo apresenta-se um cenário de integração entre esforços de investigação que de um modo simbiótico podem facilitar a partilha de conteúdos multimédia em ambiente Web. As afinidades entre a Web Semântica e a norma ISO/IEC MPEG-7 vão para além da simples similaridade sintáctica. O verdadeiro potencial reside na possibilidade de uma integração plena entre descrições MPEG-7 e Web Semântica de um modo geral, permitindo assim que os futuros sistemas de processamento para a Web (agentes) possam reconhecer e trabalhar com essa informação descritiva de um modo automático.

1 Introdução

Nos últimos anos tem-se assistido a avanços tecnológicos significativos nas áreas dos dispositivos digitais para criação e manipulação de conteúdos multimédia como as maquinas fotográficas digitais ou as câmaras de filmar digitais. A massificação do uso de tecnologias digitais e o seu baixo custo, estão correlacionados, o que se traduz no facto da taxa de criação de conteúdos multimédia actualmente sofrer um crescimento exponencial em termos temporais.

Outra área onde se sente igualmente uma "explosão" tecnológica é a das telecomunicações. Torna-se cada vez mais fácil trocar informação de e para qualquer lado do planeta a débitos cada vez mais elevados. O aparecimento da Internet veio revolucionar modos e hábitos em várias esferas da nossa sociedade como por exemplo a educacional, a comercial, a cultural ou a do entretenimento. No entanto, novos desafios estão continuamente a ser lançados. Se até há pouco tempo, era impraticável usar a Internet para a distribuição de conteúdos multimédia, dado o requisito de elevada largura de banda que estes impunham (particularmente o vídeo), neste momento caminha-se para um cenário em que esse factor tenderá a ser cada vez menos um obstáculo.

No campo da Internet tem-se assistido ao avançar da segunda geração da Web, a Web Semântica, cujo lema é que a World Wide Web atingirá todo o seu pleno potencial quando se tornar num ambiente onde a informação possa ser efectivamente partilhada e processada quer por pessoas directamente quer por ferramentas de um

modo automático [4]. A organização W3C (World Wide Web Consortium) agrega os esforços nesse sentido [12].

No campo dos conteúdos multimédia tem havido um grande esforço de investigação na área da descrição dos mesmos seguindo-se este a uma vaga em que a ênfase foi colocada no desenvolvimento de técnicas de codificação e compressão eficiente dos diversos tipos de informação multimédia. Historicamente, o organismo internacional de normalização ISO/MPEG tem desenvolvido um grande trabalho nessa área [7].

O objectivo desta apresentação é traçar uma rota comum a estas duas áreas de investigação que já começou a ser delineada com a criação de normas como a meta-linguagem XML-Schema do W3C e a norma de descrição de informação multimedia MPEG-7 do ISO/MPEG [5]. Estas normas têm como principal objectivo abrir caminho para mais esforço de investigação e desenvolvimento nas áreas de investigação referidas funcionando como uma alavanca e estabelecendo também plataformas comuns e abertas.

Assim, a estrutura do documento baseia-se na apresentação da Web Semântica, na secção seguinte. Na secção 3, é apresentada com mais detalhe a norma de descrição de conteúdos multimédia MPEG-7. Na secção 4 descreve-se como actualmente os sistemas de processamento multimédia tendem para uma unificação entre a descrição de conteúdo e a descrição mais conceptual ou estrutural. Na secção 5 traça-se uma rota de integração entre a Web (Semântica) e a criação, descrição e consumo de conteúdos multimédia para permitir partilhar quer os conteúdos multimédia quer as suas descrições de conteúdo e de nível mais conceptual. Apresenta-se também um cenário que pretende ilustrar o descrito anteriormente. Termina-se o documento com algumas considerações finais.

2 A Web Semântica

O projecto de uma Web semântica representa uma evolução da Web actual com a intenção de a tornar mais usável quer por pessoas quer por máquinas. Tipicamente, a informação dispersa na Web sofre de vários problemas dos quais dois são apontados a seguir. O primeiro está relacionado com o facto de a linguagem HTML, o formato tradicional usado na Web, promover a promiscuidade nos seus documentos entre a camada de conteúdo ou informação e as camadas de transformação e de apresentação ou formatação. O segundo relaciona-se com a interpretação semântica pois, por exemplo, apesar de ser possível criar hiperligações em HTML, por vezes torna-se difícil para um humano, e ainda mais para uma máquina interpretar o significado de uma ligação entre uma página HTML e outra. Ainda dentro do mesmo contexto, pode-se apontar a linguagem HTML com sendo desprovida de mecanismos sintácticos que permitam expressar um nível semântico mais elevado aos seus documentos.

Quanto à separação em camadas, o primeiro problema apontado à linguagem HTML, a gama de normas à volta da linguagem XML e da Web Semântica oferece soluções bastante mais poderosas e flexíveis. Assim, temos a linguagem XSL – Extensible Stylesheet Language, nomeadamente XSLT (XSL Transformations) uma

das suas partes constituintes que é responsável por assegurar a tarefa de transformação de documentos XML noutros documentos XML (ou um formato intermédio designado por XSL-FO [3]), permitindo efectuar mudanças de conteúdo como adições, remoções ou alterações, convertendo deste modo o conteúdo num formato de dados suportado pela aplicação final [6]. As últimas camadas de formatação de estilo e apresentação, antes da composição final, podem ser asseguradas quer pela XSL quer pelo CSS (Cascading Style Sheets) um mecanismo mais antigo usado também com documentos HTML.

