XATA 2004

XML, Aplicações e Tecnologias Associadas

1ª Conferência Nacional

Editores:

José Carlos Ramalho

Alberto Simões

12-13 Fevereiro, 2004

Faculdade de Engenharia da Universidade do Porto

Ficha Técnica:

Design Gráfico: Benedita Contente Rangel
Redacção: José Carlos Ramalho
Composição: Alberto Simões
Impressão e Acabamento: Reprografia da UM
Tiragem: 100 exemplares

Prefácio: "XML everywhere!"

A História repete-se ciclicamente. Este é um facto bem conhecido e que tem motivado muitos investigadores a estudar o nosso passado. A informática é uma área que não é excepção e, de vez em quando, surge uma nova tecnologia que consegue impôr-se e atravessar transversalmente todos os domínios de aplicação da Informática. O `XML` é a última tecnologia a conseguir este feito. Neste momento, invadiu todos os domínios da Informática levando-nos rapidamente para uma situação em que o desconhecimento da tecnologia é sinónimo de analfabetismo. Hoje em dia, consegue assistir-se, muitas vezes, a situações deveras caricatas do tipo "a minha aplicação até gera `XML` se o utilizador o desejar"e quando se olha bem de perto para a dita aplicação ou produto fica-se sem perceber a que propósito o `XML` ali entrou. Provavelmente, para atrair consumidores menos alfabetizados no tema, mas que sem saberem bem porquê sentem que devem seguir esta nova onda.

Neste momento, os domínios de intervenção do `XML` são inúmeros e todos os dias surgem novas aplicações. A título de exemplo podemos enumerar as seguintes:

Publicação electrónica - foi a primeira área de intervenção e continua a ser uma das principais; quem trabalha nesta área já ouviu, de certeza, falar de duas aplicações `XML` que são o `DocBook` (utilizado na edição de livros técnicos) e o `TEI` (utilizado em documentos nas áreas de ciências humanas, teatro, discurso, ...).

Música - a representação da música num formato textual descritivo sempre levantou muitos problemas e motivou inúmeros curiosos; para este tipo de aplicação existe o `MusicML`.

Matemática - a matemática representou sempre uma dor de cabeça para os processadores de texto; actualmente há apenas um sistema robusto para tratar a edição e formatação da matemática, o TeX de Donald Knuth, e foi baseada neste sistema que foi desenvolvida a aplicação `XML` de nome `MathML`.

Intercâmbio de informação entre aplicações - muitas vezes é necessário transferir informação entre várias aplicações informáticas; até hoje, nunca houve consenso no formato a utilizar para a realizar essa transferência; hoje, o `XML` tem assumido esse papel e é o formato aconselhado por algumas das maiores organizações de software mundiais para a transferência e intercâmbio de informação.

Outras - existem muitas outras aplicações do `XML` como a representação de estruturas moleculares em química, a representação do genoma em biologia, a representação de jogadas de xadrez, representação da informação em sistemas de informação geográficos, representação de informação em todo o tipo de indústria, ...

Pode então surgir a seguinte questão: "Mas donde vem esta euforia em torno do XML? Porque é que lhe dão tanta importância? Afinal, o que é que vem a ser o XML?".

Um pouco de história

A ideia de que os documentos estruturados poderiam ser trocados e manipulados se fossem editados num formato normalizado e aberto data dos anos 60, altura em que foram iniciados esforços para que isso se tornasse uma realidade.

De um lado, a associação norte-americana Graphic Communications Association (GCA) criou uma linguagem designada por GenCode para desenvolver anotações para os documentos dos seus clientes, que utilizavam diferentes aplicações e geravam diferentes formatos. O GenCode permitiu à GCA integrar, no mesmo conjunto, os vários documentos provenientes desses clientes.

Num esforço paralelo, a IBM desenvolveu outra linguagem, designada por Generalized Markup Language (GML), na tentativa de resolver os seus problemas internos de publicação electrónica. O GML foi desenhado de modo a permitir que o mesmo documento pudesse ser processado para produzir um livro, um relatório ou uma edição electrónica.

Como os tipos de documento começaram a proliferar, tendo cada um requisitos diferentes e exigindo, por isso, um conjunto de anotações diferente, a necessidade de publicar e de manipular, de uma forma normalizada, cada tipo de documento também foi crescendo. No princípio dos anos 80, os representantes do GenCode e do GML juntaram-se, formando um comité do *American National Standards Institute* (ANSI) designado por *Computer Languages for the Processing of Text*. O seu objectivo era normalizar a metodologia de especificação, a definição e a utilização de anotações em documentos, o qual foi atingido com a criação do SGML.

A metalinguagem SGML, Standardized Generalized Markup Language, foi lançada publicamente em 1986 como a norma ISO 8879. É uma linguagem desenhada com o objectivo de permitir a definição e a utilização de formatos de documentos. É suficientemente formal para permitir validar os documentos, tem estrutura suficiente para permitir a especificação e o manuseamento de documentos complexos e é extensível de modo a suportar a gestão de grandes repositórios de informação.

No fim da década de 80, o SGML tinha já penetrado nalgumas instituições de grande dimensão, como a indústria aeronáutica e a de maquinaria pesada. No entanto, foi no laboratório suíço do CERN, que um investigador, então a desenvolver a sua aplicação de hipertexto, lhe achou graça e pensou incluí-la na sua aplicação. Com efeito, Tim Berners-Lee, pai do World Wide Web (WWW), escolheu um conjunto de anotações retirado do SGML, e adoptou essas anotações como a linguagem da sua aplicação. Em Nexus, o editor e navegador original do WWW, ele usou essas anotações, folhas de estilo para formatar visualmente as páginas e aquilo que lançou definitivamente este género de aplicações: links.

Em 1993, altura em que apareceu o Mosaic (primeiro browser a ter uma grande divulgação e utilização), os utilizadores estavam já a estender ao máximo o HTML, puxando pelos seus limites. Mesmo a última versão do HTML, a 4.0, lançada em Julho de 1997, fornece apenas um conjunto limitado de anotações. E, se pensarmos um pouco, depressa chegaremos à conclusão de que nenhum conjunto fixo de anotações será suficiente para anotar devidamente todos os documentos que circulam na Web.

Desde 1992, o HTML evoluiu de uma sintaxe adhoc para uma linguagem de anotação definida em SGML. A ideia era fornecer um suporte formal à linguagem e permitir que as ferramentas da Web passassem a implementar o HTML como um caso específico do SGML com uma formatação por omissão, i.e., as ferramentas deixariam de ser específicas do HTML e passariam a ser mais genéricas. Desta maneira, qualquer alteração à sintaxe do HTML seria automaticamente propagada a todas as ferramentas, bastando para isso alterar a especificação do HTML e dos estilos. Esta era uma ideia avançada para a época, os seus custos eram grandes e, nessa altura, a Web não estava preparada para uma linguagem de anotação genérica mas sim para um pequeno conjunto de anotações que qualquer pessoa pudesse aprender em pouco tempo.

A definição do HTML em SGML foi o primeiro passo para aproximar o SGML da Web e, desta forma, cativar o interesse da indústria de software.

Estavam, nesse momento, presentes os ingredientes que viabilizariam o aparecimento do XML, eXtensible Markup Language: por um lado o reconhecimento do SGML como uma boa metodologia para estruturar e representar documentos, por outro, a identificação de limitações do conjunto fixo de anotações do HTML.

A aposta em XML veio possibilitar três funcionalidades críticas:

Extensibilidade - o autor pode definir novas anotações, relacioná-las com as existentes na estrutura e acrescentar-lhe os atributos que desejar.

Estrutura - o autor pode definir uma estrutura para uma dada família ou classe de documentos que pode mais tarde ser associada às instâncias documentais.

Validação - o autor pode proceder, ou solicitar, a validação do conteúdo do documento relativamente à estrutura.

Mas afinal, o que é o XML?

Hoje assiste-se a uma grande proliferação de formatos e de suportes de informação. Os documentos electrónicos podem conter imagens, música, vídeo e texto. No meio deste naipe de suportes de informação o texto ainda continua a ser a plataforma integradora. Mas esta plataforma deve ser normalizada para que facilidades como a partilha, a modularidade e a reutilização possam ser alcançadas. O XML fornece esta plataforma integradora.

XML é a abreviatura de *Extensible Markup Language*. Esta linguagem não é pertença de nenhuma empresa. Foi desenvolvida por um grupo de pessoas independentes, sob a égide do W3C (World Wide Web Consortium), de acordo com experiências anteriores como o SGML ou o HTML.

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, ...

O XML representa o formato ideal para a representação de informação estruturada (por exemplo, a que existe numa base de dados) ou semi-estruturada (por exemplo, o texto de um livro). Um documento XML para além de texto contém umas marcas especiais, chamadas anotações ou etiquetas, que normalmente identificam componentes estruturais do documento. O fragmento abaixo foi retirado dum poema de Álvaro de Campos anotado por mim:

```
<poema>
...
  <terceto>
    <verso>a notícia a essa estranha <nome>Cecily</nome</verso>
    <verso>que acreditava que eu seria grande...</verso>
    <verso>Raios partam a vida e quem lá ande!</verso>
  </terceto>
...
</poema>
```

Como se pode ver, o texto contém algumas anotações: poema, terceto, verso e nome. Estas marcas identificam as componentes estruturais do documento e facilitam o trabalho do computador na hora de processar a informação.

Para além da publicação electrónica, a informação que é transferida ou transaccionada entre programas ou sistemas de computação está a tornar-se mais rica e mais complexa à medida que novas aplicações distribuídas, assentes sobre serviços de rede, são desenvolvidas. A informação que é transaccionada tem que ser autodescritiva para que a aplicação cliente possa receber, interpretar e processar a informação de acordo com as preferências o utilizador sem necessitar de voltar a contactar o servidor que lhe passou a informação. Apesar das raízes históricas do XML estarem ligadas à publicação electrónica, o XML tem vindo a ser utilizado, cada vez mais, para representar estruturas de dados complexas que, provavelmente, nunca serão publicadas ou visualizadas num browser da Internet. Por exemplo, transacções monetárias:

```
<transacção id="T123R">
  <data valor="20021115"/>
  <montante>124.87</montante>
  <moeda tipo="US-dollar"/>
  <de id="jr670201">José Carlos Ramalho</de>
  <para id="jr410602">José dos Santos Ramalho</para>
</transacção>
```

Um exemplo de uma norma que utiliza o XML desta maneira é o SMIL ("Synchronized Multimedia Integration Language"), que usa anotações XML para identificar e controlar a apresentação de documentos que contêm texto, imagens, som e fragmentos de video, e assim criar apresentações multimédia.

Vamos terminar este preâmbulo com uma resposta directa à pergunta "Porquê tanta euforia à volta do XML?":

Um documento XML é texto, e o texto é a representação mais simples para a informação; lembro-me ainda da abertura da conferência mundial de SGML em 1997 (onde pela primeira vez se falou de XML), onde o guru das bibliotecas digitais dizia "Não se deixem ofuscar pela multimédia ", nessa altura era a moda, "a representação de informação passará sempre por formas textuais".

O XML é neutro, e este é, talvez, o argumento com mais peso a favor dele, não depende de plataformas de software ou hardware, nenhuma empresa pode reclamar direitos sobre ele; se eu tiver a minha informação em XML, consigo processá-la da mesma maneira num sistema Linux Windows ou Mac sem alterar uma vírgula.

XATA 2004

Passado um ano após a realização da primeira workshop sobre XML, a XATA 2003, o interesse na área não diminuiu, muito pelo contrário, assiste-se a um interesse crescente na área traduzido numa grande procura por cursos de formação e bibliografia na área.

A XATA 2004 surge, então, naturalmente, na sequência da XATA 2003. Este ano optou-se por designar o evento de Conferência tentando aumentar o nível de exigência e qualidade do mesmo: os autores submeteram artigos completos, houve um painel de revisores que realizaram um trabalho cuidado de revisão, os autores tiveram feedback dos seus trabalhos e receberam sugestões no sentido de os melhorar para a publicação final.

Ao analisar o conteúdo deste volume conclui-se facilmente que o tema que despertou maior interesse da comunidade XML gira, actualmente, à volta dos Web Services. O desenvolvimento de Web Services ocupa metade das actas e é aqui discutido e apresentado de uma forma multifacetada. Outras áreas como metainformação, bases de dados, dialectos, métodos formais e transformação de documentos, também estão presentes mas com menor expressão.

Estrutura da Workshop

A Workshop encontra-se dividida em oito sessões temáticas:

s1,s2,s3 e s4 – Estas sessões são dedicadas aos Web Services que este ano é o tema forte. Assim, temos várias apresentações convidadas de empresas, ViaTecla, FORDESI, MIND e Microsoft, e outras do meio académico e do meio empresarial (PT Inovação) que responderam ao pedido de trabalhos.

s5 – Esta sessão está dedicada ao desenvolvimento de aplicações XML e ao intercâmbio de informação entre aplicações.

s6 – A Metainformação é já um caso tradicional da aplicação do XML. Nesta sessão, são apresentados trabalhos focando vários contextos de aplicação.

s7 – Nesta sessão, apresentam-se várias linguagens XML desenvolvidas ou utilizadas e adaptadas pelos autores com objectivos aplicacionais específicos.

s8 – Debaixo do tema "Tecnologias", 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.

Penso que o XATA 2004 tem todos os ingredientes necessários para motivar a comunidade XML portuguesa a juntar-se durante dois dias para discutir e partilhar ideias.

No fim, talvez todos fiquemos a conhecer mais um pouco do que por cá se faz e quem sabe não surjam embriões de futuros projectos ou colaborações.

Uma boa XATA para todos

Fevereiro 2004 José Carlos Ramalho

Agradecimentos

Os editores deste livro não querem, aqui, deixar de agradecer a todas as pessoas, sem as quais esta obra não teria sido possível.

Em primeiro lugar, agradecemos a todos os autores a produção dos trabalhos que aqui são publicados. Sem eles não haveria conferência nem livro de actas. Obrigado por terem decidido participar nesta aventura.

Agradecemos aos revisores o terem tornado possível, com o seu trabalho de revisão cuidada a passagem do *preprint* ao *posprint* dos trabalhos que foram submetidos à conferência. Sem eles a qualidade do material aqui impresso seria com certeza outra.

Aos membros da Comissão Organizadora, agradecemos as várias diligências empreendidas para que o XATA2004 se tornasse uma realidade.

Também não nos podemos esquecer do pessoal administrativo, na FEUP e no DI/UM, que directa ou indirectamente também colaborou connosco para agilizar os pormenores administrativos. A eles o nosso obrigado.

Por último, a todos os participantes por tornarem real este fórum que vai crescendo à volta do XML. Obrigado e voltem nas próximas edições.

Uma boa XATA2004 com os nossos agradecimentos
José Carlos Ramalho
Alberto Simões

Comissão Ciêntífica

Adérito Marcos	CCG/DSI/UM
Ana Alice Baptista	DSI/UM
Gabriel David	FEUP/UP
João Correia Lopes	FEUP/UP
José Carlos Ramalho	DI/UM
José João Almeida	DI/UM
José Paulo Leal	DI/FC/UP
Luis Carriço	DI/FC/UL
Luis Ferreira	IPCA
Maria Cristina Ribeiro	FEUP/UP
Marta Jacinto Henriques	ITIJ
Miguel Mira da Silva	IST/UL
Pedro Antunes	DI/FC/UL
Pedro Henriques	DI/UM

Comissão Organizadora

Gabriel David	FEUP/UP
Maria Cristina Ribeiro	FEUP/UP
João Correia Lopes	FEUP/UP
José Carlos Ramalho	DI/UM
Pedro Henriques	DI/UM
Vitor Santos	Microsoft
José António Silva	Microsoft
Jorge Gustavo Rocha	DI/UM
Giovani Librelotto	DI/UM
Alberto Simões	DI/UM
Gustavo Arnold	DI/UM

Patrocinadores

http://www.softwareag.com/portugal/

Conteúdo

I WebServices: arquitecturas e casos práticos

Web Services aplicados a sistema de vigilância de recursos hídricos 3
Luís Abreu

Cyclone Code Generator 4
Nuno Bento

Interacção Independente de Dispositivo e de Localização com uma Arquitectura de Componentes 6
Carlos Ângelo Pereira, João Correia Lopes

KEYforTravel: Disponibilizar produto Turístico de uma forma eficiente e modular 21
Pedro Seabra, Filipe Costa Clérigo

Novas Arquitecturas baseadas em Web Services 22
José António Silva

Acesso Móvel a Serviços Bancários com tecnologias XML 23
Miguel Biscaia, Francisco Costa, José Alves de Castro e Vitor Santos

II Web Services: desenvolvimento de aplicações

Web Services: Metodologias de Desenvolvimento 33
Carlos J. Feijó Lopes, José Carlos Ramalho

Partilha de dados usando serviços web 45
José Paulo Leal, Rogério Ferreira

Memórias de Tradução Distribuídas 59
Alberto Manuel Simões, José João Almeida, Xavier Gomez Guinovart

III Web Services e Informação Geográfica

M-GIS — Sistema Móvel Interoperável de Informação Geográfica 71
Jorge Cardoso, Artur Rocha, João Correia Lopes

Web Services na Informação Geográfica 88
Mário André Araújo, Jorge Gustavo Rocha

SVG WebSlide & SVG WebChart: aplicações com gráficos vectoriais escaláveis em XML 101
Helder Filipe P.C. Ferreira

IV Desenvolvimento de Aplicações e Intercâmbio de Informação

Desenvolvimento Estruturado de Aplicações Web com Cocoon 115
Márcio Ferreira, João Pias, Célio Abreu, Rui Alberto Golçalves

Interoperabilidade entre Sistemas de Informação Universitários 123
Sérgio Sobral Nunes, Gabriel David

Administração e monitorização de uma solução Disaster Recovery 136
André Vila Cova, José Ruivo, Tomás Pereira

V XML e Metainformação

Aquisição e Armazenamento de Metainformação no Contexto de um Arquivo 147
Miguel Ferreira mferreira@dsi.uminho.pt DSI/UM, José Carlos Ramalho jcr@di.uminho.pt DI/UM

Utilizacion de RDF y bases de datos de metadatos nativas dentro del proyecto Omnipaper 166
Cesar Ariza, Ana Alice Baptista

Um Extrator de Topic Maps a partir de Recursos Heterogêneos de Informação 170
Giovani Rubert Librelotto, José Carlos Ramalho, Pedro Rangel Henriques

VI Dialectos XML

JavaML 2.0: Enriching the Markup Language for Java Source Code 183
Ademar Aguiar, Gabriel David, Greg Badros

Protótipo de um sistema para elaboração e manutenção de um manual da qualidade usando tecnologia XML e Docbook 197
Marco Rodrigues, Jenny Ferreira

Música e XML 205
David Freitas, Jorge Amaral

VII XML e Tecnologias

TX — validação de XML baseada em tipos dinâmicos 217
José João Almeida, Alberto Manuel Simões

Comparação de Linguagens para descrição de Interfaces baseadas em XML 225
Ricardo Alexandre Martins, José Carlos Ramalho, Pedro Rangel Henriques

Gerador de Web Services para cadeias de tranformações de documentos XML 226
José Carlos Ramalho, Pedro Taveira, Ricardo Ferreira, Vasco Rocha

Author Index 233

Parte I

WebServices: arquitecturas e casos práticos

Web Services aplicados a sistema de vigilância de recursos hídricos

Luís Abreu

MIND, SA

Resumo Pretende-se apresentar um caso prático de Web Services na reconversão tecnológica de um sistema proprietário na área da vigilância e monitorização.
A abstracção das camadas persistentes de apresentação, de regras de negócio e de dados em tempo real permitem uma homogeneidade no acesso de múltiplas aplicações-cliente desenvolvidas em plataformas distintas mantendo no entanto o carácter desconectado do modelo proprietário original.
A complexidade de tratamento dos dados e detecção de situações de alarme, encapsulada nas antigas aplicações cliente, foi transferida para serviços ”broker”, mantendo desta forma o carácter ”stateless”dos web services.
Múltiplos serviços, como por exemplo sistema de alertas para SMS, website WAP e PDA beneficiam desta arquitectura, constituindo a base para meios futuros de disseminação e consulta.