Em virtude do exposto, assume-se que na Web Semântica há claramente uma preocupação em identificar e isolar o que é o conteúdo fonte (textual, gráfico, áudio ou vídeo) a partir do qual se vai compor uma apresentação. Esse mesmo conteúdo irá também estar disponível para ser processado pelos sistemas automáticos.

É unanime considerar o XML como um componente fundamental para suportar a Web Semântica dado o seu potencial sintáctico para suportar a expressão de bases de conhecimento. Com o desenvolvimento da linguagem de ligação de recursos XLink, um documento XML ganha capacidade de expressar não só relacionamentos para qualquer tipo de recurso (URI-Uniform Resource Identifier) como também de incorporar semântica nesses relacionamentos. Os aspectos descritos neste parágrafo representam algumas das soluções para contornar o segundo dos problemas apontados ao HTML.

A Web Semântica permitirá, por exemplo, que qualquer agente produtor de conteúdos possa expor a sua informação de modo que um outro programa não tenha que "adivinhar" através de relacionamentos não explícitos, como acontece na geração HTML, factos que são semanticamente relevantes. Isso poderá ser feito através da geração de ficheiros que, de um modo explicito, clarificam quais os diversos relacionamentos entre os vários "pedaços" de informação, como por exemplo um "link semântico" entre o campo 'autor' de um formulário de uma página web com uma fotografia e o campo 'autor' de uma base de dados de fotografias e suas descrições.

No contexto Web Semântica, as ontologias visam permitir a partilha de um vocabulário comum ao nível da semântica, i.e., quando um interveniente usa uma determinada terminologia sabe que o significado que atribui a esses termos está normativamente aceite e é conhecido por todos os outros intervenientes. Para que tal seja possível, o W3C procura desenvolver ferramentas como a linguagem de descrição de informação semântica OWL (Web Ontology Language) com base no núcleo da linguagem RDF de descrição de recursos.

3 A Descrição de Conteúdos Multimédia em MPEG-7

A norma MPEG-7 ocupa-se da descrição dos conteúdos multimédia, como áudio, vídeo e imagem parada. Conceptualmente, MPEG-7 é uma norma integradora de representação de informação audiovisual, no entanto, a sua estrutura assenta em várias partes (8 partes no total). Ela especifica um conjunto de normas para representação dos descritores (*Descriptors*) que podem ser usados para a descrição de características dos vários tipos de informação multimédia. Existem dois documentos que normalizam

os descritores MPEG-7, um só com descritores para áudio e outro só com descritores visuais como por exemplo os descritores de cor, textura e forma para imagem fixa.

Outra parte importante da norma designada por Esquemas de Descrição Multimedia (MDS - Multimedia Description Schemes) envolve quer os descritores áudio quer os descritores visuais. Esta parte contempla esquemas de descrição mais elaborados que usam como blocos básicos quer os descritores quer tipos básicos definidos na norma quer ainda relacionamentos entre si.

Os descritores e os esquemas de descrição são definidos usando uma linguagem de definição de descrições (DDL-Description Definition Language) que pode ser usada ainda para definir novos esquemas de descrição em extensão aos actuais. A DDL constitui assim mais uma parte da norma MPEG-7. Durante o processo de desenvolvimento da norma, foi decidido adoptar a linguagem XML-Schema do W3C como a linguagem DDL. Esta decisão implicou que os modelos de descritores e esquemas de descrição existentes na norma sejam esquemas expressos em XML-Schema, e que (instâncias de) descrições MPEG-7 sejam documentos XML.

As descrições devem existir associadas aos conteúdos multimédia para permitirem uma mais rápida e eficiente pesquisa de material de interesse para um dado utilizador. Uma premissa importante da norma MPEG-7 foi a de se limitar a normalizar o formato das descrições, i.e., a sua sintaxe, semântica e codificação binária tratada pela parte de sistemas da norma. Significa isto que os métodos de extracção dos descritores não estão incluídos na norma, sendo aquilo que se designa pela parte não normativa da documentação que faz parte do MPEG-7.

4 União entre Informação Conceptual e de Conteúdo

Dos vários formatos de informação existentes como áudio, gráficos, imagem fixa ou em movimento e texto, de um modo geral ao longo deste artigo será dada preferência ao formato imagem, particularmente imagem fixa, visto que está directamente relacionada com o trabalho desenvolvido pelo autor.

Os sistemas de indexação e pesquisa de imagem, sofreram fases distintas desde que começaram a ser implementados em computadores. Nos primeiros sistemas, as descrições eram puramente textuais e a anotação da informação visual era um processo manual. Estes sistemas apresentam como principais desvantagens: primeiro, absorvem uma enorme quantidade de tempo no processo de descrição das imagens; segundo, propiciam a existência de um certo grau de subjectividade nas descrições, i.e., dois anotadores podem criar diferentes anotações para a mesma imagem ou até, em instantes diferentes, o mesmo anotador pode acabar por criar anotações diferentes para a mesma imagem.

Numa segunda fase, surgiram os sistemas de indexação e pesquisa de imagem por conteúdo (CBIR - Content Based Image Retrieval) [2],[8]. Estes sistemas baseiam-se em criar descrições extraídas automaticamente das imagens com base em características que se pretende que sejam discriminatórias do conteúdo. No caso das imagens, tipicamente as principais características ou descritores dividem-se entre as categorias de cor, textura e forma.

Actualmente assiste-se a uma tendência para usar a riqueza conceptual das descrições textuais combinada com as descrições do conteúdo geradas automaticamente [1],[10],[11].

Os esquemas de descrição multimédia tendem a ter uma perspectiva integradora como é o caso do esquema desenvolvido ao longo do projecto Metamedia [9] ou da família de descritores e esquemas de descrição da norma MPEG-7.

A indexação dos conteúdos multimédia no futuro tenderá a congregar várias características de modo a potenciar descrições mais ricas quer dos pontos de vista conceptual, estrutural e de conteúdo. Assim, algumas das tendências que poderemos esperar dos sistemas de indexação de meta-informação multimédia serão:

- A indexação/associação da maior quantidade de informação possível em tempo de criação dos conteúdos multimédia (exemplo: meta-informação sobre uma câmara de vídeo como a abertura da lente, etc.);
- Normalizar a informação estrutural: quando uma colecção de n items multimédia possuir descrições comuns, fazer emergir essas descrições para um nível hierárquico superior, neste caso um nível de colecção, em vez de associar essas descrições a cada um dos n items descritos (ver [9]) ;
- Potenciar a utilização de ferramentas de reconhecimento e classificação automática de informação de mais alto nível (por exemplo, reconhecimento de objectos visuais de interesse numa dada imagem). Estes sistemas tipicamente usam técnicas de aprendizagem para aumentarem o seu conhecimento e capacidade de classificação;
- Incrementar a utilização de ferramentas de inferência de nova informação a partir das descrições ou factos existentes (exemplo: sistemas de inferência usando lógica de primeira ordem);
- Integrar as várias bases de dados de descrições e conteúdo de modo a que sistemas de processamento como os agentes de indexação e pesquisa possam derivar ou inferir mais conhecimento. Este é um ponto onde a Web Semântica e mecanismos de ligação semântica como o XLink poderão e deverão desempenhar um papel fundamental.

5 Descrição e Partilha de conteúdos e semântica multimédia na Web

Uma visão possível da cadeia de produção, armazenamento, distribuição, partilha e pesquisa de conteúdos multimédia usando como meio de suporte a Web Semântica aponta para o ambiente de partilha de semântica multimédia. Esta partilha significa não só que os conteúdos, as imagens o vídeo ou o áudio, estão disponíveis, como também significa que conhecimento sobre estes conteúdos se encontra associado. As implicações disto podem ser profundas pois o autor está convencido que neste caso o todo é muito maior do que a soma das partes. Estamos a falar de conhecimento, e como o conhecimento também é um conjunto de relacionamentos estabelecidos entre factos, se programas autónomos tiverem acesso aos factos poderão eles próprios derivar novo conhecimento.

Assim, prevê-se que haja uma integração das técnicas maioritariamente automáticas de extracção/indexação de características discriminatórias dos conteúdos multimédia, que serão tipicamente feitas na altura da produção dos conteúdos multimédia (por exemplo: quando alguém está a tirar uma fotografia, a localização da máquina poderá ser associada à imagem tirada) com as técnicas de processamento de informação factual para, por exemplo, derivar relacionamentos não explícitos ou técnicas de aprendizagem usadas por exemplo para operações de reconhecimento ou classificação de conteúdos multimédia. Percebe-se assim que as normas referidas anteriormente funcionam mais como elos de ligação entre a produção de informação e a utilização ou processamento da mesma.

A título de exemplo, suponhamos que temos dois sites distintos com a seguinte informação:

- Site 1: Temos uma imagem (foto_1) que está relacionada por uma ligação semântica do tipo *temLocalização* a um conjunto de coordenadas geográficas (coords_1) captadas pela respectiva máquina fotográfica digital usando um sistema de GPS. Adicionalmente, um sistema de segmentação identificou um objecto visual (obj_1) na imagem com *boundingBox* (box_1);
- Site 2: Temos uma imagem (foto_2) onde está identificado/segmentado um objecto visual (obj_2) com *boundingBox* (box_2). Essa imagem possui uma ligação semântica do tipo *temLocalização* a um conjunto de coordenadas geográficas (coords_2). O objecto segmentado possui as ligações ou propriedades *edifícioTipo* = ”Torre” e *edifícioNome* = ”Clérigos”

Suponhamos que o agente que está a processar informação multimédia tem acesso à informação desses dois sites e possui ainda as seguintes regras no seu sistema de regras:

- Regra 1: Para um objecto com (*boundingBox* = b) ser classificado como (*edifícioTipo* = ”Torre”) tem de respeitar a condição Altura(b) > Largura(b);
- Regra 2: Para dois objectos visuais (obj_1 e obj_2) poderem representar a mesma entidade (no caso descrito o mesmo edifício), as suas localizações não podem diferir mais do que uma dada distância *Distância*(*Localização*(obj_1), *Localização*(obj_2)) < *limite*;

Além disso o agente possui a capacidade de aplicar um algoritmo de classificação ao objecto visual identificado na foto_1 usando quer características extraídas directamente das imagens como cor, textura e forma quer as regras atrás descritas. Desse modo o agente poderia inferir com algum grau de confiança que o obj_1 representa a mesma entidade que o obj_2, i.e., a Torre dos Clérigos no Porto.