Cyclone Code Generator

Nuno Bento

FORDESI
Formação Desenvolvimento e Investigação, SA

Resumo O Cyclone é uma plataforma de desenvolvimento ágil de software que dado um meta-modelo da aplicação a desenvolver, gera automaticamente um conjunto de componentes de suporte ao desenvolvimento que, em alguns casos, chega a representar 90% do total do código. Através da eliminação das tarefas de programação repetitivas e altamente influenciáveis por erros, o Cyclone permite aumentar significativamente a eficiência e fiabilidade das tarefas de desenvolvimento reduzindo ao mesmo os longos ciclos de aprendizagem e adaptação.
O sistema pode ser disponibilizado localmente numa intranet, ou na Internet criando um serviço de geração de código, no qual o cliente escolhe a arquitectura, define o modelo da aplicação e de seguida obtém o código gerado. Neste caso o Cyclone define um novo conceito de comércio electrónico de software à medida no qual se vende um framework de desenvolvimento gerado à medida do modelo que o cliente pretende desenvolver cliente.
Existe um repositório centralizado que contem os modelos e um componente cliente que através de serviços Web acede à informação contida no repositório e gera o código.
O repositório é alimentado através do upload de ficheiros no formato XMI (standard OMG) este formato pode ser obtido através de interfaces de exportação existentes na maior parte das ferramentas UML.
O funcionamento do sistema é suportado pelos componentes da figura 1.

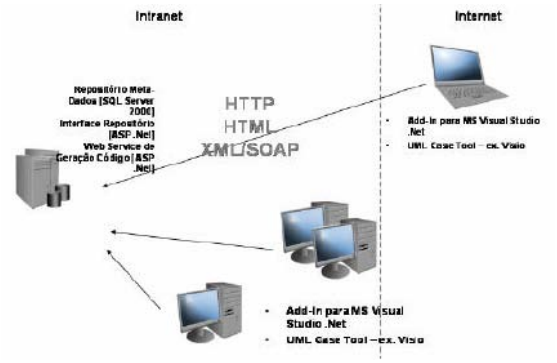


Figura 1. Arquitectura

- Repositório do Meta Modelo, é gerada uma base de dados para cada projecto criado no sistema.
- Web Service de importação de XMI, este serviço recebe um ficheiro XMI e o identificador do projecto a que se destina para processar e armazenar o seu conteúdo no repositório correspondente.
- Web Service para obtenção de informação relativa ao modelo, meta-dados e estrutura de geração do projecto. É usado pela aplicação de geração cliente.
- Aplicação Web para subscrição e configuração do serviço de geração de código.
- Add-In para o Visual Studio .NET que é descarregado do servidor e instalado remotamente, no cliente para que o código e a estrutura do projecto sejam automaticamente gerados.

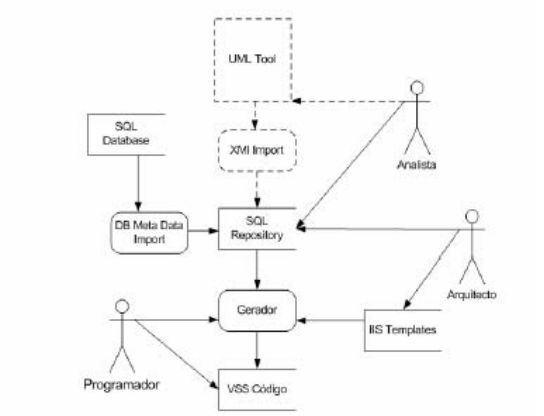


Figura 2. Processo de Geração

O processo de geração é baseado na aplicação de padrões de desenvolvimento, definidos pela arquitectura, à definição da aplicação contida no repositório.
O próprio meta-modelo é gerado e pode por isso suportar uma grande variedade de famílias de aplicações, sendo no entanto a mais típica a família das aplicações empresariais baseadas em bases de dados.
Deste processo resultam componentes de software prontos a ser utilizados por programadores no desenvolvimento.
O código é gerado no cliente pelo Add-In que obtém a estrutura do projecto a gerar, bem como os meta-dados contidos no repositório central. Em cada passo de geração é executado um padrão de geração com um conjunto de meta-dados resultando um determinado componente da arquitectura. Uma arquitectura é composta por um conjunto de camadas e respectivos padrões de geração, e por um meta-meta-modelo que define o formato do meta-modelo para uma dada família de aplicações.

Interacção Independente de Dispositivo e de Localização com uma Arquitectura de Componentes

Carlos Ângelo Pereira[1], João Correia Lopes[2]

[1] EFACEC Sistemas de Electrónica S.A., Portugal
{capereira}@se.efacec.pt
[2] Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto, Portugal
jlopes@fe.up.pt
http://www.fe.up.pt/∼jlopes/

Resumo Recentemente têm surgido equipamentos de pequeno porte e a custos suportáveis que permitem aceder remotamente aos sistemas de informação das empresas. Estes equipamentos, tais como *Personal Digital Assistants* (PDAs) e telemóveis com WAP, possuem tipicamente capacidades de apresentação e de largura de banda de transferência limitadas e, por isso, requerem a produção de interfaces para o utilizador específicas.
Os sistemas SCADA (*Supervisory Control and Data Acquisition*) são muito usados na indústria, nomeadamente em redes de energia eléctrica, e o acesso à informação por eles tratada, através de sistemas de pequeno porte, pode resultar numa mais valia importante para as empresas.
Neste trabalho propõe-se uma arquitectura, baseada em Serviços Web e em XML, que possibilita a criação e fácil manutenção de interfaces para componentes baseados no paradigma editor/subscritor em uso na Efacec para sistemas SCADA. Os componentes desenvolvidos usando SOAP e WSDL, permitem a interacção a partir de qualquer ponto da Web, mesmo através das *firewalls* instaladas nas empresas por razões de segurança. A camada de apresentação é conseguida com recurso a um componente desenvolvido seguindo o padrão *Front Controller*, uma variante do padrão MVC (*Model View Controller*).
Por forma a validar a arquitectura proposta foi desenvolvida uma aplicação que permite a realização de ordens de manobra. Os testes conduzidos permitem concluir que os requisitos de interacção deste tipo de aplicações foram satisfeitos, em conjunto com o desiderato de independência de localização e de dispositivo.

1 Introdução

A Web permite o acesso através de interfaces universais a informação localizada em qualquer ponto da Internet, usando protocolos universais. O protocolo em uso na Web para interacção com a lógica do negócio das aplicações, *Hypertext*

Transfer Protocol (HTTP), é actualmente permitido nas *firewalls* instaladas nas empresas para proteger a informação de acessos não autorizados, estando normalmente a porta usada por este protocolo acessível do exterior. No entanto este protocolo simples, leve e sem manutenção de estado, suporta apenas pedidos de recursos e respectiva devolução de conteúdos estáticos ou produzidos dinamicamente. O XML (*Extensible Markup Language*), sendo um formato universal para troca de dados entre aplicações [3], possibilitou o aparecimento de novos protocolos. Um destes protocolos, o SOAP (*Simple Object Access Protocol*), foi desenvolvido para suprir as limitações de acesso através de *firewalls* ou *proxys* experimentadas pelas aplicações distribuídas desenvolvidas em CORBA, EJB ou outros *middlewares*. O SOAP pode ser utilizado sobre HTTP, entre outros protocolos, sendo as mensagens representadas em XML.

Os Serviços Web (*Web Services*) [13] permitem o desenvolvimento de aplicações distribuídas, com interacção entre componentes localizados em qualquer local da Internet, desenvolvidos em qualquer linguagem de programação e disponibilizados por qualquer plataforma computacional. Para isso baseiam-se em protocolos standard em uso na Internet, permitindo a chamada remota a procedimentos através da troca de mensagens SOAP usando o protocolo HTTP, podendo desta forma atravessar as *firewalls*.

Os sistemas SCADA (*Supervisory Control and Data Acquisition*) [2] são bastante usados na indústria, nomeadamente em sistemas de geração de energia eléctrica ou em redes de transporte e distribuição de energia eléctrica, gás ou água. Estes sistemas possuem unidades remotas de aquisição e controlo, uma camada de comunicações e um centro de comando. Por cima dos sistema SCADA para redes de distribuição de energia eléctrica podem existir sistemas de suporte à decisão (DMS) e o acesso à informação por eles tratada através de sistemas de pequeno porte pode resultar numa mais valia importante para as empresas.

A implementação de um sistema de suporte a aplicações móveis justifica-se quando o número de utilizadores que precisa de aceder à informação a partir de dispositivos móveis é significativo. A existência deste acesso pode ainda contribuir para eliminar ineficiências, auxiliar o colaborador no exterior da empresa a decidir de forma documentada e com acesso a informação actualizada e pode ainda aumentar o grau de satisfação do cliente final.

A empresa "EFACEC — Sistemas de Electrónica S.A." desenvolve sistemas SCADA para a indústria eléctrica há alguns anos e, recentemente, desenvolveu um SCADA/DMS de nova geração denominado GENESys. Para suportar o desenvolvimento destas aplicações distribuídas, foi especificada e implementada uma arquitectura de software (BUS [9]) que utiliza o paradigma editor/subscritor para troca de informação entre componentes CORBA desenvolvidos em C++. Nesta empresa decorreram três teses de mestrado que procuraram alargar a utilização desta arquitectura a outras linguagens de programação e constituir pontes para a Web e para outras utilizações [10,7,6]. Na primeira destas teses, um dos autores deste artigo desenvolveu e demonstrou uma arquitectura que, utilizando XML e Serviços Web, permite interagir com independência de dispositivo e de localização, com o sistema SCADA/DMS GENESys.

São muito reduzidas as referências bibliográficas relacionadas com o tema da interacção com sistemas SCADA com independência de localização e de dispositivo, nomeadamente através de dispositivos de pequeno porte. O trabalho reportado em [15] tem por objectivo permitir a utilização de aplicações de controlo e aquisição de informação em clientes leves, possuindo interfaces sofisticadas, com um custo de implementação e utilização reduzido.

Na Secção 2 é apresentada a arquitectura de componentes distribuídos BUS que suporta o desenvolvimento de sistemas SCADA/DMS na EFACEC. Na Secção 3 são enumerados os requisitos a cumprir e é apresentado o desenho da arquitectura proposta neste trabalho, constituída por um "Mediador Web" (Web Mediator) e por uma "Zona Desmilitarizada" (DMZ). Na Secção 4 descreve-se a implementação do servidor SOAP, que suporta a interacção da zona desmilitarizada com a arquitectura editor/subscritor (BUS) e o "Mediador Web" que, para além de ser um servidor SOAP, é também um componente que recebe e envia eventos para o BUS. Na Secção 5 descreve-se a implementação do cliente SOAP e a lógica do negócio que envolve a validação e transformação XSLT para WML ou HTML do conteúdo XML vindo do BUS, dependendo do cliente Web em utilização. Na Secção 6 é apresentada a camada de interacção com o utilizador e o seu desenvolvimento usando o padrão *Front Controller*. Na Secção 7 descreve-se uma aplicação pensada com o objectivo de validar e avaliar a arquitectura desenvolvida, sendo apresentadas imagens retiradas da sua execução. A realização desta aplicação permitiu avaliar e colocar no lugar toda a panóplia de tecnologias necessárias ao funcionamento da arquitectura proposta. Finalmente, na Secção 8 são apresentadas as conclusões deste trabalho, juntamente com algumas possibilidades de melhorias futuras.

2 BUS — Arquitectura de Software

Nesta secção descreve-se a arquitectura de componentes distribuídos — BUS — que suporta a implementação do GENESys, um sistema SCADA/DMS de nova geração desenvolvido conjuntamente pela "EFACEC — Sistemas de Electrónica S.A." e pela "EDP — Electricidade de Portugal".

Os sistemas SCADA (*Supervisory Control and Data Acquisition*) são complexos e distribuídos e caracterizam-se pela dispersão de utilizadores e de dispositivos que supervisionam, normalmente à distância. Estes sistemas devem garantir elevada disponibilidade e segurança, dada a natureza sensível dos sistemas controlados (com riscos económicos mas também, e mais importantes, com riscos de integridade pessoal). Num sistema SCADA não se pode perder informação: todos os eventos reportados têm de ser tratados, quer para controlo em tempo real dos processos, quer para posterior estudo do comportamento dos equipamentos. Para sistemas de grande dimensão é usual associar ao sistema SCADA sistemas de apoio à decisão como, por exemplo, o DMS (*Distribution Management System*) para redes de energia eléctrica.

O BUS é uma arquitectura de software, especificada e implementada de forma a dar suporte às características do sistema SCADA/DMS GENESys. Esta arqui-

tectura utiliza o paradigma editor/subscritor para troca de informação entre componentes CORBA desenvolvidos em C++ [9]. Um sistema distribuído desenvolvido com recurso ao BUS é constituído por um conjunto de componentes que colaboram entre si através da troca de eventos. Um componente só envia ou recebe eventos de acordo com os tópicos que publica ou que subscreve, sendo os eventos entregues ao componente numa *callback*. A relação entre tópicos e eventos é especificada num meta-modelo de eventos.

Um componente é a unidade lógica de processamento fundamental do sistema, fornecendo uma funcionalidade, ou parte dela, e os componentes colaboram entre si para oferecer o conjunto de funcionalidades que se espera dum sistema. A colaboração entre componentes assenta no "modelo Push". O componente publicador faz o *push* de um evento para o BUS; baseado na configuração do meta-modelo, o BUS decide qual o tópico de entrega do evento e encaminha-o, pela informação do tópico, para todos os componentes subscritores desse tópico. O encaminhamento de eventos é assíncrono e desacoplado, isto é, a partir do momento em que um componente entregou o evento ao BUS, está livre para continuar com o seu próprio processamento. O componente não tem qualquer conhecimento dos potenciais interessados no evento: os componentes que subscrevem o tópico. De igual forma, não tem qualquer controlo sobre o momento de entrega dos eventos aos componentes subscritores.

Um evento é um objecto com estado, constituindo a quantidade mínima de informação que pode ser trocada entre componentes. É definido por um nome e constituído por pares atributo/valor, com significado para o utilizador humano. O evento tem uma representação serializada em texto, o "formato stringuificado" [8], para poder ser passado de acordo com o "modelo Push". Os objectos que representam o evento devem ser capazes de o construir a partir desse formato, mas também devem saber gerar essa representação.

Uma variante do evento, o "Data Request", pode ser visto como um serviço; é um pedido específico de informação realizado por um componente, que recebe, como resposta, dados que satisfazem esse pedido, orientados para o emissor do pedido. Um "Data Request" contém um evento e mantém as características de objecto que pode ser enviado por componentes. Possui, no entanto, duas características ligadas ao conceito de serviço: URN (*Universal Request Name*) que identifica o pedido de forma universal e RN (*RequestName*), o nome do pedido utilizado para diferenciar pedidos do mesmo tipo, caso seja necessário.

Sob o ponto de vista de implementação, o BUS é uma *framework* desenvolvida em C++, que se encontra construída no topo do ORB (*Object Request Broker*) Orbix. Esta *framework* estende as funções do ORB de forma a suportar um modelo de eventos de publicação/subscrição que permita uma comunicação baseada em ligações ponto-a-ponto.

3 Arquitectura Proposta

Este trabalho pretende desenhar, implementar e validar uma arquitectura que permita interagir, com independência de dispositivo e de localização, com o sis-

tema SCADA/DMS GENESys, nomeadamente a partir de dispositivos de pequeno porte e afastados geograficamente.

Assim, um dos requisitos óbvios da arquitectura procurada é o de suportar o envio e recepção de eventos com o BUS do sistema GENESys. Uma vez que os eventos são representados num formato proprietário é necessário traduzi-los para um formato universal que permita a utilização de processadores gerais standard, que podem ser retirados gratuitamente da Web e usados na lógica de negócio. Depois é então necessário processar estas representações em formatos universais por forma a construir uma interface adequada ao dispositivo de interacção.

Por forma a garantir independência de localização, é necessário que a arquitectura esteja acessível a partir da Web. O requisito de independência de dispositivo leva à necessidade de reconhecimento do tipo de dispositivo que faz o pedido e espera a respectiva resposta e a uma separação entre as camadas de apresentação e de lógica de negócio.

Por último e não menos importante, é necessário garantir que os pedidos e respostas atravessem as *firewalls*, instaladas pelas empresas para impedir acessos não autorizados a partir do exterior.

A figura 1 ilustra a arquitectura proposta, que inclui uma divisão em duas partes: "Mediador Web" (Web Mediator) e "Zona Desmilitarizada" (DMZ). O Web Mediator é um *proxy* que recebe pedidos da camada da lógica de negócio, fala com o BUS e produz documentos XML para serem transformados e apresentados nos dispositivos de interacção. A DMZ inclui os componentes de software que suportam a apresentação multi-dispositivo e toda a restante lógica de negócio.

A DMZ inclui o servidor Web e o servidor de aplicações e encontra-se limitada por duas *firewalls*. A primeira estabelece políticas de segurança em relação à rede exterior, deixando passar os pedidos ao servidor Web e as respectivas respostas. A segunda encontra-se entre a "Zona Desmilitarizada" e o sistema GENESys, que contém a informação crítica. Os utilizadores da rede exterior apenas têm acesso ao servidor Web que reside na DMZ, ficando assim garantida a segurança da informação crítica.

4 Interacção com o BUS

Por forma a que os pedidos da DMZ atravessem a *firewall* e assim possam aceder ao BUS, foi desenvolvido um Serviço Web, em que os pedidos são realizados em SOAP sobre HTTP. O cliente SOAP reside na camada da lógica de negócio, realizada recorrendo à plataforma Java, e o servidor SOAP estará do lado do sistema GENESys e, por isso, será implementado em C++.

Este Serviço Web interactua com o BUS através de um "Data Request". A sua interface (descrita por um ficheiro WSDL não incluído) inclui três parâmetros e uma string como resultado:

```
string xmlResponse DataRequest (string stringifiedEvent,
    string universalRequestName, string requestName)
```

sendo:

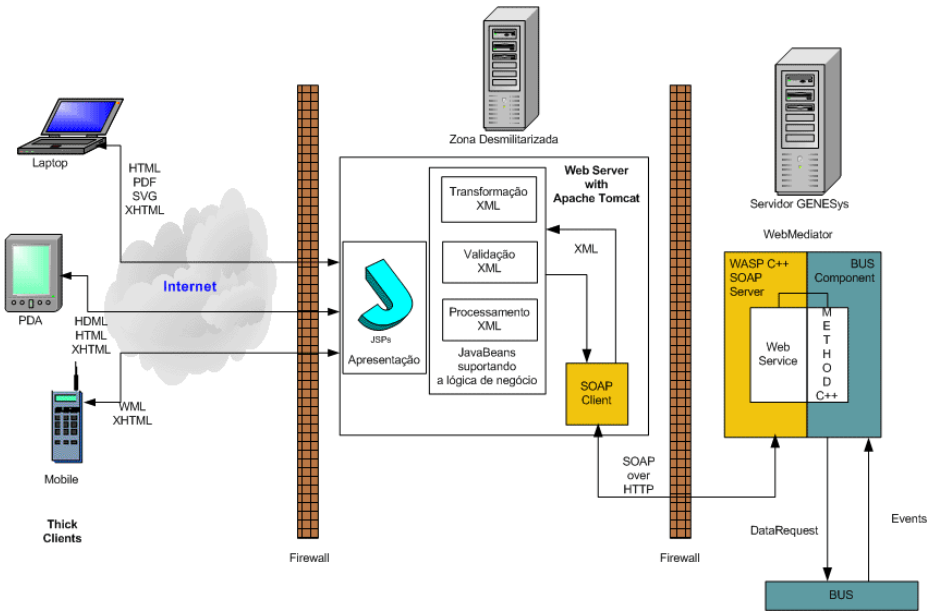


Figura 1: Arquitectura Proposta

`stringifiedEvent` O evento "stringuificado", obedecendo às regras definidas em [8], que permite definir a informação pedida (pelo nome do evento e atributos com valor associado);

`universalRequestName` Um URN (*Universal Request Name*) que permite identificar o tipo de pedido;

`requestName` Um RN (*Request Name*) que permite diferenciar pedidos do mesmo tipo;

`xmlResponse` Uma string com um evento descrito em XML construído com base na informação recebida num evento do BUS.

Para implementação do servidor SOAP foi escolhido o WASP da Systinet [14] dadas as suas vantagens de compatibilidade total com o standard SOAP 1.1, interoperabilidade comprovada com .NET, Apache AXIS e servidores J2EE, portabilidade para várias plataformas (Linux, Solaris, Windows, etc), suporte de segurança, bom desempenho e custo zero para instalação em máquinas de um só processador.

Para além da classe que implementa o Serviço Web, o Web Mediator tem ainda de ser um componente do BUS para enviar o "Data Request" e esperar pelo respectivo evento resposta. Assim, o Web Mediator desempenha as seguintes funções:

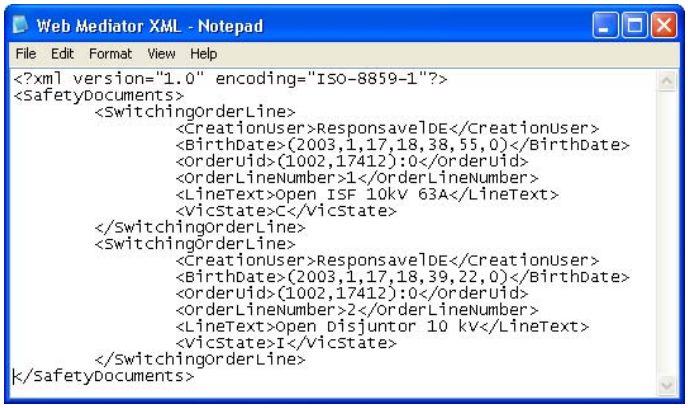


Figura 2: XML produzido pelo Web Mediator

Componente do BUS Acesso à API BUS conseguindo subscrever e publicar tópicos, enviar e receber eventos de outros componentes do GENESys e processar essa informação em "callbacks";

Servidor SOAP Ouvir os pedidos SOAP e invocar a instância que os processa;

Transformar eventos em XML Utilizar a API do BUS para processamento de eventos, conseguindo fazer a sua transformação num documento XML (por exemplo, o XML da Figura 2).

5 Camada da Lógica de Negócio

A camada da lógica de negócio, para além de conter o cliente do Serviço Web disponibilizado pelo Web Mediator, contém ainda as classes Java que realizam a validação e a transformação XSLT do conteúdo XML vindo do BUS para WML ou HTML, dependendo do cliente Web em utilização.

O cliente SOAP foi conseguido utilizando WASP Java e, dado que é conhecida a localização do serviço a utilizar, não foi necessário usar UDDI (*Universal Description, Discovery and Integration*). No Exemplo 1 apresenta-se o método `invokeDataRequestService` que invoca o Serviço Web.

Para ler o documento XML e analisar a sua estrutura foi usado o *parser* Xerces-J da Apache [12] e para processar o conteúdo do evento descrito em XML foi desenvolvida a classe `DOMGenesysParse`, que utiliza a API DOM da especificação JAXP (*JAVA APIs for XML Processing*). A validação do documento XML é opcional e pode ser omitida por razões de eficiência, sempre que este tiver origem numa fonte de confiança. Para a validação dos documentos XML representando os eventos, são usados esquemas XML (*XML Schema*) [5] sendo a API DOM JAXP notificada dos erros.

No GENESys já existe o meta-modelo de eventos contendo, guardada numa base de dados, a modelização da estrutura de tópicos, eventos e "Data Requests".

```
public String invokeDataRequestService (String stringifiedEvent, String
                                        universalRequestName, String requestName) {
  String serviceURL= "http://seetde24.se.efacec.pt:8080/genesys/wsdl/DataRequest.wsdl";
  String returnVal;
  try {
    WebServiceLookup lookup= (WebServiceLookup)Context.getInstance(
          Context.WEBSERVICE_LOOKUP);
    DataRequestPortType client= (DataRequestPortType)lookup.lookup(serviceURL,
          DataRequestPortType.class);
    returnVal= client.dataRequest (stringifiedEvent, universalRequestName,
          requestName);
  } catch (WebServiceLookupException e){
    returnVal= e.toString();
  }
  return returnVal;
}
```

Exemplo 1: Cliente SOAP Proxy.java

Para obter os correspondentes esquemas foram desenvolvidos procedimentos em PL/SQL que os produzem, por consulta à base de dados. Existem vários ficheiros com esquemas XML: um contendo a informação de todos os tópicos existentes; outro contendo as definições dos tipos de dados existentes (os tipos de dados que são utilizados pelo BUS); finalmente, um ficheiro por cada tópico do sistema contendo a definição de eventos para esse tópico e os atributos que pertencem a cada evento.

Para produzir uma representação adequada a cada dispositivo de interacção, o evento representado em XML é transformado usando uma transformação XSLT [4]. Na arquitectura proposta, o suporte de novos dispositivos reduz-se, praticamente, ao desenvolvimento de novos ficheiros XSL, para transformação dos conteúdos XML no formato de saída que esses dispositivos suportam. Numa primeira implementação, são suportados HTML e WML como formatos de saída.

Para a realização das transformações XSLT é usado o processador Xalan [11] da Apache. A API JAXP fornece um camada de adaptação aos processadores XSLT, à semelhança do que acontecia para os *parsers*. Esta abordagem, e tal como acontece para o processamento, permite a substituição de um motor XSLT por outro, caso seja necessário. A versão JAXP 1.1 inclui a API standard para motores XSLT (TRaX — *Transformation API for XML*), que foi usada na implementação.

A implementação da transformação dividiu-se em duas tarefas: a implementação de uma classe Java para realizar a transformação e o desenvolvimento de várias folhas de estilo XSLT (*.xsl) com as especificações das transformações a aplicar. A classe Java implementada (`TransformerBean`) permite encapsular todo o comportamento da transformação. Esta classe disponibiliza o método `doTransformation`, que tem como argumentos a folha de estilo XSLT (`xslFile`) e uma string XML (`xmlStream`).

As folhas de estilo realizam a transformação de XML para um outro formato, mais adequado às características do dispositivo cliente. No Exemplo 2 apresenta-se parte do código da folha de estilo XSLT que efectua a transformação para WML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="wml" indent="yes"/>
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="SafetyDocuments">
  <wml>
    <card id="GeneralSO" title="SOrder">
      <p><b>Order UID: </b>
      <xsl:value-of select="SwitchingOrderLine/OrderUid"/></p>
      <p><b>Created by: </b>
      <xsl:value-of select="SwitchingOrderLine/CreationUser"/></p>
      <do type="accept" label="Return">
        <go href="view_soid.wml"/>
      </do>
      <do type="accept" label="More">
        <go href="#SOLines"/>
      </do>
    </card>
    <card id="SOLines" title="SOLines">
      <xsl:attribute name="title">SO:
          <xsl:value-of select="SwitchingOrderLine/OrderUid"/>
      </xsl:attribute>
...
  </wml>
</xsl:template>
</xsl:stylesheet>
```

Exemplo 2: wml.xsl

A implementação realizada, por usar Java, constitui a uma solução de baixo custo e consegue ainda independência em relação à plataforma que a suporta, uma vez que funciona em qualquer Sistema Operativo.

6 Camada de Apresentação

A camada de apresentação realiza a interface com o utilizador. Recebe os pedidos dos utilizadores Web, analisa o tipo de pedido e os parâmetros definidos pelo utilizador e, utilizando a camada da lógica de negócio, constrói uma página com as características adaptadas ao tipo de dispositivo, para que o utilizador que fez o pedido consiga visualizar e interagir com a apresentação.

A adaptação de conteúdos pode ser efectuada no servidor, num intermediário (*proxy*) ou no navegador. A adaptação consiste na transformação do conteúdo que existe num formato, possivelmente mais propício à sua transmissão, num outro, adaptado às características do dispositivo que o vai consumir. Na adaptação no servidor ou por intermediação, as entidades que a realizam necessitam de saber as características do dispositivo. As características podem ser obtidas directamente, pela sua especificação, ou através dum identificador único do dispositivo cliente, utilizado para pesquisar a especificação num repositório.

Os servidores e *proxys* podem determinar a especificação de características particulares dum dispositivo utilizando o cabeçalho do pedido (*header request*) do protocolo HTTP. Existem outros mecanismos de especificação de características em desenvolvimento, que poderão ser utilizados alternativamente: o standard

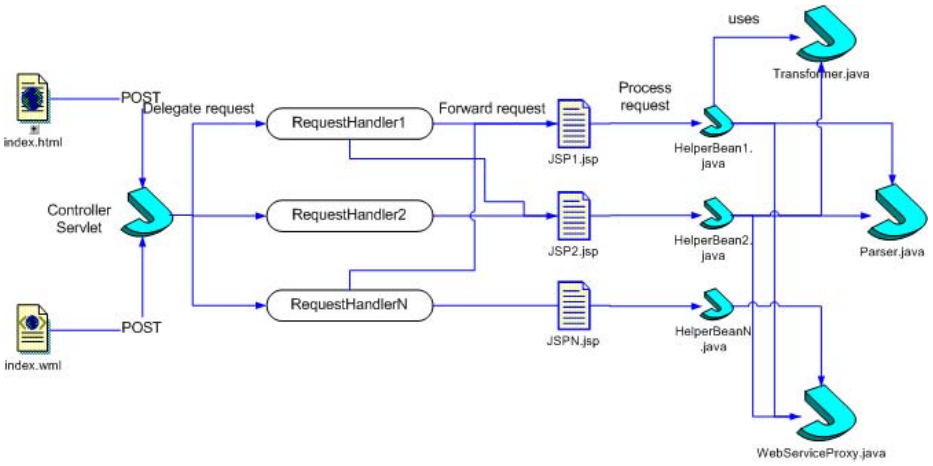


Figura 3: Camada de Apresentação

W3C *Composite Capability/Preferences Profile* (CC/PP), o standard *WAP User Agent Profile* (UAPROF), o standard *SyncML Device Information* (DevInf) e o standard *Universal Plug and Play* (UPnP).

A identificação das características do dispositivo não é realizada na implementação, devido à dificuldade em obter os parâmetros do cabeçalho HTTP pretendidos, quando o pedido é originado por certos navegadores (Internet Explorer 5, por exemplo). Por outro lado, as outras abordagens para identificação de características de dispositivos ainda estão em fase de desenvolvimento e também não foram consideradas. É utilizado um parâmetro do pedido que permite identificar, não só o tipo de cliente, mas também o contexto da aplicação que gerou o pedido (por exemplo, `wml-login` ou `html-login` para as interfaces de login em WML e HTML, respectivamente).

A implementação segue o padrão *Front Controller*, uma aplicação do padrão MVC (*Model View Controller*) [1] usado frequentemente para a implementação de aplicações Web, dadas as suas vantagens em termos de escalabilidade e facilidade de manutenção. Tal como é ilustrado na Figura 3, é usado um *servlet de controlo*, um *request handler* por cada tipo de pedido, *Java Beans* que disponibilizam a informação do modelo para as vistas, e um conjunto de páginas JSP (*Java Server Pages*) que disponibilizam as vistas. O *servlet de controlo* recebe todos os pedidos e delega o processamento nas classes que tratam cada tipo de pedido; cada *request handler* examina os parâmetros do pedido, actualiza o estado da aplicação, obtém a informação necessária, escolhe a vista JSP para a qual o controlador deve encaminhar a resposta e disponibiliza-lhe a informação necessária. As páginas JSP usam um `helperBean` para produzir uma representação adequada ao tipo de dispositivo e usam os componentes da lógica

de negócio `WebServiceProxy` (para obter a informação necessária), `Parser` (para a processar) e `Transformer` (para a transformar).

7 Exploração e Validação da Arquitectura

Nesta secção é apresentada uma aplicação que utiliza a arquitectura apresentada nas secções anteriores. Esta aplicação foi pensada com o objectivo de validar e avaliar essa arquitectura.

Uma ordem de manobras (OM) é uma lista de operações a efectuar sobre a rede eléctrica, seja planeada ou para gerir um acidente. O operador que está no Centro de Comando é responsável pela evolução de uma ordem de manobra, gerindo-a com o auxílio de uma aplicação informática. Algumas instruções (marcadas com "I" — "em instrução" — na Figura 4) são executadas no local, pelo piquete; este leva uma cópia da ordem de manobra, comunicando posteriormente ao operador a sua realização; o operador regista o facto (marcando a ordem com "C" — "confirmada"). Na Figura 2 apresenta-se um evento em XML, contendo informação sobre ordens de manobra, que foi capturado numa execução da aplicação desenvolvida.

A aplicação desenvolvida permite que um utilizador, que faça parte de um piquete, possa ter acesso a informação sobre ordens de manobra usando um PDA com HTML ou um telemóvel com WAP. O utilizador pode actualizar a ordem de manobra no local, imediatamente após a execução da manobra. A aplicação possui as actividades ilustradas na Figura 5: identificação do utilizador, identificação da OM, visualização da OM, selecção de uma linha da OM e confirmação de uma linha de OM em instrução.

Na Figura 6 apresentam-se as interfaces HTML e WML para o caso em que não existem ordens de manobra em instrução. Nessa situação, para a interface

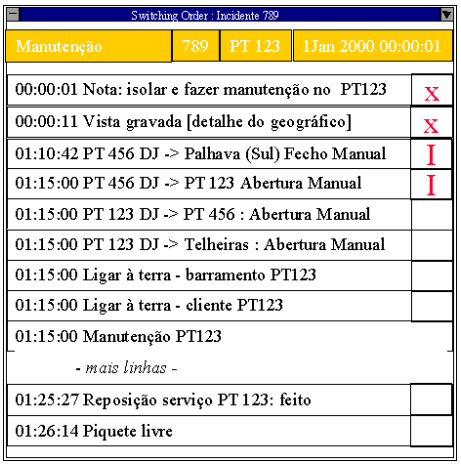


Figura 4: Ordem de manobras

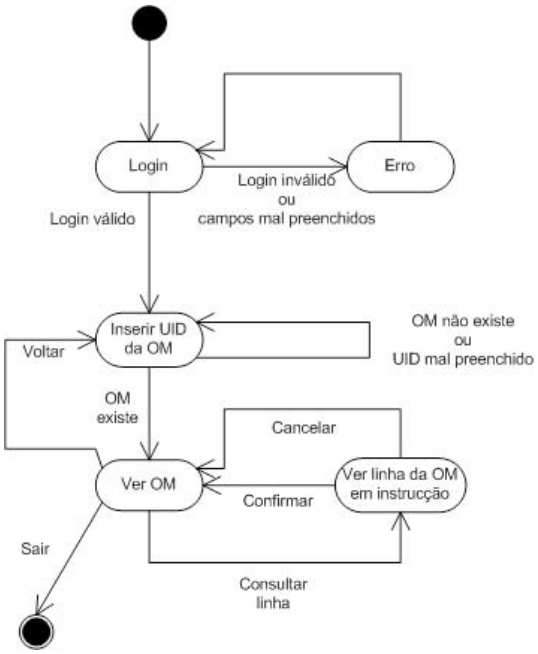


Figura 5: Actividades da Aplicação de OM

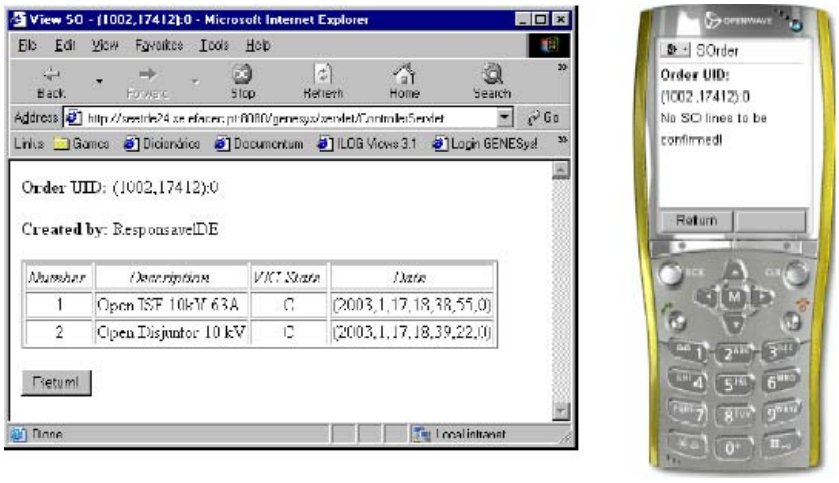


Figura 6: Exemplo de Interface HTML e WML

HTML todas as linhas serão mostradas com o respectivo de estado de confirmado ("C"), não existindo possibilidade de selecção. Na interface WML é apresentada uma mensagem que indica a inexistência de linhas em estado de instrução (com possibilidade de serem confirmadas).

A aplicação desenvolvida ilustra a interacção com o sistema GENESys, utilizando um navegador Web e um emulador WAP. A análise do desempenho não foi realizada, por este não ter sido um requisito identificado no âmbito deste trabalho.

8 Conclusões e Trabalho Futuro

O objectivo deste trabalho era o de propor e validar uma arquitectura que possibilitasse a interacção, através de dispositivos depequeno porte, com um sistema SCADA/DMS, cuja implementação se baseia numa arquitectura de componentes. A arquitectura deveria suportar independência da localização.

Analisando a aplicação desenvolvida e apresentada na Secção 7, podemos concluir que a interacção com o sistema GENESys foi demonstrada para dois tipos de dispositivos: clientes que utilizam navegadores Web e clientes que possuem telefones móveis com suporte para WAP. A utilização da Internet é comum a ambos, garantindo a independência de localização. Utilizando a arquitectura proposta, a interacção com o sistema foi conseguida numa aplicação que envolve identificação de utilizadores, verificação de privilégios, visualização e interacção com o BUS através de comunicação de eventos nos dois sentidos.

A utilização de Serviços Web permitiu a interoperabilidade entre sistemas heterogéneos, suportados em plataformas e linguagens de implementação diferentes, utilizando protocolos standard. A sua utilização permitiu a troca de informação entre o Web Mediator desenvolvido em C++ e as aplicações da "Zona Desmilitarizada", desenvolvidas em Java.

A utilização da família de tecnologias XML garante um formato estruturado que permite que a informação seja auto-descrita. A utilização do XML na arquitectura facilitou a interoperabilidade e a utilização da tecnologia Java para a sua manipulação.

O suporte em Java da camada de apresentação e da camada de lógica de negócio, tecnologia amplamente utilizada na implementação de sistemas de apresentação sofisticados e na implementação de lógica de negócio em vários sistemas Web, facilitou a implementação pela compatibilidade evidente com XML; a portabilidade e custo reduzido são outras grandes vantagens da sua utilização na arquitectura desenvolvida.

A utilização de uma solução baseada no padrão MVC permite uma separação clara entre a camada de apresentação e a camada de lógica de negócio, permitindo que o desenvolvimento de aplicações seja dividido em diferentes fases e realizado por diferentes técnicos (desenho de páginas Web, programação da lógica de negócio, integração de sistemas); a utilização desta solução trouxe vantagens, tais como a elevada reutilização e versatilidade, tempo de desenvolvimento curto e esforço de manutenção reduzido.

As possibilidades de trabalhos futuros são diversas, começando pela melhoria de desempenho e das funcionalidades da implementação desenvolvida, uma vez que esta envolve uma panóplia de tecnologias que foi necessário colocar no lugar: Tomcat, Xerces-J, Xalan, AXIS, JAXP, SOAP, WASP, servlets, JSP, HTML e WML. Como neste trabalho apenas foram realizadas interfaces para HTML e WML, é possível a extensão a outros tipos de formatos de saída, como por exemplo o XHTML ou o SVG (*Scalable Vector Graphics*) para representação de sinópticos (diagramas da rede eléctrica).

Uma maior preocupação com a segurança, já que têm aparecido propostas na área dos Serviços Web neste sentido e a ligação desta arquitectura a outros sistemas legados para a além do GENESys, são outros desafios interessantes para a extensão deste trabalho.

Com a previsível evolução das capacidades dos dispositivos portáteis de uso pessoal, tornar-se-á possível o desenvolvimento de "clientes gordos". Será possível desenvolver aplicações utilizando maior capacidade de processamento do lado dos clientes e já existem pacotes de software que permitem desenvolver um cliente SOAP a ser utilizado em dispositivos móveis de pequeno porte. O suporte de SOAP pelos computadores de bordo de automóveis seria o ideal para o desenvolvimento de novas aplicações, por exemplo para serem utilizadas por equipas de piquete.

De salientar que este trabalho tem suporte em aplicações industriais e o seu resultado beneficia essas aplicações, para além do interesse de investigação envolvendo a aplicação deste tipo de novas tecnologias.

Agradecimentos: Este trabalho foi suportado pela "EFACEC Sistemas de Electrónica S.A.".

Referências

1. Subrahmanyam Allamaraju, Cedric Buest, and John *et al.* Davies. *Professional Java Server Programming J2EE 1.3 Edition*. Wrox Press Ltd., Setembro 2001.
2. Stuart A. Boyer. *SCADA: Supervisory Control and Data Acquisition*. ISA — The Instrumentation, Systems and Automation Society, 2nd edition, Janeiro 1999.
3. T. Bray, C.M. Paoli, J. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, Outubro 2000. `http://www.w3.org/TR/2000/REC-xml-20001006`.
4. James Clark. XSL Transformations (XSLT) Version 1.0, W3C Recommendation. Technical report, W3C, Novembro 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.
5. David C. Fallside. XML Schema Part 0: Primer, W3C Recommendation. Technical report, W3C, Maio 2001. `http://www.w3.org/TR/xmlschema-0/`.
6. Ricardo Jorge Fernandes. Interfaces Web para Aplicações SCADA. Master's thesis, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, Março 2003.
7. Miguel Pereira Gomes. Canal SCADA na Web. Master's thesis, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, Fevereiro 2003.

8. Dave Marsh. O formato stringuificado. Technical report, Documento interno da EFACEC S.E., Março 2000.
9. Dave Marsh and Paulo Viegas. The BUS architecture. In *2001 IEEE Porto Power-Tech Proceedings, Porto, Portugal*, Setembro 2001.
10. Carlos Ângelo Paiva Pereira. Interacção com uma arquitectura de componentes com independência de dispositivo e localização. Master's thesis, Departamento de Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, Abril 2003.
11. Apache Xalan Project. Xalan, Dezembro 2002. `http://xml.apache.org/xalan-j/`.
12. Apache XML Project. Xerces-J, Dezembro 2002. `http://xml.apache.org/xerces-j/`.
13. James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilley & Associates, Inc., 2000.
14. Systinet. WASP Server for C^{++}, Dezembro 2002. `http://www.systinet.com/products/wasp_cserver/overview`.
15. Lukas Tan and Ken Taylor. Mobile SCADA with thin clients — a web demonstration. In *International Conference on Information Technology & Applications (ICITA 2002), Bathurst, Australia (25-28 Novembro de 2002)*, 2002.

KEYforTravel: Disponibilizar produto Turístico de uma forma eficiente e modular

Pedro Seabra and Filipe Costa Clérigo

VIATECLA
Soluções Informáticas e Comunicações SA

Resumo Com o objectivo de disponibilizar produto turístico em tempo real 24/7 a ViaTecla modulou a sua oferta dos motores do KEYforTravel através de XML Web Services.
Esta abordagem permitiu à ViaTecla focar o seu desenvolvimento nos vários motores de negócio, com as suas regras específicas, únicas ao mercado do turismo, sem ter que se preocupar em desenhar uma forma de os interligar nem estar preocupada com a camada de interface.
Com a adopção da .Net Framework, o KEYforTravel permite, através dos XML Web Services, uma forma standard de interligação e disponibilização dos seus motores de negócio em vez de utilização de uma API própria.
Esta apresentação pretende descrever o processo de criação desta solução. Mostrar a situação actual e real de utilização e discutir desafios de futuro.
Palavras-Chave: XML WebService; Interoperabilidade; Motores de Negocio; Turismo; Solução Modular

Novas Arquitecturas baseadas em Web Services

José António Silva

Microsoft

Resumo Ao longo do tempo, o nível de abstracção a que as funcionalidades são especificadas, publicadas e consumidas tem vindo a elevar-se gradualmente. Progredimos com os módulos, objectos (OO), componentes (CBD) e agora procuramos expor serviços (SOA). Embora estas arquitecturas orientadas para serviços não sejam novidade, foi a adopção generalizada dos XML Web Services (SOAP) por toda a indústria que fez renascer o interesse neste modelo.
Diferentes empresas estão a descobrir nos Web Services a solução para uma maior interoperabilidade, redução de dependências e uma forma de agilizar os seus sistemas. No entanto, mesmo com web services, também é possível implementar arquitecturas demasiado acopladas e síncronas, que mais se assemelham aos Remotings tradicionais (RPC).
Nesta sessão procuramos ainda fazer um ponto de situação das especificações e ferramentas que nos vão permitir implementar soluções transaccionais, robustas e seguras usando o SOAP

Acesso Móvel a Serviços Bancários com tecnologias XML

Miguel Biscaia, Francisco Costa, José Alves de Castro e Vitor Santos

PT Inovação

{miguel-b-pias,fcosta}@ptinovacao.pt

jac@natura.di.uminho.pt vsantos@ptinovacao.pt

Resumo Em tempos pretéritos, a implementação de serviços de telecomunicações móveis de valor acrescentado era um processo pouco flexível e de alto custo para o operador de telecomunicações. Desta forma, e com a evolução espectral de aplicações das tecnologias XML, surgiu a possibilidade de integração e acesso em tempo real a funcionalidades de próxima geração. Neste artigo, à luz dos conceitos de Web Services e XML, é feita a descrição e análise de uma plataforma de suporte ao acesso móvel a serviços bancários, como exemplo da multiplexagem de conhecimentos de telecomunicações e tecnologias XML.

1 Introdução

Em inúmeras regiões do globo, o acesso a serviços bancários pode ser muito limitado. A realização de uma simples operação bancária, por exemplo, pode implicar um deslocamento por parte do actor da operação de algumas centenas de quilómetros. Outras questões também se levantam no que diz respeito à segurança e comodidade.

Estas dificuldades foram minimizadas primeiramente com o surgimento dos terminais ATM e posteriormente com o advento do *Internet Banking*. As tecnologias de comunicação móvel vêm agora tornar ainda mais versátil o acesso a serviços bancários, não só tornando-o mais cómodo que o actual, mas também tornando-o possível em locais onde anteriormente não o era.

O sistema implementado pela PT Inovação para o operador sul-africano Mascom (Botswana) incorpora um conjunto de facilidades que permitem ao utilizador de um terminal móvel aceder à conta respectiva de um cartão de crédito através da rede Visa (*VisaNet*). Diversas operações bancárias poderão ser efectuadas: desde um simples carregamento de um telemóvel até ao pagamento de serviços em qualquer localização. De facto, as tecnologias XML foram centrais no processo de integração da plataforma de telecomunicações móveis com a rede Visa, quer no processo de submissão de novas operações, quer na disponibilização de uma interface de taxação das operações efectuadas.

2 Arquétipo do Sistema

2.1 Arquitectura Lógica

O processo de realização de uma operação bancária inicia-se com uma chamada telefónica efectuada através de um terminal móvel para um destino pré-definido. A rede de telecomunicações reencaminha essa mesma chamada para um IVR [1] (Interactive Voice Response), com o qual o utilizador navega em menus. O IVR, aquando da solicitação de uma nova operação por parte do utilizador, entra em comunicação com o elemento responsável pela interacção com a rede Visa (elemento Payment Gateway da figura 1). Este, após validar a informação que lhe é transmitida (e.g., número do cartão de crédito e identificação do utilizador), comunica à rede Visa a solicitação de nova operação e aguarda por uma resposta. É inerente à rede Visa o processo de interacção com as instituições financeiras relevantes para cada operação.

Após uma resposta de sucesso na execução da operação submetida, o Payment Gateway comunica através de uma interface SOAP com o sistema de taxação do operador de telecomunicações (Billing System), que procede à cobrança do serviço prestado ao cliente. Finalmente, é transmitido ao IVR a conclusão da operação, que por sua vez notifica o cliente. A arquitectura lógica de suporte é ilustrada na figura 1.

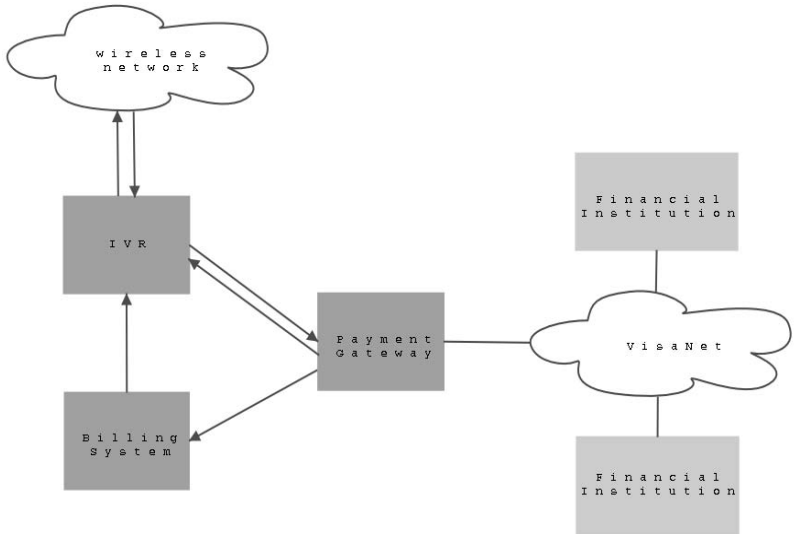


Figura 1. Arquitectura Lógica

[1] Um IVR consiste num elemento da rede capaz de interpretar *dial tones* e de tocar anúncios de uma forma programável.

2.2 Arquitectura Física

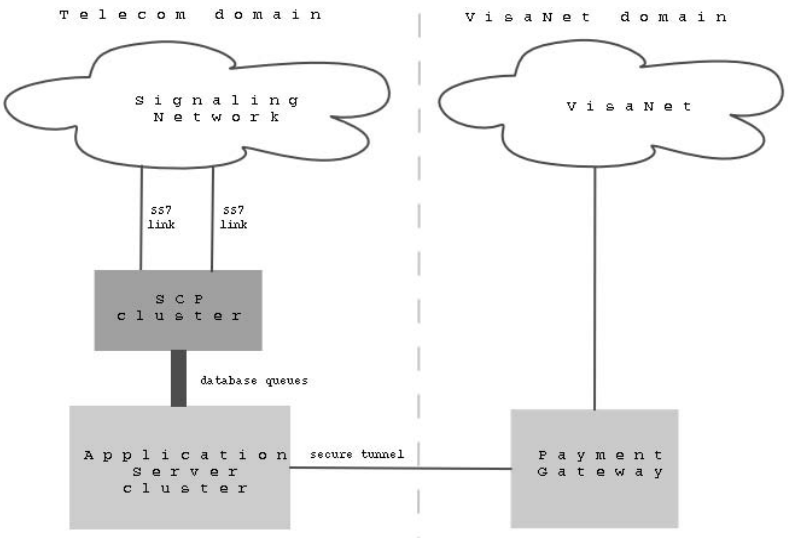


Figura 2. Arquitectura Física

Quando é iniciada uma nova chamada para o destino do serviço, o controlo da chamada é transferido para o elemento SCP (Service Control Point), responsável pela execução de toda a algoritmia do serviço e pela interacção com o utilizador. Adicionalmente, o SCP possui toda a informação relevante do cliente em base de dados e a capacidade de comunicar com outros sistemas através de *queues* de registos [2].

O Application Server consome a informação das *queues* do SCP e reencaminha-a para o Payment Gateway, com recurso ao protocolo HTTP, aguardando a sua resposta e colocando-a numa *queue* de entrada do SCP. Este elemento providencia também uma interface de taxação dos serviços bancários efectuados pelo cliente. A comunicação entre o Application Server e o Payment Gateway é assegurada por um túnel seguro, implementado com recurso ao protocolo SSH. Ao nível de hardware, os elementos SCP e Application Cluster são virtualizados por *clusters* com capacidades de *failover*. A arquitectura física do sistema é detalhada no domínio do operador, conforme descrito na figura 2.