6 Conclusão

Tal como o cérebro humano é composto por memória visual ou auditiva para além da memória de nível mais conceptual, também a Web Semântica poderá, de alguma maneira, aproximar-se da nossa memória.

Em jeito de síntese, o dar possibilidade aos sistemas automáticos que processam a informação de poderem «entender» a informação que estão a manipular, constitui um objectivo comum, quer na comunidade de investigadores ligada à Internet quer na ligada ao processamento de informação multimédia. Este objectivo tem dado origem a avanços em ambos os campos de investigação referidos, caminhando-se para um cenário em que o exemplo descrito na secção anterior se poderá aplicar a uma escala maior. Este irá constituir sem duvida um dos maiores desafios que a Web da segunda geração irá enfrentar.

Referências

1. Benitez A. B., Chang S.-F.: Multimedia Knowledge Integration, Summarization and Evaluation. International Workshop On Multimedia Data Mining (MDM/KDD-2002), Canada, July (2002)
2. Bimbo A. del: Visual Information Retrieval. Morgan Kaufmann Publishers, Inc. San Francisco, California (1999)
3. Deach, Stephen: What Is XSL-FO and When Should I Use It?. The Seybold Report - Analyzing Publishing Technologies, Vol. 2, Num. 17, December 9, (2002), ISSN: 1533-9211 (url: http://www.seyboldreports.com/TSR/free/0217/techwatch.html)
4. Lee, Tim Berners and Miller, Eric: The Semantic Web lifts off. ERCIM News, Num. 51, October (2002) 9-11 (url: http://www.ercim.org/publication/Ercim_News/enw51/)
5. Manjunath B. S., Salembier P., Sikora T. (eds.): Introduction to MPEG-7, Multimedia Content Description Interface. John Wiley & Sons Ltd (2002)
6. Jung, Benjamin and Nixon, Lyndon JB: Integrating Multimedia Components into a Semantic Web. ERCIM News, Num. 51, October (2002) 38-39 (url: http://www.ercim.org/publication/Ercim_News/enw51/)
7. ISO/MPEG - Moving Picture Experts Group (url: http://www.cselt.it/mpeg/standards.htm)
8. Niblack, W., R. Barber, W. Equitz, M. Flickner, E. Glassman, D. Petkovic and P. Yanker: The QBIC Project: Querying images by content using colour, texture and shape, in SPIE 1908, Storage and Retrieval for Image and Video Databases, February (1993)
9. Ribeiro, Maria Cristina and David, Gabriel: A Metadata Model for Multimedia Databases. in David Bearman, Franca Garzotto, Proceedings of ICHIM01: International Cultural Heritage Informatics Meeting- Cultural Heritage and Technologies in the Third Millennium, pp.469-482, (2001)
10. Tansley R.: The Multimedia Thesaurus: Adding a Semantic Layer to Multimedia Information. Ph.D. Thesis, University of Southampton, Southampton-UK, August (2000)
11. Torres, José and Parkes, Alan: A Modular Framework for Visual Object Information Retrieval. 4th European Workshop on Image Analysis for Multimedia Interactive Services, U.K., April (2003)
12. The W3C Semantic Web (url: http://www.w3c.org/2001/sw)

Desafios e contribuições resultantes da utilização da XML na norma ISO/MPEG-7 e suas aplicações

Rui J. Lopes, Adam T. Lindsay, David Hutchison
Comp.Dep.-Lancaster University
lopes@comp.lancs.ac.uk

Palavra-Chave: MPEG-7

Resumo

A meta-linguagem XML e as tecnologias a ela associadas tem sido utilizadas de forma crescente quer na especificação das normas ISO/MPEG quer nas suas aplicações.

Entre os domínios de aplicação da meta-linguagem XML nas normas ISO/MPEG tem particular importância a sua utização para definição da estrutura e formato de representação:

- textual da estrutura do conteúdo audio/visual (formato XMT na norma ISO/MPEG-4),
- da descrição de conteúdos multimedia (norma ISO/MPEG-7, *Multimedia Content Description Interface*), e
- da descrição de recursos e sua utilização em plataformas abertas para a produção, transferência, processamento e consumo de conteúdos multimedia (norma ISO/MPEG-21, *Mutilmedia Framework*).

Embora o formato de representação seja a utilização mais comum da XML nas normas ISO/MPEG exitem outros aspectos relacionados com a natureza dos documentos/objectos multimédia que requerem outras funcionadidades que constituem desafios à utilização da XML nestas normas e suas aplicações. Esta comunicação aborda estes assuntos no contexto da norma ISO/MPEG-7.