```
<CardinalMobileMPI>
 <version>1.0</version>
 <category>BalanceInquiry</category>
 <msg_type>paym8_cli_lookup</msg_type>
 <msisdn>0836652770</msisdn>
</CardinalMobileMPI>
```

Figura 3. Exemplo de pedido XML de início de operação

3 Definição das Operações

Todas as operações executadas pelo sistema obedecem ao diagrama de estados ilustrado na figura 6. Para uma melhor compreensão do diagrama, são descritas as transições de estado assinaladas pelas letras em subscrito:

a) A transição do estado inicial para o seguinte acontece aquando do evento de recolha de dados do cliente e envio de um pedido de início de operação, por parte do IVR ao Payment Gateway. Este pedido segue o formato de uma mensagem definida em XML, conforme a figura 3, usando como transporte o protocolo HTTP. Neste ponto, o estado corrente da operação sinaliza que os dados colectados foram recebidos com sucesso pelo Payment Gateway.

b) No caso de necessidade de informação adicional para que a operação seja completada, o Payment Gateway envia uma mensagem em XML ao IVR solicitando essa mesma informação, transitando a operação para um estado de colecta de novos dados.

c) O IVR, após nova colecta de dados, envia-os novamente ao Payment Gateway através de uma mensagem XML em tudo idêntica à da figura 3. A operação retorna assim ao estado alcançado pela transição (a). Várias transições (b) e (c) consecutivas poderão ocorrer, dependendo da quantidade de informação necessária à execução da operação.

d) Durante a execução da operação, o Payment Gateway poderá enviar ao IVR uma mensagem de insucesso, caso do processamento dos dados recolhidos se possa concluir a impossibilidade de execução da operação (e.g., a conta bancária do cliente está congelada). A operação atinge um estado terminal de insucesso.

e) Numa operação bem sucedida, o Payment Gateway executa um procedimento remoto de taxação, recorrendo para tal à interface em Web Services disponibilizada pelo elemento Billing System. A mensagem SOAP enviada encapsula o pedido de execução de um procedimento remoto, tal como é exemplificado na figura 4.

f) Se a mensagem SOAP de resposta comportar um código de erro (elemento *code* do exemplo da figura 5) diferente de zero, ou uma mensagem de SOAP *fault* for devolvida, a operação transita para o estado de impossibilidade de

[2] A implementação das *queues* é suportada pelo pacote de software *Oracle Advanced Queueing*.

```
POST /axis/services/visa HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, multipart/related, text/*
User-Agent: Axis/1.1
Host: apserv1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "balanceenquiry"
Content-Length: 793

<?xml version="1.0"encoding="UTF-8"?>
 <soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance≫
 <soapenv:Body>
 <BalanceEnquiry
 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/≫
 <operator xsi:type="xsd:string≫qwert</operator>
 <msisdn xsi:type="xsd:string≫71403468</msisdn>
 <timestamp xsi:type="xsd:date≫2004-01-08</timestamp>
 <txid xsi:type="xsd:string≫qwrerty</txid>
 <balance xsi:type="xsd:float≫10.2</balance>
 <currency xsi:type="xsd:string≫072</currency>
 <srcacc_number xsi:type="xsd:string≫ola</srcacc_number>
 <srcacc_type xsi:type="xsd:string≫ola2</srcacc_type>
 </BalanceEnquiry>
 </soapenv:Body>
 </soapenv:Envelope>
```

Figura 4. Exemplo de mensagem SOAP de pedido de execução de taxação

taxação. Note-se que durante a transição de estado, o Payment Gateway desfaz junto das instituições financeiras a operação bancária corrente.

g) Dada a impossibilidade de taxação do serviço prestado, é sinalizado o insucesso ao cliente, transitando a operação para um estado final.

h) Esta transição de estado corresponde à notificação por parte do Payment Gateway ao IVR de sucesso na execução da operação bancária e na taxação do valor correspondente.

i) Por último, é reportado ao cliente o sucesso da operação pretendida.

4 Potencialidades da Arquitectura

A arquitectura desenvolvida permite uma eficaz integração do sistema da rede de telecomunicações com qualquer outro sistema externo. De facto, o módulo de software responsável pela transmissão de mensagens XML para o Payment

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Thu, 08 Jan 2004 11:23:56 GMT
Server: Apache Coyote/1.0
Connection: close

<?xml version="1.0"encoding="UTF-8"?>
 <soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance≫
 <soapenv:Body>
 <BalanceEnquiryResponse
 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/≫
 <request_id xsi:type="xsd:string≫
 714034682004010811 3258
 </request_id>
 <code xsi:type="xsd:int≫0</code>
 <description xsi:type="xsd:string≫OK</description>
 </BalanceEnquiryResponse>
 </soapenv:Body>
 </soapenv:Envelope>
```

Figura 5. Exemplo de mensagem SOAP de resposta comportando um código de erro

Gateway é genérico o suficiente para interagir com qualquer outra entidade, desde que esta suporte o mecanismo de troca de mensagens sobre o protocolo HTTP (adicionalmente, o módulo comporta a facilidade de transformação das mensagens de saída e entrada do SCP, através de XSLT).

As facilidades intrínsecas aos Web Services (nomeadamente do protocolo SOAP e da linguagem WSDL) possibilitam a rápida adaptação da interface de taxação a novos sistemas.

5 Conclusões

Apesar da dificuldade inicial de consolidação de conhecimentos e de acompanhamento da rápida evolução dos standards XML, a panóplia de tecnologias de diversas capacidades permitem o desenvolvimento eficaz e rápido de soluções que comportam a integração com sistemas externos, sem recorrer a tecnologias proprietárias difíceis de manter. Na realidade, a abertura e a quantidade de informação disponível sobre os standards, aliada à flexibilidade da linguagem XML, possibilitam o desenvolvimento de soluções com valor acrescentado para os clientes das redes móveis. O sistema implementado entrará em exploração proximamente num operador móvel do sul de África, sendo de prever a sua adopção por outros operadores da região.

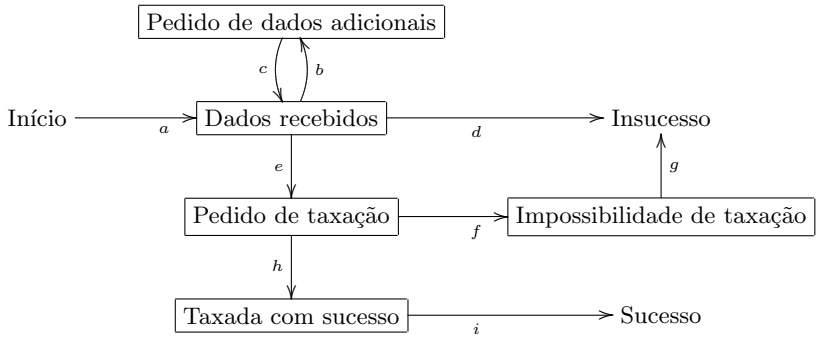


Figura 6. Diagrama de estados da operação

6 Referências

[HM01] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*, O'Reilly, Janeiro 2001.

[Wl01] Heather Williamson. *XML: the complete reference*, Osborne/McGraw-Hill, 2001.

[Pa02] *Paym8Secure – Mobile Prepaid Top-up IVR Integration Guide*, Cardinal Commerce, Novembro, 2002.

[Sh02] Rahul Sharma *Java API for XML-based RPC, JAX-RPC 1.0*, Sun Microsystems, Inc., 2002.

[Q1211] Recomendação ITU Q.1211: *Introduction to Intelligent Network CS–1*, Genebra, Maio 1995.

[W3C1] Recomendação W3C: *SOAP Version 1.2 Part 0: Primer*, Junho 2003.

[W3C2] Recomendação W3C: *SOAP Version 1.2 Part 1: Messaging Framework*, Junho 2003.

[W3C3] Recomendação W3C: *SOAP Version 1.2 Part 2: Adjuncts*, Junho 2003.

[W3C4] Recomendação W3C: *XML Schema Part 0: Primer*, Maio, 2001.

[HP97] T. Dehni, J. O'onnell, N. Reguideau, *Intelligent Networks and the HP OpenCall Technology*, Hewlett Packard.

[Ma96] T. Magedanz, R. Popescu-Zeletin *Intelligent Networks: Basic Technology, Standards and Evolution*, International Thompson Publishing, 1996.

[W3C5] *W3C Draft: SOAP Version 1.2 Usage Scenarios*, Junho 2002.

[W3C6] *W3C Note: Web Services Description Language (WSDL) Version 1.1*, Março 2001.

Parte II

Web Services: desenvolvimento de aplicações

Web Services:
Metodologias de Desenvolvimento

Carlos J. Feijó Lopes and José Carlos Ramalho

Departamento de Informática
Universidade do Minho

Resumo Os *Web Services* são uma tecnologia emergente, sobre a qual muito se tem especulado. No decorrer deste artigo efectua-se uma primeira contextualização, aproveitanto ao mesmo tempo para apresentar a arquitectura de funcionamento de um qualquer *Web Service*. De seguida, e aproveitando a implementação de um caso de estudo em quatro plataformas de desenvolvimento distintas, propomos metodologias de desenvolvimento para *Web Services* e respectivas aplicações cliente.

1 Contextualização

Os *Web Services* [12,5] são uma tecnologia emergente, sobre a qual muito se tem especulado. Uns apontam-na como o caminho a seguir no desenvolvimento de aplicações distribuídas, enquanto que outros vêm nelas apenas mais uma evolução de um conceito antigo.

Sendo aplicações modulares, auto-descritivas, acessíveis através de um URL, independentes das plataformas de desenvolvimento e que permitem a interacção entre aplicações sem intervenção humana, os *Web Services* apresentam-se como a solução para os actuais problemas de integração de aplicações.

Estas suas características devem-se em grande parte ao facto de se basearem em normas *standard*, de entre as quais se destacam: XML, SOAP, WSDL e UDDI.

Segue-se uma breve descrição das funcionalidades de cada uma destas normas:

XML - metalinguagem de anotação na qual estão definidas todas as outras normas que servem de base aos *Web Services*[6,7].

SOAP - linguagem de anotação com a qual se pode descrever o protocolo de comunicação, responsável pela troca de mensagens de e para os *Web Services*[1] (uma mensagem SOAP é um documento XML).

WSDL - linguagem de anotação definida em XML e que tem como objectivo descrever a API de um *Web Service*[4,11].

UDDI - linguagem de anotação definida em XML com a qual se cria a metainformação característica de um *Web Service*; vários registos UDDI são agrupados em repositórios; estes repositórios possuem uma interface/API de pesquisa para permitir a uma aplicação cliente pesquisar e localizar um serviço [3].

2 Arquitectura de funcionamento de um *Web Service*

O ciclo de vida de um *Web Service* compreende quatro estados distintos: `Publicação`, `Descoberta`, `Descrição` e `Invocação`. Vejamos com mais pormenor cada um deles:

`Publicação`: Processo, opcional, através do qual o fornecedor do *Web Service* dá a conhecer a existência do seu serviço, efectuando o registo do mesmo no repositório de *Web Services* (UDDI).

`Descoberta`: Processo, opcional, através do qual uma aplicação cliente toma conhecimento da existência do *Web Service* pretendido pesquisando num repositório UDDI.

`Descrição`: Processo pelo qual o *Web Service* expõe a sua API (documento WSDL); desta maneira a aplicação cliente tem acesso a toda a interface do *Web Service*, onde se encontram descritas todas as funcionalidades por ele disponibilizadas, assim como os tipos de mensagens que permitem aceder às ditas funcionalidades.

`Invocação`: Processo pelo qual cliente e servidor interagem, através do envio de mensagens de *input* e de eventual recepção de mensagem de *output*.

A cada um destes estados corresponde uma das normas anteriormente referidas, nomeadamente:

```
Publicação, Descoberta -> UDDI,

Descrição              -> WSDL,

Invocação              -> SOAP.
```

A conjugação destes quatro estados permite constituir o ciclo de vida de um *Web Service*, o qual passamos a descrever:

- O fornecedor constrói o serviço utilizando a linguagem de programação que entender;
- De seguida, especifica a interface/assinatura do serviço que definiu em WSDL;
- Após a conclusão dos dois primeiros passos, o fornecedor regista o serviço no UDDI;
- O utilizador (aplicação cliente) pesquisa num repositório UDDI e encontra o serviço;
- A aplicação cliente estabelece a ligação com o *Web Service* e estabelece um diálogo com este, via mensagens SOAP.

3 Metodologias de desenvolvimento

Com o objectivo de se definirem metodologias de desenvolvimento de *Web Services* e aplicações cliente, foram escolhidas quatro plataformas distintas: o módulo SOAP-Lite para Perl [13,9], o NuSOAP para PHP [2], o WASP Server para Java [8] e a .Net da Microsoft com recurso à linguagem C#.

Figura 1. Ciclo de vida de um *Web Service*

Como caso de estudo escolhemos um *Web Service* muito simples capaz de gerar um nº aleatório entre dois valores limite indicados por uma aplicação cliente.

No decorrer desta secção, tanto o desenvolvimento do *Web Service* como da respectiva aplicação cliente é efectuado na mesma plataforma de desenvolvimento, dada a intenção de explorar de forma abrangente cada uma das plataformas escolhidas. No entanto, e como indica o próprio conceito de *Web Service*, um *Web Service* desenvolvido por exemplo em PHP pode obviamente ser acedido por um cliente em C#. Note que todos os clientes desenvolvidos podem trabalhar com qualquer um dos servidores havendo apenas que mudar o ponto de acesso ao serviço.

Dado o estado prematuro do UDDI, as fases de Publicação e de Descoberta não foram tidas em conta na elaboração das metodologias. O atraso na adopção do UDDI, prende-se também com a inexistência de uma interface normalizada para os *Web Services* de determinada espécie, por exemplo, as agências de viagens podiam disponibilizar várias operações, no entanto, se tiverem diferentes assinaturas para a mesma operação o UDDI poderá indicar ao cliente um *Web Service* que não disponha dum método a invocar com o nome pretendido, mas sim um método com a mesma funcionalidade mas com nome diferente, o que poderá originar erros na aplicação cliente.

O primeiro passo na obtenção de metodologias de desenvolvimento de *Web Services*, consistiu na abordagem ao problema sob dois pontos de vista distintos: a criação do servidor, ou seja, do *Web Service* propriamente dito, e a criação da aplicação cliente.

Esta será também a abordagem a utilizar neste artigo, pelo que vamos se seguida apresentar a análise à criação do *Web Service* propriamente dito.

3.1 O Servidor

Ao falarmos na criação de um *Web Service*, estamos a contemplar duas ideias fundamentais: a implementação das funcionalidades pretendidas e a criação duma instância de um servidor SOAP com o qual pretendemos interagir.

É aqui que começam a surgir as primeiras diferenças entre as plataformas escolhidas.

Se por um lado temos o caso das plataformas NuSOAP e .Net, onde esta distinção é relativamente ténue (ver exemplos 1 e 2), para SOAP-Lite e WASP Server a distinção é bastante notória.

Exemplo 1: Servidor - PHP

```
<?php

require_once('nusoap.php');

$s = new soap_server;
$s->register('geraNumAleatorio');

function geraNumAleatorio($min, $max) {
    srand((double)microtime()*1000000);
    $numAleatorio = rand($min, $max);
    return $numAleatorio;
}

$s->service($GLOBALS["HTTP_RAW_POST_DATA"]);
?>
```

Como se pode observar, de uma só vez implementamos as funcionalidades e criamos uma instância de um servidor SOAP.

Exemplo 2: Servidor - .Net

```
...
namespace Aleatorios
{
    [WebService(Name="Números Aleatórios",
     Description="Web Service que gera um nº"+
     " aleatório entre um valor mínimo e um valor
     máximo dados pela aplicação cliente",
      Namespace="urn:CarlosLopes")]
```

```
9     public class Aleatorios : System.Web.Services.WebService
10    {
11        ...
12        [WebMethod(MessageName="geraNumAleatorio",
13         Description="método que gera um n° aleatório de acordo"+
14         " com os valores mínimos e máximos indicados")]
15        public int geraNumAleatorio(int min, int max)
16        {
17            Random r = new Random();
18            return r.Next(min,max);
19        }
20        ...
21    }
22 }
```

Neste caso basta indicar que a classe é um *Web Service* através do atributo `WebService`, e colocar o atributo `WebMethod` nos métodos que passarão a constituir a API do serviço.

Tal como foi indicado anteriormente, nas restantes plataformas a situação é um pouco diferente, isto é, existe uma clara separação entre o que é a implementação das funcionalidades e a sua associação ao servidor SOAP.

No caso do Perl com SOAP-Lite, para construirmos um serviço é necessário antes de mais criar uma instância de um servidor SOAP (exemplo 3).

Exemplo 3: Instância de Servidor SOAP - Perl

```
1 #!c:/perl/bin/perl.exe -w
2 use SOAP::Transport::HTTP;
3 SOAP::Transport::HTTP::CGI
4     -> dispatch_to('aleatorio')
5     -> handle;
```

No elemento `dispatch_to` indicamos qual o módulo onde estão implementadas as funcionalidades que se pretendem disponibilizar.

Estando na posse do nosso servidor SOAP, basta agora criar um módulo com as funcionalidades pretendidas (exemplo 4).

Exemplo 4: Implementação das funcionalidades - Perl

```
#!c:/perl/bin/perl.exe -w
package aleatorio;
sub geraAleatorio
{
    my ($self, $min, $max) = @_;
    $random = int(rand($max-$min)) + $min;
    return $random;
}
1;
```

No caso da plataforma WASP Server para Java, a situação apesar de manter esta noção de separação, é um pouco diferente do que se passou com o Perl. Neste caso, e em primeiro lugar, é necessário criar uma classe (aleatorio.java) com as funcionalidades pretendidas e só depois registar essa classe no servidor WASP, criando assim uma classe *proxy*[1] que representará o *Web Service* (exemplo 5 e figura 2 respectivamente) .

Exemplo 5: Implementação das funcionalidades - WASP Server

```
package aleatorio;
...
public class aleatorio {
    ...

    // métodos a disponibilizar remotamente
    public int geraAleatorio(int minimo, int maximo)
    {
        Random rand = new Random();
        return (rand.nextInt(maximo-minimo) + minimo);
    }
}
```

E estas são as grandes diferenças entre as plataformas quanto à criação do *Web Service* propriamente dito.

3.2 Aplicação cliente

Quando se fala na construção de uma aplicação cliente para um determinado *Web Service*, é comum confundir dois conceitos distintos: a interface de interacção

[1] O conceito de classe *proxy*, será abordado com maior detalhe na secção 3.2

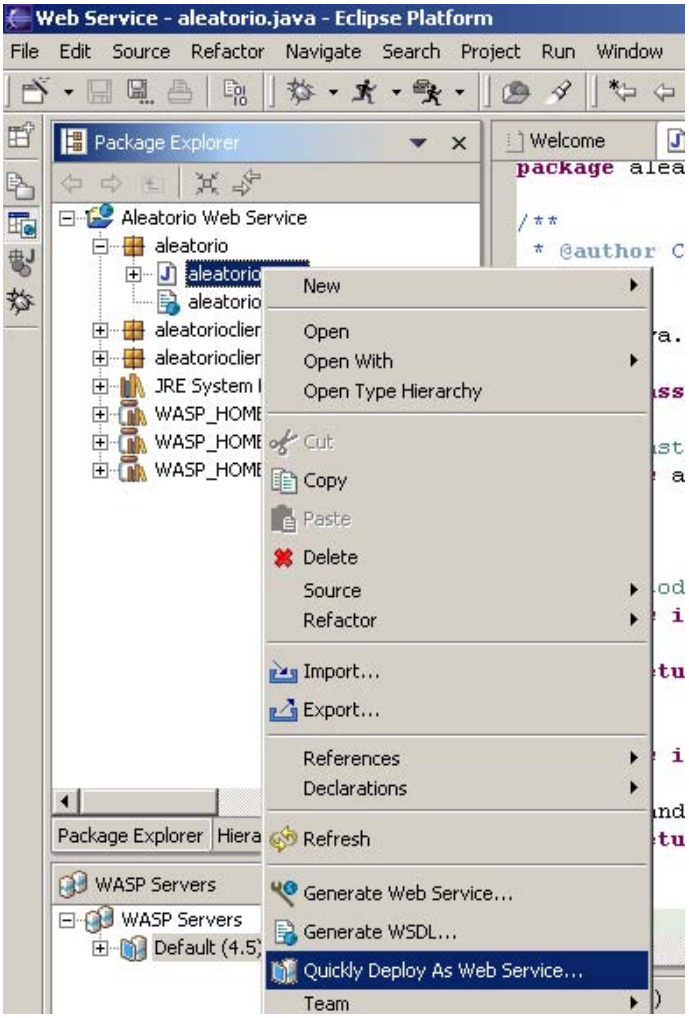


Figura 2. Criação de um *proxy* a partir da classe em Java

com o utilizador, e uma outra classe responsável pela ligação ao *Web Service* e respectivo tratamento dos dados.

Se por um lado a interface de interacção com o utilizador é bastante linear em todas as plataformas abordadas, uma vez que estamos a falar de aplicações *windows* ou *web* que correm do lado do cliente, o mesmo não se passou com a classe responsável pelo tratamento dos dados, pois é esta que terá de comunicar com o *Web Service*.

Aqui a separação deve ser feita em plataformas *Open Source* e plataformas proprietárias ou comerciais.

Vejamos então como criar um cliente em PHP com NuSOAP:

Exemplo 6: Cliente - PHP

```
<?php

require_once('nusoap.php');
$min = $_REQUEST["minimo"];
$max = $_REQUEST["maximo"];

$localizacaoServidor = 'http://localhost:8080/nusoap/
                        aleatorio/aleatorioserver.php';

$parameters = array('min'=>$min,
                    'max'=>$max);

$soapclient =& new soapclient($localizacaoServidor);
$resultado=$soapclient->call('geraNumAleatorio',$parameters);
...
?>
```

Como se pode verificar a criação da aplicação cliente não poderia ser mais simples, bastando criar uma instância de um cliente SOAP e de seguida invocar o método pretendido.

No caso do Perl com SOAP-Lite a criação do cliente é muito semelhante (exemplo 7).

Exemplo 7: Cliente - Perl

```
...
use SOAP::Lite;
use CGI qw(:standard);
```

```
4
5  my $soapserver = SOAP::Lite
6      -> uri('http://localhost:8080/aleatorio')
7      -> proxy('http://localhost:8080/cgi-bin/aleatorio.cgi');
8
9  my $min = param('minimo');
10 my $max = param('maximo');
11
12 my $resultado = $soapserver->geraAleatorio($min,$max);
13 ...
```

Como podemos ver, para criar a aplicação cliente basta criar uma instância de uma classe para comunicação com o servidor SOAP, onde no método `uri()` é necessário indicar o módulo onde se encontram definidas as funcionalidades disponibilizadas pelo *Web Service* (ver exemplo 4). Ao mesmo tempo é necessário indicar, no método `proxy()`, o endereço do servidor SOAP, i.e., a localização da *script* que efectua o acesso ao módulo (ver exemplo 3).

Como foi possível observar, no caso das plataformas *Open Source* apresentadas, a aplicação cliente acede directamente ao *Web Service*. Esta é uma das principais diferenças em relação às duas plataformas comerciais analisadas. Efectivamente, nestas últimas é introduzido o conceito de *proxy class* [10]. A classe *proxy* é construída a partir do documento WSDL do *Web Service* em causa, apresentando uma API de interacção com a aplicação cliente, constituída por métodos com os mesmos nomes dos métodos expostos remotamente pelo *Web Service*.

A classe *proxy* para além de permitir o estabelecimento da ligação entre o *Web Service* e a aplicação cliente, encapsula toda a complexidade inerente à utilização do SOAP e dos protocolos de transporte a utilizar. Na prática, a aplicação cliente liga-se à classe *proxy* e é esta que estabelece contacto com o *Web Service* propriamente dito.

Vejamos então o exemplo dum cliente desenvolvido em .Net.

Exemplo 8: Cliente - .Net

```
1 ...
2
3 AleatorioClient.localhost.NúmerosAleatórios ws =
4    new AleatorioClient.localhost.NúmerosAleatórios();
5
6 int aleatorio = ws.geraNumAleatorio(minimo, maximo);
7 ...
8 }
```

Como podemos verificar a criação da aplicação cliente é de facto muito simples. Após a criação da classe *proxy* (`AleatorioClient.localhost`), basta utilizá-la para criar uma instância do *Web Service* e a partir daí invocar o método `geraNumAleatorio` definido no exemplo 2.

Vejamos agora o caso de um cliente em WASP Server:

Exemplo 9: Cliente - WASP Server

```
...
import aleatorioclient.iface.Aleatorio;
...
String wsdlURI = "http://carloslopes:6060/aleatorio/wsdl";
String serviceURI = "http://carloslopes:6060/aleatorio/";
ServiceClient serviceClient = ServiceClient.create(wsdlURI,
                                          Aleatorio.class);
serviceClient.setServiceURL(serviceURI);
serviceClient.setWSDLServiceName(
        new QName("urn:aleatorio.aleatorio", "aleatorio"));
serviceClient.setWSDLPortName("aleatorio");
service = (Aleatorio) Registry.lookup(serviceClient);

int aleatorio = service.geraAleatorio(min, max));
...
}
```

Note-se a referência à classe *proxy* a utilizar para este *Web Service*.

```
import aleatorioclient.iface.Aleatorio
```

Os restantes passos para a criação da aplicação são bastante simples como se pode observar, bastando criar uma instância de um cliente do serviço, indicar a localização do serviço e invocar o método pretendido.

Estando apresentadas as principais diferenças, quer na criação do *Web Service* quer na criação da respectiva aplicação cliente, entre as quatro plataformas escolhidas, é tempo de apresentar um quadro resumo com as metodologias daí retiradas [10].

Tabela 1: Metodologias para desenvolvimento de *Web Services*

	Servidor	**Cliente**
PHP	Criação de uma instância de um servidor; Registo das funções a invocar remotamente; Especificação das funções;	Criação de um array com os parâmetros necessários à função a aceder remotamente; Criação de instância de cliente SOAP; Efectuar chamada ao método ao qual pretendemos aceder, e obtenção dos resultados;
Perl	Criação do módulo com as funcionalidades pretendidas; Criação da CGI com o servidor;	Criação de uma instância de uma classe para comunicação com o servidor SOAP; Invocação dos métodos pretendidos e obtenção de resultados;
.Net	Criação de classe funcional + extensão .asmx + tag <% Web Service ... %> **ou em alternativa** criar um novo projecto do tipo *Web Service*; Indicação dos métodos a expôr remotamente com o atributo `WebMethod`;	Criação de um *proxy*; Invocação dos métodos expostos pelo WS e obtenção de resultados;
WASP Server	Criação da classe funcional; Registo da classe como sendo um *Web Service* e consequente obtenção de classe *proxy*;	Inicialização de uma instância da classe *proxy*; Invocação dos métodos pretendidos e obtenção dos resultados;

De um ponto de vista mais abstracto podem-se retirar da tabela 1, alguns pontos comuns que convém salientar:

Servidor :

1. Criação de classe onde se implementam as funcionalidades que se pretendem disponibilizar para acesso remoto;
2. Identificação dessa classe como sendo um *Web Service*.