Segundo a norma ISO/MPEG-7 a metada associada a um conteúdo/essência audio/visual tem como realização um documento XML cujos elementos descrevem as suas características/propriedades. A norma ISO/MPEG-7 especifica de forma normativa, através de esquemas (schema), a estrutura e sintaxe desses documentos XML (partes 3, 4 e 5 da norma). A norma ISO/MPEG-7 define também de forma normativa uma linguagem para a definição de novos esquemas de descricão (DDL - *Description Definition Language*, parte 5 da norma). Esta linguagem é baseada na XML Schema introduzindo-lhe algumas modificações relacionadas com o domínio do audio/visual.

Eventualmente o maior conjunto de desafios e contribuições resultantes da utilização da XML na norma ISO/MPEG-7 encontra-se no conjunto de funcionalidades definidas na sua parte sistema.

Esta parte da norma define o comportamento normativo de um descodificador que consume um fluxo de descrições (description stream) e constroi progressivamente um documento XML correspondente ao estado actual da descrição do conteúdo audio/visual.

Um fluxo de descrições é composto por um conjunto de unidades básicas que transportam sequências de comandos para a actualização da descrição construída pelo descodificador. A sintaxe destas unidades básicas é definida também por esquema próprio.

Este aspecto dinâmico da descrição construída no descodificador resulta como requisito da natureza intrinsecamemte dinâmica da generalidade do contéudo audio/visual. A introdução de um caracter dinâmico em documentos XML e a sua transformação num fluxo constituem os maiores desafios colocados pela utilização da XML para a descrição de conteúdo audio/visual.

Embora o comportamento do descodificador e a sintaxe para os fluxos de descrição sejam especificados de forma inambigua pela norma a forma como os fluxos de descrições são criados, i.e., o comportamento do codificador é deixado livre à criatividade e inovação.

A participação da Universidade de Lancaster na definição da norma MPEG-7 centra-se em particular na parte sistemas, i.e., na especificação do formato, transporte e consumo de fluxos de descrições.

Neste contexto foram desenvolvido dois protótipos:

- um codificador de parte sistema que gera fluxos de descrições compostos por multiplas unidades elementares, e
- um terminal para o consumo de fluxos de descrições segundo o formato textual definido pela norma.

O conjunto destes dois protótipos tem sido utilizado para diversas contribuicões para o forum ISO/MPEG, nomeadamente o estabelecimento de fluxos de referência para testes de compatibilidade, e para o desenvolvimento de testes a novas funcionadidades para a norma (*core experiments*).

Estes protótipos têm sido utilizados também em aspectos não normativos relacionados com a utilizaçao da norma, nomeadamente:

- cenários envolvendo a sincronização entre a metada e o conteúdo, interactividade sobre diferentes meios de distribuição (arquivos, Internet, difusão),
- aplicação de metadata em que a pesquisa e/ou a selecção/filtragem de conteúdo audio/visual (ou parte deste) é de importância capital, e
- o transporte de fluxos de descrições utilizando redes em que existe a possibilidade de processamento desses fluxos nos nós da rede.

Entre os aspectos da utilização da XML actualmente em estudo no forum ISO/MPEG possuem particular importancia a utilização de multiplos fluxos de descrições,

a definição de codificações binárias eficientes para os fluxos de descrição e o seu transporte/processamento em diferentes meios.

Titulo: Omnipaper- Descrição de recursos de noticias digitais em RDF

Autores

Teresa Susana Mendes Pereira e Ana Alice Baptista

{tpereira, analice }@dsi.uminho.pt

Resumo

Este artigo pretende introduzir a tecnologia *Resource Description Framework* - RDF, desenvolvida pela W3C para a web baseada em metadados, e a sua utilização prática no projecto Omnipaper, que as autoras estão envolvidas.

O projecto Omnipaper (*Smart Access to European Newspapers*) é um projecto do programa IST (*Information Society Technologies*) da comissão europeia, que visa promover o acesso distribuído a diferentes tipos de fontes de informação.

Com este projecto, pretende-se chegar a um protótipo de um sistema que permita aos utilizadores (quer ocasionais, quer profissionais) um acesso estruturado, personalizado e multilíngua a todo um conjunto de artigos de notícias disponibilizados pelas empresas parceiras (a que chamamos arquivos locais).

Neste artigo será descrito a forma como é usada a tecnologia RDF na descrição dos diferentes tipos de artigos de noticias, seguida de uma descrição elaborada sobre o processo de manipulação e tratamento da informação estruturada em RDF, através da ferramenta *RDF Gateway*, ainda em fase de desenvolvimento.

Keywords: Metadados, RDF – *Resource Description Framework*, *Dublin Core*, Triplos, Recuperação de Informação, Web, Recursos.