Cliente :

1. Criação de uma classe cliente;
2. Indicação à classe cliente da localização do serviço;
3. Invocação dos métodos pretendidos.

4 Conclusões

Os *Web Services* apresentam-se como a solução para muitos dos problemas associados aos sistemas distribuídos, nomeadamente nas questões relacionadas com

a integração de sistemas heterogéneos, razão pela qual têm estado rodeados por uma euforia nem sempre benéfica.

No entanto o facto de serem componentes de *software* modulares, auto-descritivas, que se baseiam em protocolos *standard* e que permitem a interacção entre aplicações sem intervenção humana, torna-os sem dúvida o futuro dos sistemas distribuídos.

A existência de plataformas de desenvolvimento que tornam transparente ao programador todo esforço de criação/interacção de mensagens SOAP e criação/utilização dos documentos WSDL que descrevem o *Web Service*, facilitam a sua adopção comercial, fomentando assim o seu desenvolvimento.

Dada a relevância dos *Web Services* no futuro das tecnologias de informação, torna-se oportuna a proposta de metodologias para o seu desenvolvimento. Para tal, aproveitou-se a implementação de um caso de estudo, para apresentar uma espécie de "receita"pronta a utilizar no desenvolvimento dos mesmos.

Referências

1. Simple object access protocol (soap). http://www.w3.org/TR/SOAP.
2. Site oficial do projecto php. http://www.php.net.
3. Universal description, discovery, and integration (uddi). http://www.uddi.org.
4. Web services description language (wsdl). http://www.w3.org/TR/WSDL.
5. Ethan Cerami. *Web Services Essentials*. O´Reilly, 2002.
6. W3C: World Wide Web Consortium. Extensible markup language (xml): version 1.0. http://www.w3.org/XML, Dezembro 1997.
7. J.C.Ramalho and Pedro Henriques. *XML & XSL Da Teoria à Prática*. FCA, 2002.
8. Tyler Jewell and Davis Chapel. *Java Web Services*. O'Reilly, 2002.
9. Paul Kulchenko. Quick start guide with soap and soap::lite. http://guide.soaplite.com, 2001.
10. Carlos Jorge Feijó Lopes. Web services: Aplicações distribuídas sobre protocolos internet. Master's thesis, Universidade do Minho, Janeiro 2004.
11. Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP and UDDI*. Addison-Wesley, 2002.
12. P.Cauldwell, R.Chawla, V.Chopra, G.Damschen, C.Dix, T.Hong, F.Norton, U.Ogbuji, G.Olander, M.Richman, K.Saunders, and Z.Zaev. *Professional XML Web Services*. Wrox Press, 2001.
13. Randy J. Ray and Pavel Kulchenko. *Programming Web Services with Perl*. O´Reilly, 2002.

Partilha de dados usando serviços web

José Paulo Leal[1] and Rogério Ferreira[1]

DCC/FCUP
{zp,roger}@ncc.up.pt

Resumo Os sistemas de informação são instrumentos fundamentais no funcionamento e na gestão das organizações, sejam estas empresas, instituições públicas ou privadas. Em organizações compostas por múltiplos organismos com sistemas de informação heterogéneos, a coordenação da partilha de dados é um problema complexo e crucial. Os serviços web surgem como uma possibilidade para a comunicação ponto a ponto entre estes organismos mas, quando o seu número é elevado, levantam o problema da sua definição, coordenação e controlo centralizado.
Este trabalho apresenta uma abordagem baseada em *standards* XML para lidar esse problema. A abordagem consiste numa sequência de etapas, desde a definição do modelo de dados do sistema de informação da organização, passando pela criação automática de formatos de dados comuns a partir do modelo de dados, e terminando na definição de serviços de a troca de dados entre os organismos.

1 Introdução

A comunicação de dados entre sistema heterogéneos é uma das áreas em que o impacte do XML tem sido mais preponderante: as linguagens XML permitem especificar formatos comuns para os dados; os documentos contendo esses dados podem ser validados usando linguagens como o XML Schema; e a comunicação desses dados pode ser feita com protocolos como o SOAP usando serviços da Internet como a web ou o email.

Os serviços web (ou *web services*) são a articulação destas tecnologias de forma a permitir expor funcionalidades dum Sistema de Informação (SI) a aplicações externas, por meio da Internet. Tal como numa aplicação web, o serviço web permite o acesso dum programa cliente a um SI, mas enquanto numa aplicação web os programas cliente são primordialmente navegadores com os quais interage um utilizador humano, num serviço web os programas cliente executam funções que só indirectamente estão relacionadas com um utilizador.

Os serviços web facilitam a comunicação de dados entre sistemas heterogéneos por duas razões principais: o formato de transferência dos dados (documentos XML) é independente das plataformas ou ambientes de desenvolvimentos, e facilmente convertível noutros formatos; os protocolos de transporte (HTTP ou SMTP) são suportados nos principais sistemas e são compatíveis com os mecanismos de protecção que rodeiam os servidores dos SI (como *firewalls*).

Os serviços web surgem assim como uma solução para a comunicação ponto a ponto entre vários SI. Este tipo de comunicação pode ser usado em vários cenários em que é necessário automatizar a transferência de dados. Neste trabalho focamos na questão da partilha de dados numa organização com SI heterogéneos. A existência de SI heterogéneos ocorre frequentemente em organizações duma certa dimensão constituidas por organismos com alguma autonomia. Apesar dessa autonomia a organização necessita de regular e optimizar o fluxo de dados entre os organismos que a compõem.

A título de exemplo consideremos uma faculdade constituída por vários organismos, como departamentos, bibliotecas, secretaria, etc. Cada um destes organismos poderá ter sistemas de informação autónomos mas para benefício do funcionamento global há conveniência em que os seus dados sejam partilhados: dados pessoais dos alunos, recolhidos pelos secretaria, podem ser enviados aos departamentos que reciprocamente podem enviar aos primeiros dados referentes ao percurso académicos dos alunos.

Usando serviços web cada organismo pode expor os seus dados aos restantes, permitindo que todos beneficiem dos dados da instituição. A opção pelos serviços web não colide com as escolhas de ambiente de implementação de cada organismo. No entanto esta abordagem descentralizada, em que cada organismo fornece os dados que produz, não facilita a definição de politicas comuns de distribuição de dados, definidas centralmente, por exemplo, pela direcção duma faculdade. Do mesmo modo não facilita que centralmente possam ser definidos quais os organismos com direitos de acesso a determinados dados, ou com a responsabilidade pela sua produção. Finalmente, esta abordagem descentralizada não facilita que a disponibilidade e consistência dos dados seja controlada por uma entidade reguladora, responsável pela qualidade dos dados.

Este trabalho toma como ponto de partida a utilização dum conjunto de serviços web para implementar a partilha de dados dentro duma organização, e descreve uma abordagem para minorar os problemas relacionados com a definição, coordenação e controlo centralizado dum conjunto de serviços web interrelacionados, usados para partilha de dados num organização.

1.1 Objectivos

Com este trabalho pretendemos organizar a partinha de dados entre organismos duma mesma organização, usando um conjunto de serviços web. Parte-se do princípio que cada organismo disponibilizará através dum serviço Web os dados que produz e que poderá através de programas-cliente aceder à informação de outros organismos (usando os seus serviços web). No âmbito deste trabalho pretendemos:

- especificar/representar globalmente o sistema de informação da organização;
- definir responsabilidades e direitos dos organismos relativamente aos dados;
- normalizar o formato de comunicação de dados;
- definir os serviços de dados dos organismos.

Pretendemos também que essas acções possam ser realizadas com linguagens e metodologias *standard*, com ferramentas disponíveis nas principais plataformas e ambientes de desenvolvimento, permitindo que a implementação dos serviços de dados possa ser feita em qualquer um deles. Note-se que este trabalho tem por objectivo apenas a definição e regras e a coordenação da partilha de dados entre organismos, não pretendendo focar questões de implementação.

1.2 Abordagem

A abordagem proposta consiste numa sequência de três etapas interrelacionadas, partindo da definição do modelo global de dados, passando pela definição dos formatos de dados comuns, concluindo com a definição dos serviços de dados, conforme esquematizado na Figura 1.

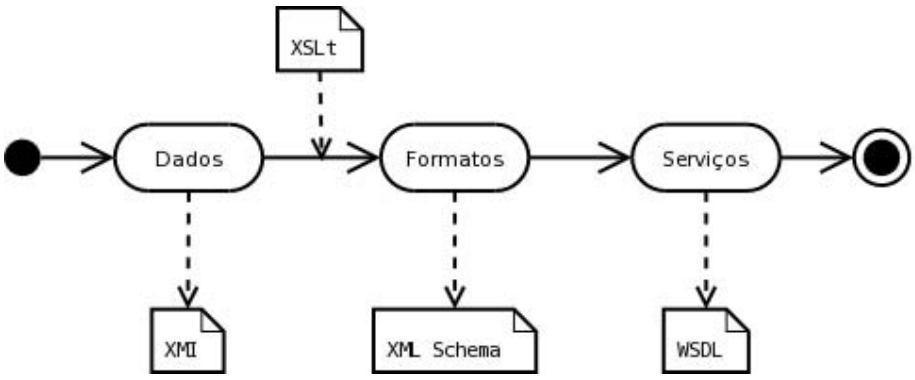


Figura 1. Esquema geral da abordagem.

Cada uma destas etapas tem como objectivo específico produzir um documento em formato electrónico que além de servir como referência para a implementação dos serviços de web possa também ser usado na automatização a sua implementação.

As etapas indicadas na tabela da Figura 2 estão relacionadas e todas tiram partido dos *standards* XML. A modelação de dados recorre ao UML sendo os modelos processados a partir da sua representação no formato XMI. A definição de formatos comuns para os dados a transferir recorre ao XML *Schema*. Esta definição de dados obtêm-se por aplicação de transformações XSLT sobre o modelo de dados em formato XMI. A comunicação de dados tem por base o protocolo de SOAP sendo a definição dos serviços prestados por cada organismo representados em WSDL. Nas secções seguintes são analisadas cada uma destas etapas referidas, focando a utilização das tecnologias XML associadas.

Etapa	Define	Linguagem	Descrição
Dados	Modelo de dados global	XMI	Representação dos dados relevante para mais do que um organismo e relacionamento entre dados. O modelo deve reflectir os direitos e responsabilidades dos organismos. É a este nível de abstracção que se tomam decisões para o sistema de partilha.
Formatos	Formatos de dados comuns	XML Schema	A partir do modelo de dados da organização define tipos de dados comuns para a partinha de dados. É obtido uma estrutura por cada organismo, que contem informação do seu âmbito.
Serviços	Serviços de dados	WSDL	Usando os formatos de dados são definidas as características dos serviços web dos diversos organismos.

Figura 2. Etapas da abordagem

1.3 Trabalho relacionado

Tanto quanto é do conhecimento dos autores não foram ainda publicados trabalhos sobre os problemas relacionados com a utilização de serviços web na partilha de dados, o que é provavelmente justificado por estas tecnologias serem bastante recentes. Contudo, existem diversos trabalhos que focam alguns dos problemas analisados e que influenciaram esta abordagem.

O estudo feito por Skogan [7] comprovou-nos que o UML com algumas restrições pode ser usado para definir a estrutura e semântica dos dados, e que aliado ao formato XML é o adequado para permitir interoperabilidade.

Existem vários projectos que definiram mapeamentos de modelos UML para definições de dados XML como: DISGIS European ESPIRIT project n. 22.084, o ISO Technical Committee n. 211 Geographic Information e ISO 15046. [5]. Estes projectos também serviram de inspiração para a nossa abordagem, embora se centrem apenas na definição de informação distribuída geograficamente, não abordando a partilha da mesma.

2 Modelos de dados

Para modelar o esquema de dados global da organização utilizamos o *Unified Modeling Language* (UML). O UML é uma linguagem gráfica para visualização, especificação, construção e documentação de sistemas baseados em *software* [3]. Esta linguagem é hoje em dia um *standard* de modelação, e num estudo feito por David Skogan [7] concluiu-se que o UML com algumas restrições pode ser usado para definir a estrutura e semântica dos dados.

Os modelos produzidos nas ferramentas de modelação UML podem ser guardadas no formato XMI - uma linguagem XML para intercâmbio de modelos UML [11], o que facilita a interoperabilidade das ferramentas UML. De facto, existem vários projectos de modelação de dados que definiram este processo de mapeamento de modelos UML para definições de dados XML como: *DISGIS European ESPIRIT project n. 22.084*, o *ISO Technical Committee n. 211 Geographic Information* e *ISO 15046* [4].

Na Figura 3 é apresentado um modelo UML duma hipotética Faculdade relacionando os dados de vários departamentos, usando o processo de modelação de dados desta abordagem esboçado na continuação desta secção.

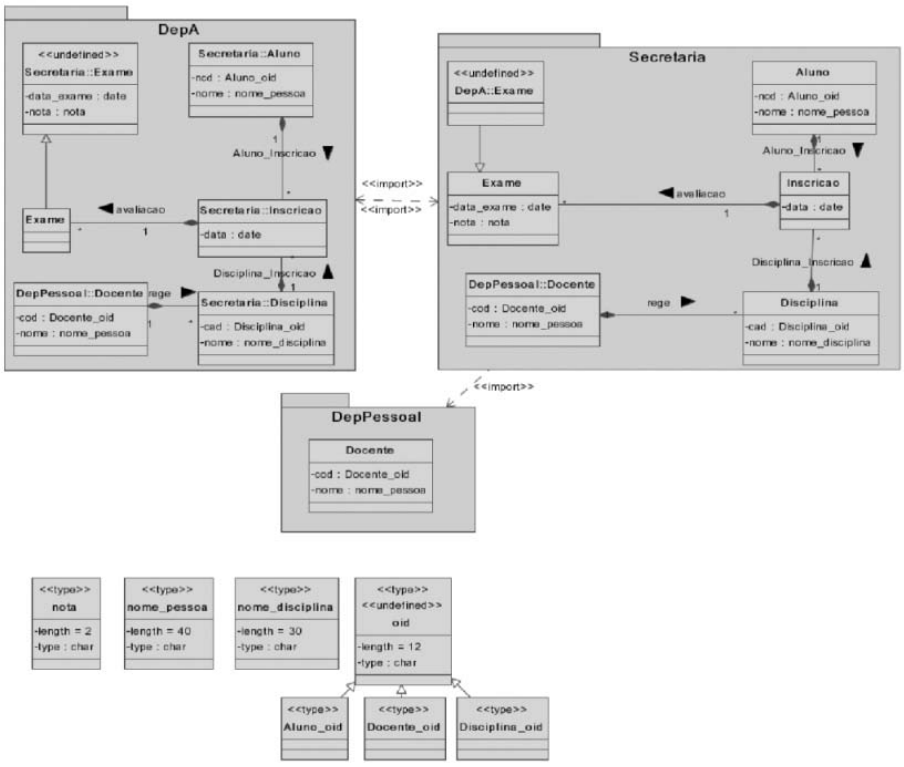


Figura 3. Modelo de dados em UML.

2.1 Restrições

A linguagem UML define diferentes tipos de modelos, cada qual adaptado a uma faceta dum sistema informático: alguns dos modelos focam características

estáticas da estrutura do sistema, outros características dinâmicas do seu funcionamento; uns modelos focam a estrutura do *software* utilizado pelo sistema, outros a estrutura do *hardware*. Neste trabalho usamos exclusivamente o **modelo de classes** por ser o mais adequado à representação duma estrutura do dados.

Este modelo é constituído por um conjunto de diagramas de classes nos quais podem ser usados vários componentes do UML. No entanto, para efeitos desta abordagem foram usados apenas alguns desses componentes que são referidos nos parágrafos seguintes.

- Os **pacotes** (*packages*) são usados para representar organismos duma organização. A divisão dum diagrama de classes em pacotes facilita a atribuição de responsabilidades e permissões. A gestão dos objectos das classes definidas num pacote é da responsabilidade do organismo com o mesmo nome do pacote. Os pacotes são denotados graficamente por rectângulos com uma aba (como uma pasta) e podem conter fragmentos de diagramas, neste caso modelando os dados dum organismo.
- As **classes** são usadas para representar as entidades básicas do sistema de informação. Cada classe é definida por um nome, um conjunto de atributos e de operações. Nesta abordagem as operações são omitidas e os nomes são geralmente precedidos dum prefixo identificador do pacote (organização) responsável pela informacão, seguindo as normas do UML para nomes compostos (`pacote::classe`). As classes são denotadas por rectângulos com três áreas empilhadas nas quais são representadas: nome e estereótipos; atributos; operações (vazios nos nossos modelos).
- As **dependências** são usadas para representar permissões de importação de informação entre organismos (pacotes). As dependências são denotadas por setas tracejadas com pontas abertas.
- As **generalizações** são usadas primordialmente para relacionar classes com o mesmo nome em pacotes (organismos) diferentes. A classe mais geral agrega informação proveniente de cada um dos organismos, enquanto as especializações são apenas uma parte desse conjunto. As generalizações são denotadas por setas terminadas por um triângulo fechado.
- As **associações** relacionam classes entre o mesmo pacote (organismo) e podem ser divididas em **composições** e **agregações**, cada qual com um semântica visual distinta. As composições criam uma relação em que os elementos não persistem no grupo, sendo destruidos quando o grupo termina. Nas agregações os elementos persistem à destruição do grupo. Das várias decorações permitidas pelo UML para as associações utilizamos apenas a **cardinalidade** em 5 valores pré-definidos: 0, 1, *, 0..1, 1..*. Para simplificar a construção de estruturas de dados a partir do XMI não é suportado nenhuma cardinalidade muitos-para-muitos. No entanto o mesmo efeito pode ser conseguido separando o relacionamento das duas classes por meio duma terceira com associações anteriores.

A Figura 3 ilustrada a utilização dos componentes anteriormente referidos na definição dum modelo de dados.

2.2 Extensões

A linguagem UML define mecanismos de extensão de forma a permitir modelar aspectos não cobertos pela definição básica. Esses mecanismos são os **estereótipos**, as restrições e os valores marcados. No presente trabalho fizemos uso dos primeiros para definir: novos tipos de dados, a autorização da importação de dados, e o formato dos identificadores dos objectos.

A definicão de tipos de dados é feita recorrendo a uma classe com o estereótipo `type` definida directamente no pacote raíz (isto é, não incluída em nenhum pacote associado a um organismo). As classes com estas características são processadas de forma específica na conversão do XMI para Schema, dando origem novos tipos. Na definição de tipos de dados enumerados surgiu a necessidade de representar os elementos pertencentes ao tipo e usamos uma classe com estereótipo `enumeration` em que o nome de cada atributo representa um valor admissível da enumeração. A Figura 3 apresenta exemplos da definição destas classes.

Como referido anteriormente, são usadas dependências para para indicar o permissão de importação. Para precisar este papel das dependências usamos o estereótipo `import`.

Para processar objectos é conveniente existir uma forma de os identificar. Existem diversas formas para fazer essa identificação mas a mais usual é a utilização de uma propriedade que caracterize univocamente o objecto. A definição do formato deste identificador fica em aberto neste projecto. É apenas requerido que o formato das referências seja um tipo de dados seguindo as regras referidas anteriormente, com a restrição do nome do tipo de dados ter como sufixo `_oid`.

3 Formatos de dados

O ponto central desta abordagem é a definição de formatos comuns para os documentos usados para a partilha de dados. Estes formatos permitem validar os dados gerados e/ou recebidos por cada uma das organizações e são essenciais para a implementação dos serviços de dados. No entanto estes formatos podem também ser usados para validar dados transferidos em ficheiros por qualquer forma. A definição destes formatos pode ser obtida automaticamente a partir da definição dos modelos definidos na etapa anterior, mapeando o documento XMI num conjunto de documentos XML Schema por aplicação duma folha de estilos.