1. Introdução

Nas últimas décadas o crescimento da informação digital foi exponencial. O mesmo se verifica no crescimento da Internet. A informação está cada vez mais disponível em formato electrónico e a sua acessibilidade na Internet tem vindo a aumentar rapidamente. Este crescimento e disponibilidade contribuí para a necessidade de agrupar a informação, uma vez que tanto o seu acesso como a comparação com outras fontes que se encontram geograficamente dispersas é fisicamente suportado pela Internet, contribuindo para a necessidade de integrar a informação a nível semântico.
Um dos importantes desafios dos utilizadores da Web, na pesquisa e navegação na rede reside na necessidade de organizar o imensurável número de páginas Web que surgem todos os dias a todas as horas na Internet.
É neste contexto, que surge o projecto Omnipaper, que pretende investigar formas de promover o acesso distribuído a diferentes tipos de fontes de informação. Permitindo aos utilizadores um acesso estruturado, personalizado e multilíngua a todo o conjunto de artigos de noticias disponibilizados pelas empresas parceiras (a que denominamos por arquivos locais). O Omnipaper não é um projecto de digitalização de noticias mas sim portador de noticias digitais provenientes de diversos locais e de diversos formatos.
O projecto prevê a realização em paralelo de dois protótipos seguindo duas abordagens distintas aos metadados, tanto no Local como no *Overall Knowlege Layer*: a abordagem *Resource Description Framework* (RDF) e a abordagem *Topic Maps* (TM), que contribuem significativamente, para o desenvolvimento da tecnologia na Web e na publicação electrónica, nomeadamente no desenvolvimento de Bibliotecas Digitais.

Pretende-se com este artigo explicar, todo o trabalho desenvolvido na definição da estrutura dos metadados utilizados na descrição dos diferentes géneros de artigos que fazem parte do protótipo, assim como a abordagem RDF utilizada na sua definição.

A estrutura do artigo será composta pelas seguintes secções: na Secção 2 será realizada uma abordagem ao Modelo RDF e à sua Sintaxe, na Secção 3 apresentaremos o Protótipo da descrição dos documentos em RDF/XML

acompanhado de um exemplo de uma descrição do tipo RDF/XML, e das regras de descrição utilizadas, na Secção 4, será apresentado o processo de gestão e tratamento das descrições, utilizando um sistema de gestão de bases de dados nativas RDF - o RDF *Gateway*, por fim apresentaremos as conclusões e trabalho futuro na Secção 5.

2. *O Resource Description Framework (RDF)*

O *Resource Description Framework* [1] contém, antes de tudo, um modelo para expressar semântica.

Uma asserção RDF faz declarações sobre recursos, usando uma propriedade e tendo como resultado da aplicação dessa propriedade ao recurso, um valor. Uma asserção pode ser vista como um triplo composto por três elementos: propriedade (predicado), recurso (sujeito) e valor (objecto). Um recurso pode ser qualquer coisa identificável por um URI.

O modelo RDF é simplesmente um modelo de triplos, o que o torna muito poderoso, mas difícil de implementar. Por definição, a descrição usando os triplos, usando o grafo ou usando a sintaxe RDF/XML é equivalente. O *parser* RDF/XML é responsável por ler, verificar a sintaxe RDF/XML, e transformar o código escrito na sintaxe RDF/XML num conjunto de triplos e, eventualmente, num grafo RDF.

O RDF está dividido em duas partes, contendo duas especificações distintas :

(1) A *RDF Model and Syntax Specification* (RDFMSS) [2] que é uma recomendação do W3C [3] que contém um modelo para representar metadados RDF, bem como uma sintaxe para codificar e transportar metadados de forma a maximizar a interoperabilidade de servidores e clientes Web desenvolvidos independentemente;

(2) A *RDF Schema Specification* [4] é uma especificação de esquemas. Com o Esquema RDF podem-se desenhar e implementar de uma forma consistente, vocabulários de metadados específicos. Estes podem ainda ser desenvolvidos no seio de outros projectos gerando, assim uma rede de esquemas de metadados. Por exemplo, determinados termos de um vocabulário a ser desenhado podem perfeitamente ser definidos como refinamentos de elementos de DC ou de outro qualquer vocabulário anteriormente definido. Na definição de metadados a utilizar na descrição dos nossos documentos foram utilizados como referência outros

vocabulários, como por exemplo o NITF (*News Industry Text Format*) [5] e NewsML (*News Agency Implementation Guidelines*) [6], para além dos elementos do DC [7]. Neste momento o *Dublin Core* é já utilizado em diversos locais do mundo e é o principal alicerce na construção e desenvolvimento do RDF.

3. Protótipo da descrição dos documentos em RDF/XML

A codificação RDF/XML não é simples. Se o modelo RDF é em si mesmo de uma simplicidade extraordinária e se o rigor que permite conferir às descrições é de uma lógica matemática pura, o mesmo já não se passa com a sintaxe RDF/XML para sua implementação. De facto, num curto espaço de três anos, já vários documentos que explicam ou tentam normalizar a descrição de metadados em RDF/XML foram criados e tornados obsoletos. A DCMI já lançou vários *drafts* sobre o assunto, mas até à data em que escrevemos este artigo ainda não há nenhuma recomendação aceite pelo DCUB. O próprio grupo de trabalho *RDF CORE* [8] do W3C está a refazer uma nova especificação para o RDF com dois objectivos fundamentais: (1) tornar a especificação mais fácil de ler e entender e (2) eliminar alguns erros assumidos da sintaxe RDF/XML.
Neste sentido, nas nossas descrições é usado tanto quanto possível as recomendações feitas no documento *Expressing Qualified Dublin Core in* RDF/XML [9], apesar de esta ainda ser uma recomendação candidata.

De seguida apresentaremos algumas das regras aplicadas à descrição dos documentos em RDF/XML, que integram o conjunto de documentos do protótipo.

(1) Recursos como valor de uma propriedade
Existem duas alternativas principais para codificar um recurso como valor de uma propriedade: ou como conteúdo do elemento XML respeitante à propriedade, ou como conteúdo do atributo *rdf:Resource* associado ao mesmo elemento XML.
Por uma questão de simplicidade e clareza da descrição, codificamos os recursos que são valores de propriedades no atributo *rdf:Resource* do elemento XML correspondente à propriedade em causa.
(2) Codificação de nodos anónimos

Existem propriedades que estão relacionadas com uma outra, o que implica a utilização de nodos anónimos. Por exemplo se pretendermos descrever o autor do artigo, não apenas utilizando o seu nome, mas também outros dados, vamos ter que utilizar também outras propriedades que guardam o valor relativo a cada um desses tipos de dados. Assim, utilizamos primeiro a propriedade *dc:creator* que aponta para um nodo anónimo de onde vão sair novos arcos (que correspondem a propriedades) para nodos que já possuem valores. No exemplo apresentado em anexo, temos as propriedades *vCard:n*, *vCard:family*, *vCard:EMAIL* e *vCard:ORG*, que contribuem significativamente para uma melhor caracterização dos agentes, neste caso particular do autor do artigo.

A utilização do valor "*Resource"* no atributo *rdf:parseType*, indica-nos, que o conteúdo do elemento tem que ser tratado como se fosse o conteúdo de um elemento *Description*, contribuindo desta forma para uma melhor clareza e simplicidade do documento.

(3) Utilização de *Bags*

Na especificação do modelo e sintaxe do RDF, um *Bag* é definido como uma lista não ordenada de recursos ou literais. *Kokkelink e Schwänzl* preferem defini-lo como uma lista cuja ordem não tem significado.

No exemplo apresentado, o elemento *subject* é codificado como um *Bag*. Foi tomada está opção, na medida em que parece evidente a utilização deste *container* uma vez que um artigo pode conter vários assuntos.

De seguida apresentamos um exemplo de codificação em RDF de um artigo, seguido da tabela de triplos e do grafo do modelo de dados correspondente, permitindo alcançar uma noção da estrutura do RDF.

3.1 Documento RDF/XML

```
<rdf:RDFxmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:omn="http://xxx/2002/06/01/schema#"
xmlns:vCard="http://wwww.w3.org/2001/vcard-rdf/3.0#">
<rdf:Description rdf:about="http://www.guardian.co.uk/child/story/0,7369,777571,00.html">
 <!--TITLE -->
   <dc:title>Police face tight deadline over charges on Soham murders</dc:title>
   <dcterms:alternative>Soham police face tight deadline</dcterms:alternative>
 <!-- CREATOR -->
<dc:creator rdf:parseType="Resource">
   <vCard:n>Sarah</vCard:n>
   <vCard:family>Hall</vCard:family>
   <vCard:EMAIL /> sarah.hall@guardian.co.uk
   <vCard:ORG>Guardian Unlimited</vCard:ORG>
  </dc:creator>
 <!--DESCRIPTION -->
<dc:description>Detectives investigating the murders of Holly Wells and Jessica Chapman today face a rapidly approaching
deadline as they decide whether to bring charges against a school caretaker and his girlfriend who are suspected of killing
the girls. Cambridgeshire police have until around 6am tomorrow to interview Ian Huntley and around 4am to interview
Maxine Carr. By then, they must either secure enough evidence to charge them with murder and abduction or release
them without charge.
</dc:description>
<!--SUBJECT -->
<dc:subject>
            <rdf:Bag>
              <rdf:li>World News</rdf:li>
              <rdf:li>Internacional</rdf:li>
            </rdf:Bag>
</dc:subject>
<!-- PUBLISHER -->
<dc:publisher rdf:parseType="Resource">
   <vCard:ORG>Guardian Unlimited</vCard:ORG>
   <vCard:URL rdf:resource="http://www.guardian.co.uk" />
</dc:publisher>
<!-- DATE OF CREATION OF THE ARTICLE -->
<dcterms:created>
            <dcterms:W3CDTF>
                  <rdf:value>2002-08-20</rdf:value>
            </dcterms:W3CDTF>
</dcterms:created>
<!-- LANGUAGE -->
<dc:language>english</dc:language>
<!-- FORMAT -->
<dcterms:extent>120 KB</dcterms:extent>
<dcterms:medium>
            <dcterms:IMT>
              <rdf:value>text/html</rdf:value>
            </dcterms:IMT>
 </dcterms:medium>
 </rdf:Description>
</rdf:RDF>
```

3.2 Triplos do Modelo de Dados

Number	Subject	Predicate	Object
1	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/elements/1.1/title	Police face tight deadline over charges on Soham murders
2	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/terms/alternative	Soham police face tight deadline
3	genid:ARP7374	http://wwww.w3.org/2001/vcard-rdf/3.0#n	Sarah
4	genid:ARP7374	http://wwww.w3.org/2001/vcard-rdf/3.0#family	Hall
5	genid:ARP7374	http://wwww.w3.org/2001/vcard-rdf/3.0#EMAIL	
6	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/elements/1.1/description	Detectives investigating the murders of Holly Wells and Jessica Chapman today face a rapidly approaching deadline as they decide whether to bring charges against a school caretaker and his girlfriend who are suspected of killing the girls. Cambridgeshire police have until around 6am tomorrow to interview Ian Huntley and around 4am to interview Maxine Carr. By then, they must either secure enough evidence to charge them with murder and abduction or release them without charge.
7	genid:ARP7375	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag
8	genid:ARP7375	http://www.w3.org/1999/02/22-rdf-syntax-ns#_1	World News
9	genid:ARP7375	http://www.w3.org/1999/02/22-rdf-syntax-ns#_2	Internacional
10	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/elements/1.1/subject	genid:ARP7375
11	genid:ARP7376	http://wwww.w3.org/2001/vcard-rdf/3.0#ORG	Guardian Unlimited
12	genid:ARP7376	http://wwww.w3.org/2001/vcard-rdf/3.0#URL	http://www.guardian.co.uk
13	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/elements/1.1/publisher	genid:ARP7376
14	genid:ARP7378	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/dc/terms/W3CDTF
15	genid:ARP7378	http://www.w3.org/1999/02/22-rdf-syntax-ns#value	2002-08-20
16	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/terms/created	genid:ARP7378
17	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/elements/1.1/language	english
18	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/terms/extent	120 KB
19	genid:ARP7379	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/dc/terms/IMT
20	genid:ARP7379	http://www.w3.org/1999/02/22-rdf-syntax-ns#value	text/html
21	http://www.guardian.co.uk/child/story/0,7369,777571,00.htmll	http://purl.org/dc/terms/medium	genid:ARP7379

3.3 Grafo do Modelo de Dados

4. Sistema de gestão de bases de dados nativas RDF

O protótipo para manipulação da camada RDF do *Local Knowledge Layer* foi realizado, utilizando o *RDF Gateway* [10]. No entanto, antes de começarmos a utilizar esta ferramenta, o nosso primeiro contacto foi realizado com a ferramenta *Tamino XML Server*, mas chegamos à conclusão que não estava preparada para trabalhar no RDF. Um dos casos particulares da sua inviabilidade residia no facto de não aceitar mais do que um *namespace*.

O *RDF Gateway* é uma ferramenta que conjuga os poderes do servidor HTTP com o sistema de Gestão de bases de dados nativas RDF. O conteúdo do *RDF Gateway* pode ser acedido através de um Web browser especificando o URL da aplicação que faz parte do conteúdo de uma *package* definida no *RDF Gateway*.

As aplicações são desenvolvidas numa linguagem *script* denominada *RDF Server Pages (RSP)* semelhantes às *ASP (Active Server Pages)* e as *queries* são implementadas utilizando o *RDF Query Language (RDFQL)*. No entanto, temos encontrado várias dificuldades na utilização desta ferramenta, na medida em que se trata de uma versão beta. Deste modo, estamos neste momento a analisar a viabilidade da ferramenta TAP como alternativa ao *RDF Gateway*.

5. Conclusões e trabalho futuro

O protótipo proposto vai contribuir significativamente, no acesso a fontes (jornais) distintas no domínio cultural e cientifico. Permitindo o acesso personalizado a conteúdos específicos, combinando um elevado número de jornais digitais europeus num sistema.

Numa primeira fase do protótipo, procedeu-se à descrição dos artigos de noticias digitais seguida da implementação no *RDF Gateway* dos processos necessários à sua manipulação. Na próxima fase proceder-se-á a toda a implementação dos protótipos segundo as abordagens estabelecidas: *Resource Description Framework (RDF)* e

Topic Maps(TM) e a realização de testes, comparando a eficiência da utilização de metadados na pesquisa, em ambas as abordagens, segundo critérios de comparação definidos.

Bibliografia

1. W3C, *RDF*, http://www.w3c.org/rdf, 2003.
2. W3C, *RDF Sintax and Grammar*, http://www.w3c.org/TR/rdf-syntx-grammar/, 2003.
3. W3C, *World Wide Web Consortium*, http://www.w3c.org, 2003.
4. W3C, *RDF Schema*, http://www.w3c.org/TR/rdf-schema, 2003.
5. IPTC, *NITF - News Industry Text Format*, http://www.nitf.org, 2002.
6. IPTC, *NewsML - News Agency Implementation Guidelines,* http://www.newsml.org 2002.
7. DCMI, *Dublin Core Metadata Initiative*, http://www.dublincore.org, 2003.
8. W3C, *RDFCore Working Group*, http://www.w3.org/2001/sw/RDFCore/, 2003.
9. Kokkelink, S.S., Roland, *Expressing Qualified Duclin Core in RDF/XML.* 2002.
10. Chappell, G.R., Derrish; Michal , Aaron, *Intellidimension Gateway*,