A definição dos formatos de dados é feita usando o XML Schema [6]. Esta linguagem permite definir a estrutura, conteúdo e semântica de uma linguagem XML e existem várias ferramentas para validar um documento relativamente a um esquema. O XML Schema substitui com vantagem as formas tradicionais de definição de linguagem de anotação usando *Document Type Definitions* (DTD), especialmente na definição linguagens que codificam dados contidos em sistemas de informação [9].

A transformação dos documentos XMI em XML Schema recorre a folhas de estilo XSL. A família de recomendações XSL, a sigla de *eXtensible Style Language* [14], tem por objectivo a definição de transformações e apresentação de

documentos XML e é constituída por três partes: o XSLT, transformações de XML; o XPath, linguagem de expressões usada pelo XSLT (e outras especificações XML); e o XSL-FO, um vocabulário para especificação de semânticas de apresentação. A folha de estilos definida para esta abordagem recorre apenas ao XSLT e é independente do documento XMI utilizado.

UML (estereotipo)	XML Schema	Observações
Classe	Tipo complexo.	
Classe (`type`)	Tipo complexo	Novo tipo de dados que é usado por outros elementos. Contem outros elementos como propriedades restritivas.
Classe (`enumeration`)	Os elementos da enumeração	Elementos associados ao respectivo tipo de dados.
Atributo	Elemento	Fica associado ao tipo complexo da respectiva classe.
Atributo (`_oid`)	Elemento único	Elemento chave é definido como tendo valor único
Atributo (`required`)	Elemento	Como o anterior mas com atributos obrigatórios.
Associação	Elemento	O elemento é criado no tipo associado à classe com participação maior do que 1. Se for uma associação do tipo um-para-um, o elemento é criado na primeira extremidade da associação (esta extremidade é o ponto de partida na criação da associação).
Agregações/ Composições	Elemento	São tipos particulares de associações que obrigam à existência de uma instância da classe que é referenciada.
Generalização	Elemento	O tipo criado para a classe especializada herda os elementos associados aos atributos da classe mais geral, assim como possíveis referências a outras classes.

Figura 4. Mapeamento XMI/ XML Schema

As regras da tabela da Figura 4 são uma descrição sumária das regras implementadas em XSLT, com indicação das componentes do seu domínio e definições resultantes da sua aplicação. De notar que este mapeamento não faz uso dos mecanismos de identificação e referenciação disponíveis no XML Schema. De facto é possível num esquema declarar um atributo como chave, como também é possível declarar um atributo como uma referência a uma chave. Estas declarações são usado na validação de referências. Infelizmente no caso da partilha de dados a validação de referencias iria requerer que documento contivesse todos os objectos da base de dados. Por este motivo optou-se por não usar validação de referências neste mapeamento. Alternativamente é usado o mecanismo de vali-

dação de unicidade de valores associado aos elementos declarados como chave no modelo de dados.

Para compreender como as regras anteriormente descritas se interrelacionam é necessário ter noção da organização das definições de dados no contexto do documento. A estrutura destes documentos está dividida em três partes, definindo respectivamente o:

- o elemento que incorpora objectos
- os tipos complexos relativos às classes
- os elementos relativos aos tipos de dados

A partir do modelo de dados UML definem-se tipos de dados, que através do mapeamento dum documento XMI para XML Schema resultam definições de tipos complexos. Isto não é o suficiente, pois têm de existir elementos na definição de dados que usem esses tipos e para que possam criar instâncias de documentos XML segundo essa definição de dados. A solução foi definir um elemento, como o nome `SetDataObject`, que pode conter um conjunto qualquer de elementos correspondentes a objectos do modelo. Estes elementos referem tipos complexos com o mesmo nome que definem os elementos.

Depois são definidos os tipos complexos referentes às classes do modelo de dados que incorporam elementos correspondentes aos atributos. Se uma classe não tiver definido uma propriedade identificadora, é criado um atributo para assumir esse papel, com o mesmo nome é o mesmo da classe seguido de `_oid`.

A Figura 3 apresenta um excerto de um esquema e um documento válido na linguagem que o primeiro define. Este exemplo evidencia que a existência dos elementos `Aluno` e `Disciplina` se deve ao facto de fazerem parte do elemento `SetDataObject`, já que o processo de mapeamento apenas cria tipos complexos e elementos incorporados e não elementos de topo.

4 Serviços de dados

O nosso objectivo final é a definição dos serviços de dados (web services) que, ligados aos sistemas de informação de cada organismo, servirão de interface aos programas-cliente dos demais organismos. Como foi referido da secção 1, a implementação em concreto de cada serviço web está fora do âmbito deste trabalho. Nesta fase procuramos apenas definir o serviço web de cada organismo por forma a assegurar que corresponde aos requisitos da organização e se interliga harmoniosamente com os demais organismos.

Os serviços de dados usam o protocolo SOAP de transferência de informação SOAP [8]. Este protocolo define um tipo especial de documento XML, um envelope, que contem os documentos a transferir entre os entre serviços de dados e os seus clientes. A implementação dum serviço de dados usando este protocolo é habitualmente feita recorrendo a um motor SOAP, isto é uma estrutura de desenvolvimento de processadores SOAP, sejam estes clientes, servidores ou agentes intermediários.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:data="http://www.dataexchangeservice.com/Data-types"
   elementFormDefault="qualified"
 attributeFormDefault="unqualified"
      targetNamespace="http://www.dataexchangeservice.com/Data-types">

   <xsd:element name="SetDataObject" type="data:SetDataObject"/>
   <xsd:complexType name="SetDataObject">
      <xsd:choice maxOccurs="unbounded">
         <xsd:element name="Disciplina" type="data:Disciplina"
                      maxOccurs="unbounded"/>
         <xsd:element name="Aluno" type="data:Aluno"
                      maxOccurs="unbounded"/>
         <xsd:element name="Exame" type="data:Exame"
                      maxOccurs="unbounded"/>
         <xsd:element name="Inscricao" type="data:Inscricao"
                      maxOccurs="unbounded"/>
         <xsd:element name="Docente" type="data:Docente"
                      maxOccurs="unbounded"/>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="Disciplina">
      <xsd:all>
         <xsd:element name="cad" type="data:Disciplina_oid"
                      minOccurs="0" maxOccurs="1"/>
         <xsd:element name="nome" type="data:nome_disciplina"
                      minOccurs="0" maxOccurs="1"/>
      </xsd:all>
   </xsd:complexType>
   <xsd:complexType name="Aluno">
      <xsd:all>
         <xsd:element name="ncd" type="data:Aluno_oid"
                      minOccurs="0" maxOccurs="1"/>
         <xsd:element name="nome" type="data:nome_pessoa"
                      minOccurs="0" maxOccurs="1"/>
      </xsd:all>
   </xsd:complexType>


<?xml version="1.0" encoding="utf-8"?> <SetDataObject>
   <Aluno>
      <ncd>960307001</ncd>  <nome>Manuel António</nome>
   </Aluno>
   <Disciplina>
      <cad>1996071101</cad> <nome>Programação de Computadores</nome>
   </Disciplina>
</SetDataObject>
```

Figura 5. Schema obtido por transformação e instância

A definição de serviços de dados de cada organismos é feita recorrendo a documentos XML em WSDL, a sigla de *Web Service Description Language* [10]. Esta linguagem é uma proposta da Ariba, IBM e Microsoft e ainda não foi aceite como um *standard* pelo W3C. No entanto esta linguagem é já suportada por um conjunto apreciável de ferramentas de apoio ao desenvolvimento de serviços web.

O documento WSDL pode ser usado de várias formas, dependendo das capacidades do motor SOAP em questão. Certos motores, como o Apache Axis [2], permitem a utilização de ferramentas para criação de esqueletos de classes Java (como o `WSDL2Java`) quer na implementação do serviço, quer na implementação dos clientes. Em alguns motores SOAP poderá ser apenas possível gerar o documento WSDL a partir do serviço instalado. Neste caso o documento WSDL produzido nesta abordagem servirá para aferir se o serviço implementado corresponde à especificação pretendida. No âmbito deste trabalho não foi abordada a comparação automática de documentos WSDL para este efeito.

Um documento WSDL está estruturado em cinco secções

Tipo de elementos: contem definições em XML Schema referentes a elementos usados para descrever os dados trocados nas mensagens. Nesta abordagem esta secção contem elementos <`import`> que permitem carregar as definições de tipos produzidas na etapa anterior, bem como um conjunto de elementos que representam meta informação.

Mensagens: Define a estrutura das mensagens usadas nas operações do serviço. Cada operação possui uma mensagem de pedido e outra de resposta. Os tipos de dados de cada uma das operações podem ser tipos simples do XML Schema ou referir tipos definidos na secção anterior. Nesta abordagem foram definidas duas mensagens: `Request` que define um pedido de informação e a `Reply` identificando o tipo de dados da resposta.

Tipos de portas: Enumera as operações (funções) do serviço, identificando as mensagens que a compõem. As operações podem ser do tipo: um-sentido, pedido-resposta, solicita-resposta e notificação. Nesta abordagem foram usadas apenas operações do tipo pedido-resposta, significando que para obter dados um organismo terá de efectua um pedido.

Associações: Esta secção define a forma como as mensagens do serviço são implementadas pelo protocolo de comunicação, neste caso o SOAP.

Serviços: Enumera os serviço disponíveis. Cada serviço agrupa um conjunto de portas. Uma porta é um simples ponto definido pela combinação de uma associação e de um endereço. O endereço depende do protocolo de comunicação definido na respectiva associação.

4.1 Meta-informação

Na definição WSDL do serviço são importadas duas definições de dados: a obtida a partir do modelo de dados e meta-informação. A importação da primeira definição tem como objectivo definir os dados comunicados no corpo das mensagens SOAP e que são (des)serializados de/para o sistema de SI. A segunda define a meta-informação que descreve estes dados.

Com a meta-informação pretendemos caracterizar o tipo de mensagem (pedido ou resposta). No caso dum pedido a meta informação caracteriza o objecto pedido Tirando partido da estrutura das mensagens SOAP, a meta-informação é enviada no cabeçalho do documento, para caracterizar as mensagens, é descrito na tabela da Figura 6. O conjunto de elementos da meta-informação depende do tipo de documento (pedido ou resposta).

Elemento	Pedido	Resposta	Descrição
`Timestamp`	sim	sim	Informação relativa à data e hora da mensagem. É informação obrigatória.
`Source`	sim	sim	Informação relativa à origem da mensagem. Não existe uma restrição em relação a esta informação. Serve apenas para indicar a fonte de informação. É informação obrigatória.
`ID`	sim	sim	A mensagem tem de ter um identificador. Isto permite identificar de forma única cada mensagem enviada de qualquer local. Mais uma vez não existe uma restrição em relação ao identificador propriamente dito. É informação obrigatória.
`SinceDate`	sim	não	Isto é informação opcional.Permite restringir a obtenção de informação relativamente a uma data.
`RequestID`	não	sim	As mensagens de resposta têm que indicar o pedido correspondente.

Figura 6. Informação do cabeçalho das mensagens SOAP

As mensagens pedidos permitem identificar as classes de objectos que se pretende obter informação através dos elementos apresentados na tabela da Figura 7. O elemento `Object` pode ser utilizado de forma isolada, enquanto que o `Query` tem que ser usado em conjunto com o `Object`.

Elemento	Descrição
Objecto	Identifica a classe de objectos que se pretende obter.
Query	Filtro a aplicar sobre o conjunto de objectos da classe pretendida, para obter apenas um sub-conjunto desses objectos. Este filtro é aplicado sobre um documento com todos os objectos da classe pretendida e poderá ser uma expressão XPath ou uma questão nas linguagens XQuery [12] ou XQueryX [13]

Figura 7. Elementos que identificam os objectos

O documento 8 apresenta a definição correspondente à meta-informação enviada no cabeçalho das mensagens SOAP dos pedidos e das respostas.

5 Conclusão

Os serviços web são uma abordagem cada vez mais usada na comunicação de dados entre sistemas heterogéneos devido à independência dos formatos de dados

```
<?xml version="1.0"?>
<xsd:schema
     xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.fc.up.pt/Meta"
            xmlns:meta="http://www.fc.up.pt/Meta"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">

        <xsd:complexType name="Request">
           <xsd:sequence>
              <xsd:element name="Object" type="xsd:string"
                           minOccurs="1" maxOccurs="1"/>
              <xsd:element name="Query" type="xsd:string"
                           maxOccurs="1"/>
           </xsd:sequence>
           <xsd:attribute name="timestamp" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="source" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="ID" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="sinceDate" type="xsd:string"
                          use="optional"/>
        </xsd:complexType>

        <xsd:complexType name="Reply">
           <xsd:attribute name="timestamp" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="source" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="ID" type="xsd:string"
                          use="required"/>
           <xsd:attribute name="RequestID" type="xsd:string"
                          use="required"/>
        </xsd:complexType>
</xsd:schema>
```

Figura 8. Defnção da meta-informação das mensagens.

(baseados em XML) e à ubiquidade dos seus protocolos de transporte (como o HTTP e SMTP). Por estas razões os serviços web começam também a ser usados na partilha de dados entre organismos de uma mesma organização.

Neste artigo descrevemos uma abordagem baseada em *standards* XML para definir, coordenar e controlar centralmente a partilha de dados entre vários organismos duma mesma organização, focando tarefas como a modelação dos dados globais, as responsabilidades e direitos dos organismos sobre os dados, a normalização de formatos de transferência e a definição dos serviços de dados. Dos pontos positivos conseguidos com esta abordagem destacamos:

- a facilidade de modelação dos dados recorrendo a ferramentas UML;
- eficácia do processo de mapeamento da modelação em definições de dados;
- simplicidade na implementação e dos serviços e programas clientes, usando ferramentas baseadas no processamento de documentos WSDL;

Alguns dos objectivos iniciais ainda não foram totalmente resolvidos duma forma automática. Em particular, as responsabilidades e direitos dos organismos sobre os dados, definidos em UML e representados nos documentos XMI ainda não são usados para configurar os serviços de dados. Estes requisitos poderão ser controlados usando mecanismos de autenticação e segurança baseados no WS-Security [1].

Referências

1. Kapil Apshankar. Ws-security security for web services, 2002. `http://www.webservicearchitect.com`.
2. Webservices - axis. The Apache XML Project `http://ws.apache.org/axis/`.
3. S. Brodky. Xmi opens applications interchange. Technical report, IBM Articles, July 1999. `http://www.ibm.com`.
4. Disgis project. `http://www.ec-gis.org/disgis.htm`.
5. Cd 15046-9 geographic information - part 9: Rules for application schema. Norwegian Technology Standards Institution, `http://geomatics.ncl.ac.uk/research/projects/iso/Rules.pdf`.
6. Xml schema. World Wide Web Consortium — `http://www.w3.org/XML/Schema`.
7. David Skogan. Uml as a schema language for xml based data interchange, 1999. `http://heim.ifi.uio.no/~davids/papers/Uml2Xml.pdf`.
8. Soap: Simple object access protocol. World Wide Web Consortium `http://www.w3.org/`.
9. Kevin Williams. Why xml schema beats dtds hands-down for data, June 2001. `http://www.ibm.com`.
10. Wsdl: Web services description language. World Wide Web Consortium `http://www.w3.org/TR/wsdl`.
11. Xmi: Xml metadata interchange. Object Management Group `http://www.omg.org`.
12. Xml syntax for xquery 1.0 (xquery). World Wide Web Consortium< `http://www.w3.org/TR/xquery/`.
13. Xml syntax for xquery 1.0 (xqueryx). World Wide Web Consortium< `http://www.w3.org/TR/xqueryx/`.
14. Xsl: The extensible stylesheet language family. World Wide Web Consortium `http://www.w3.org/Style/XSL`.

Memórias de Tradução Distribuídas

Alberto Manuel Simões, José João Almeida, and Xavier Gomez Guinovart

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

Universidade de Vigo
xgg@uvigo.es

Resumo Neste documento apresenta-se o conceito de memórias de tradução distribuídas, discutindo-se o seu interesse na área da tradução, bem como as vantagens que uma ferramenta de tradução pode tirar do seu uso.
É apresentada uma possível implementação de memórias de tradução distribuídas usando WebServices numa arquitectura de cooperativismo. São definidos as mensagens (API) que um serviço deste género deve implementar para que uma ferramenta de tradução possa tirar partido da colaboração entre tradutores.

1 Introdução

Na área da tradução assistida por computador usa-se memórias de tradução (MT): correspondências de frases entre duas ou mais línguas diferentes.

$$TMX \equiv \mathcal{S}_{\mathcal{L}_\alpha} \rightharpoonup \mathcal{S}_{\mathcal{L}_\beta} \times \ldots \times \mathcal{S}_{\mathcal{L}_\omega}$$

Estas memórias são usadas pelos tradutores como bases de dados onde traduções já efectuadas são armazenadas para que futuras traduções, semelhantes a outras já realizadas possam ser reutilizadas.

Para armazenar as MT cada aplicação usa o seu formato proprietário. No entanto existe um *standard* baseado em XML[8] para o intercâmbio das MT: o *Translation Memory eXchange* — TMX[7,6,4].

1.1 O Formato TMX

Uma memória de tradução TMX segue um DTD que define dois grupos distintos: cabeçalho (`header`) e o corpo (`body`).

O cabeçalho guarda meta-informação: autor, data de criação, ferramenta que o gerou e outras propriedades, todas como atributos do elemento. O atributo mais importante presente no cabeçalho é o "`srclang`" que define qual a língua original na criação da memória de tradução (semanticamente, este campo é usado para indicar qual a língua original, da qual todas as outras foram traduzidas).

O corpo do documento TMX é composto por uma sequência de unidades de tradução que correspondem a:

```
<tu>
  <tuv xml:lang="en">
    <seg>Configure window properties</seg>
  </tuv>
  <tuv xml:lang="pt">
    <seg>Configurar propriedades das janelas</seg>
  </tuv>
</tu>
```

Cada unidade de tradução (`tu`) pode conter texto em mais do que uma língua. Cada um destes textos é colocado num elemento "`seg`" dentro de um outro denominado "`tuv`". Este é identificado obrigatoriamente pelo nome de língua, com o atributo "`xml:lang`".

Embora o formato TMX suporte um conjunto mais alargado de etiquetas XML, não é importante a sua apresentação neste documento já que não são usadas para a implementação das MT distribuídas.

1.2 Conceito de MT distribuída

Para apresentar de forma mais clara o uso e implicações das MT distribuídas, consideremos duas situações:

- enquanto trabalha, o tradutor vai construindo a sua MT. No entanto, existem empresas que fornecem as suas memórias de tradução especializadas em determinada área (por exemplo, certas indústrias automóveis) e outras que as vendem (por exemplo, as empresas de software de tradução). Quando a MT é fornecida por um terceiro, esta é enviada (quer seja por correio, e-mail, ftp, etc) para o tradutor, que passa a ser seu dono[1].
- outra situação, é um grupo de tradutores que trabalham num gabinete de tradução ou numa comunidade em que mais do que um tradutor está envolvido no mesmo projecto. Nesta situação existe grande probabilidade de traduzirem porções semelhantes, não se reutilizando trabalho.
  Um bom exemplo deste tipo de comunidade é a documentação de software open-source em que cada projecto é traduzido por pessoas diferentes e que acabam por traduzir textos idênticos.

Estas duas situações podem beneficiar da existência de mecanismos de MT distribuídas[1]:

- cada fornecedor coloca as suas MT acessíveis usando uma tecnologia de partilha de dados remota, e estas passam a estar disponíveis livremente ou sujeitas a um "subscrição" ou "aluguer";
- numa comunidade de tradução torna-se possível uma melhor reutilização do trabalho realizado por cada tradutor;

[1] Embora em certos casos existam limitações no seu uso — apenas em alguns projectos, durante algum tempo, etc.

A secção seguinte explica o conceito de memória de tradução distribuída, quais os impactos do seu uso nas ferramentas de tradução e como a arquitectura de rede deve estar organizada. Termina com a definição das mensagens que um WebService para MT distribuídas deve implementar.

Na secção 3 detalha-se a implementação de um cliente e servidor de MT distribuídas. A secção seguinte termina com as conclusões que se podem retirar desta metodologia de trabalho.

2 MT distribuídas

Esta secção pretende definir a estrutura de um serviço de MT distribuídas e como deve este reagir aos diferentes tipos de pedidos que terá de responder.

2.1 Impactos na ferramenta de tradução

Quando usadas localmente, as memórias de tradução são especialmente úteis no momento da tradução: para cada frase ou segmento a traduzir, a ferramenta de apoio ao tradutor irá pesquisar a MT por esse segmento.

Este processo é realizado de forma diferente por cada ferramenta de tradução: pesquisa por um frase completa, por porções delimitadas por *markup*, ou por inclusão parcial. O processo de *matching* pode ser ainda mais complicado, como iremos ver na secção 3.2.

Num ambiente de MT distribuídas, o processo irá ser diferente. Além de consultar a sua memória de tradução local, a ferramenta irá consultar um conjunto de servidores de MT. Esta paralelização irá produzir respostas concorrentes das quais se terá de escolher uma, ou de as conciliar.

Ao optarmos pela escolha de uma resposta apenas, a ferramenta pode usar várias heurísticas, das quais salientamos:

- escolher a resposta mais rápida — se a MT local responder, a ferramenta poderá mesmo não chegar a consultar qualquer servidor de MT;
- dada uma associações de classificação ou *ranking* de servidores de MT distribuídas, escolher a resposta do servidor mais cotado;
- escolher de acordo com uma classificação da tradução — implica que cada servidor etiquete a sua tradução com uma medida de qualidade. Esta opção obrigará à adopção pelos servidores de MT de um método comum de classificação da tradução.

2.2 A arquitectura de rede

De um ponto de vista macroscópico, a arquitectura de rede para um sistema de MT distribuídas pode ser construída de duas formas diferentes, como apresentado na figura 1, correspondentes às duas situações apresentadas na introdução.

Para cada um dos casos de estudo, estamos a utilizar uma arquitectura diferente: no primeiro caso baseada em *cliente/servidor* (C/S) e no segundo caso, baseada em *peer-to-peer* (P2P):

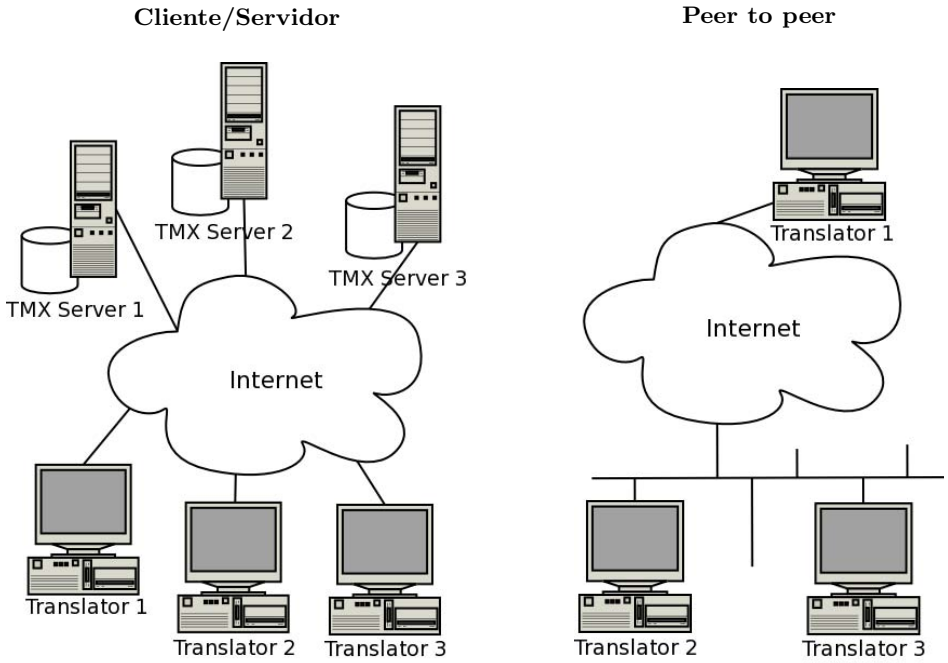


Figura 1. Arquitecturas de rede

– a arquitectura P2P é especialmente útil numa comunidade de tradução, onde vários tradutores estão a trabalhar e a partilhar automaticamente as traduções que estão a realizar;
  Pata este tipo de arquitectura podíamos dizer que uma tecnologia baseada em WebServices não é das mais adequadas já que obrigaria que cada tradutor tivesse um servidor web na sua máquina de trabalho. No entanto, não é completamente desprovido de senso a criação de um servidor dedicado apenas para a disponibilização de memórias de tradução. Neste artigo vamos supor que a arquitectura P2P é uma rede de várias máquinas, em que cada máquina é cliente e servidor ao mesmo tempo.
  Outras alternativas podiam ser baseadas em:
  - uso de uma tecnologia proprietária;
  - associação a um dos membros da rede P2P uma lista de clientes, que cada membro faz download e comunica usando, por exemplo, sockets;
  - criação de um servidor local para submissão de memórias de tradução e posterior divulgação (deixamos de ter arquitectura P2P real passando a C/S);
– Numa arquitectura C/S, será necessário um servidor dedicado para disponibilizar as unidades de tradução. Este será o método preferencial para a disponibilização de MT por terceiros.

Enquanto que na arquitectura P2P uma solução baseada em WebServices não é adequada, para uma arquitectura C/S o sistema pode ser implementado com uma qualquer tecnologia baseada em "remote procedure call", quer seja Sun RPC, Corba, Java RMI ou mesmo Web Services.

2.3 Definição do WebService

Esta secção pretende definir quais os métodos que o servidor de MT deve saber responder para a implementação de um serviço de MT distribuídas.

É usual especificar as mensagem inerentes ao processo usando a WebService Description Language (WSDL)[3]. No entanto pareceu-nos mais importante reflectir sobre a funcionalidade do WebService usando uma notação matemática do que a inclusão do documento WSDL que iria, sem dúvida, aumentar o número de páginas do artigo.

A definição deste conjunto de mensagens teve em conta o interesse de termos um servidor *stateless* — ou seja, o servidor não guarda qualquer informação sobre os vários clientes.

Num caso de subscrição de serviço em que seja necessária a autenticação do cliente, deixaremos de ter um servidor *stateless*, já que terá de armazenar informação sobre cada cliente. Também se teriam de alterar as mensagens aqui definidas, não só porque passaríamos a precisar de uma para o *login*, mas também porque teríamos de enviar em cada mensagem um *token* que identificasse o cliente. Visto que nos parece mais correcta a filosofia *open*, não nos iremos preocupar com a autenticação do serviço.

Configuração do cliente

Uma ferramenta de tradução que pretenda usar de um servidor de MT, terá de saber, para cada servidor, se o deve ou não usar, visto que as línguas disponíveis no servidor podem não ser as que o tradutor necessita.

Desta forma, o cliente deve, para cada servidor, construir uma lista de pares de MT disponíveis. Essa lista deverá ir sendo actualizada à medida que novas traduções surgem.

A configuração do cliente poderá ser feita com o envio de uma mensagem a que chamaremos `traduzes?` com um par de línguas, ao que o servidor irá responder com um valor booleano:

$$traduzes? : \mathcal{L}_\alpha \times \mathcal{L}_\beta \longrightarrow 2$$

Alem do tipo booleano que existe pre-definido, existe também um tipo chamado `Language`, sub-classe de `String` e que irá ser usado para representar línguas.

Pedido de tradução

O pedido de tradução que será dirigido a cada servidor de MT irá conter o par de línguas (língua de origem e língua destino) e uma frase na língua de

origem. O servidor irá responder com uma possível tradução e uma medida de qualidade[2] ou, nenhuma resposta.

$$traduz : \mathcal{L}_\alpha \times \mathcal{L}_\beta \times \mathcal{S}_{\mathcal{L}_\alpha} \longrightarrow \mathcal{S}_{\mathcal{L}_\beta} \times \mathcal{Q} + 1$$

Uma outra opção poderia ser a de permitir que cada servidor de MT respondesse com mais do que uma possível tradução. No entanto, parece-nos que cada servidor de MT deverá ter métodos próprios para a escolha da melhor tradução da sua base de traduções.

Concordância

Outra aplicação comum das MT a que se chama *concordância*, é o processo de, dada uma palavra ou sequência de palavras, encontrar todos os pares em que essa sequência aparece.

$$concordancia : \mathcal{L}_\alpha \times \mathcal{L}_\beta \times \mathcal{S}q_{\mathcal{L}_\alpha} \longrightarrow \left(\mathcal{S}_{\mathcal{L}_\alpha} \times \mathcal{S}_{\mathcal{L}_\beta}\right)^\star + 1$$

Embora semelhante ao pedido de tradução, a concordância é realizada com sequências muito mais pequenas de palavras e o resultado não passa pelo processo de selecção ou conciliação já que todos[3] os pares são apresentados ao utilizador.

Contribuição

Num sistema *peer-to-peer* local, todas as comunicações são efectuadas por uma rede local sem grande tráfego. Para a partilha por *peer-to-peer* numa comunidade grande e dispersa pela Internet, as comunicações podem limitar a interactividade do software de tradução.

Para colmatar este problema pode haver interesse na criação de servidores de MT distribuídos pela Internet, ligados em *peer-to-peer* em que as MT vão sendo partilhadas, e para onde cada tradutor irá contribuindo as traduções já efectuadas.

Surge a necessidade de uma nova mensagem no WebService para a contribuição de memórias de tradução para um servidor:

$$contribui : \mathcal{L}_\alpha \times \mathcal{L}_\beta \times \mathcal{S}_{\mathcal{L}_\alpha} \times \mathcal{S}_{\mathcal{L}_\beta} \longrightarrow 2$$

Outra solução seria a contribuição usando dois textos paralelos (em que um é a tradução do outro) em que o servidor teria a necessidade de o segmentar

[2] Parece-nos importante que cada tradução devolvida pelo servidor contemple uma medida de qualidade. Tradicionalmente, esta medida calcula-se como a distância de edição entre a frase armazenada na memória de tradução e a frase solicitada (distância de edição definida polo NIST como "The smallest number of insertions, deletions, and substitutions required to change one string or tree into another"(`http://www.nist.gov/dads/HTML/editdistance.html`). No entanto, o problema complica-se devido à influência da ordem das palavras e do formato do texto, entre outros factores[2]

[3] Por vezes o número de pares é demasiado grande, pelo que este serviço deve limitar o número de pares retornados.

e alinhar à frase antes de o tornar disponível. Embora mais demorado, este tipo de contribuição pode ser importante para o reaproveitamento de traduções realizadas sem o uso de ferramentas de tradução com suporte para MT.

Embora este processo de contribuição seja importante não só para a distribuição de carga mas também para equilibrar o modelo, não nos vamos debruçar sobre ela já que implicaria a definição de políticas de contribuição (contribuição totalmente aberta, com revisão, com autenticação, etc).

3 Implementação

Esta secção irá apresentar a implementação de um protótipo do conceito de MT distribuídas.

O cliente e servidor foram implementados usando Perl e SOAP (`SOAP::Lite`). Em relação a pormenores de implementação, parece-nos mais importante a parte do servidor do que a do cliente, já que esta última limita-se ao envio e recepção de mensagens, tendo somente que conciliar as respostas recebidas.

3.1 Cliente

A implementação de um cliente deveria ser feita como *plug-in* para um sistema de tradução. No entanto, quase todos os sistemas de tradução são comerciais o que não nos permite a sua alteração. Embora existam alguns *open-source* (Frankenstein[4], OmegaT[5], ForeignDesk[6]) optamos por criar apenas um protótipo de interface com o serviço de MT distribuídas, e mais tarde tentar contribuir com um destes projectos.

O código apresentado de seguida mostra o quão simples é a interface a um WebService usando Perl e `SOAP::Lite`. Na verdade, este código iria ser muito semelhante para qualquer outro tipo de sistema Cliente/Servidor.

```
1  use SOAP::Lite;

2  my $soapserver = SOAP::Lite
3    -> uri('http://localhost:80/disttmx')
4    -> proxy('http://localhost:80/cgi-bin/disttmx.cgi');

5  my $soapresult = $soapserver->traduz("pt","en","era uma vez...");

6  unless($soapresult->fault) {
7    my $result = $soapresult->result();

8    if ($result) {
9      print "$result\n";
10   } else {
```

[4] http://www.sourceforge.net/projects/frankenstein/
[5] http://www.sourceforge.net/projects/omegat/
[6] http://www.sourceforge.net/projects/foreigndesk/

```
11      print STDERR "** server does not know how to translate it **\n";
12    }
13  } else {
14    print $soapresult->faultcode,": ",soapresult->faultstring;
15  }
```

3.2 Servidor

Embora o formato de intercâmbio de MT seja o TMX, sendo este um ficheiro de texto, estruturado mas sem limites físicos bem delimitados, torna-se dificil a implementação de uma pesquisa eficiente[7].

Uma solução passa pelo uso de algum tipo de indexador da informação. Com esse objectivo usamos o *glimpse*[5], um indexador de ficheiros com pesquisas bastante eficientes. No entanto, este indexador não sabe o que o XML é, pelo que não é fácil manusear directamente a memória de tradução.

Para resolver este problema cria-se um conjunto de ficheiros, m por cada língua, em que cada linha do ficheiro corresponde a uma memória de tradução (ou seja, a linha n do ficheiro m corresponde à linha n do ficheiro $\mathcal{T}(m)$).

O *glimpse* permite a pesquisa directa por um padrão ou por uma palavra, pelo que a implementação do servidor de memórias de tradução distribuída limita-se a receber o pedido, desempacotar o *query*, executá-lo usando o *glimpse*, procurar a tradução respectiva no ficheiro da língua de destino, empacotar a tradução e responder ao cliente.

Este processo não é tão eficiente quanto desejariamos já que cada execução do *glimpse* obriga-o a carregar todos os índices de consulta. Dado que a distribuição do *glimpse* incluí um servidor que carrega os índices apenas uma vez e responde a pedidos efectuados.

Usando este servidor temos uma arquitectura a dois níveis: um servidor de MT que recebe o pedido e o encaminha no formato correcto para o servidor *glimpse*, que enviará a resposta ao servidor de MT que a empacotará e a enviará ao cliente. Este processo pode ser visto na figura 2.

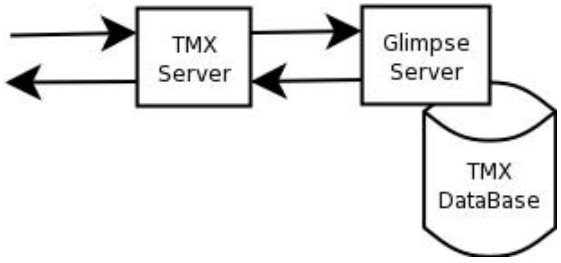


Figura 2. Arquitectura do Servidor usando o Glimpse

[7] Uma memória de tradução chega muito facilmente aos 100 Megabytes.

Embora esta implementação consiga ser eficiente, não permite o uso de algoritmos de *matching* mais complicados.

A implementação de um servidor de WebServices em Perl e `SOAP::Lite` é tão simples quanto a escrita do cliente. A CGI de tratamento de pedidos pode simplesmente invocar um módulo Perl dedicado à tarefa do servidor de forma completamente transparente:

```
1 use SOAP::Transport::HTTP;

2 SOAP::Transport::HTTP::CGI-> dispatch_to('disttmx')->handle;
```

Note-se que neste exemplo `disttmx` é o nome do módulo de gestão do servidor de MT distribuídas.

O módulo Perl que irá fazer a gestão do servidor de MT terá a seguinte estrutura:

```
1 package disttmx;
2 use strict;

3 sub traduzes {
4     my ($self, $lang1, $lang2) = @_;
5     ...}

6 sub traduz {
7     my ($self, $lang1, $lang2, $frase) = @_;
8     ...}
```

4 Conclusões

O conceito de MT distribuída pode ser importante de forma a criar utilização/construção cooperativa de recursos linguísticos. Embora não seja a nossa postura, as MT distribuídas também podem constituir uma oportunidade de negócio.

A implementação de MT distribuídas usando WebServices está ainda em fase de prototipagem, mas já demonstrou ser uma tecnologia viável. No entanto, sem a cooperação das empresas de ferramentas de tradução para a implementação desta filosofia, a construção de servidores de MT distribuídas não terá, por si só, grande repercussão no mundo da tradução.

Torna-se importante a construção de ferramentas eficientes para a indexação das MT que permitam a pesquisa por aproximação, número de palavras semelhantes e outros métodos de *matching*.

Referências

1. Joseba Abaitua. Memorias de traducción en TMX compartidas por internet. *Revista Tradumàtica*, (0), octubre 2001. `http://www.fti.uab.es/tradumatica/revista/num0/articles/jabaitua/imprim%ir.pdf`.

2. Yasuhiro Akiba, Kenji Imamura, and Eiichiro Sumita. Using multiple edit distances to automatically rank machine translation output. In *Machine Translation Summit VIII*, pages 15–25, 2001. http://www.eamt.org/summitVIII/papers/akiba.pdf.
3. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1, 2001. http://www.w3.org/TR/wsdl/.
4. Josu Gómez. Una guía al TMX. *Revista Tradumàtica*, (0), octubre 2001. http://www.fti.uab.es/tradumatica/revista/num0/articles/jgomez/imprimir%.pdf.
5. Udi Manber and Sun Wu. Glimpse: A tool to search through entire filesystems. Winter USENIX Technical Conference, 1994.
6. OSCAR. Open Standards for Container/Content Allowing Re-use — TMX home page, 2003. http://www.lisa.org/tmx/.
7. Yves Savourel. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association, 1997.
8. *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/.

Parte III

Web Services e Informação Geográfica

M-GIS — Sistema Móvel Interoperável de Informação Geográfica

Jorge Cardoso[1], Artur Rocha[1], João Correia Lopes[2]

[1] INESC Porto, R. Dr. Roberto Frias, 4200-465 Porto.
http://www.inescporto.pt/
{jcsc,artur.rocha,jlopes}@inescporto.pt
[2] Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto.
http://www.fe.up.pt/∼jlopes/
jlopes@fe.up.pt

Resumo Este artigo descreve uma forma de acesso interoperável a um sistema de informação geográfica através de dispositivos móveis de características limitadas (e.g. telemóveis ou *Personal Digital Assistants*). O sistema M-GIS segue uma arquitectura cliente-servidor tendo por base informação geográfica no formato GML que é transformada, através de XSLT, para o formato gráfico SVG. A informação geográfica em GML é servida por um *Web Feature Server*, o que permite que seja acedida de uma forma normalizada, independentemente do seu formato ou localização física, desde que seja respeitada a conformidade com a especificação. A aplicação cliente foi desenvolvida usando a tecnologia *Java Mobile Information Device Profile*. Os resultados obtidos indicam que é viável desenvolver um sistema móvel de visualização de informação geográfica recorrendo a normas e formatos abertos, com algumas limitações. As principais limitações do sistema estão relacionadas com a quantidade de informação que o cliente consegue, nesta data, processar.

1 Introdução

A ubiquidade dos dispositivos móveis como telemóveis e *PDAs* (*Personal Digital Assistants*) tem levado a uma crescente oferta de serviços orientados para essas novas plataformas. Um tipo de serviço muito útil para os utilizadores desses dispositivos poderá ser um serviço baseado em informação geográfica. A possibilidade de um utilizador consultar o seu telemóvel para obter o mapa da zona onde se encontra, ou para procurar uma determinada rua ou edifício e visualizar graficamente a sua localização é uma mais valia óbvia.

Existem já alguns sistemas deste género disponíveis, tais como o ArcPad [4] da ESRI, o GeoGIS [6] da Geo InSight e o PocketGIS [17] da Pocket Systems. Todos estes sistemas foram desenvolvidos para *PDAs* (não funcionam em telemóveis) e são descritos na secção 2. O problema destes sistemas é a interoperabilidade com outros sistemas de informação geográfica, uma vez que, regra geral, se fica limitado ao uso de informação geográfica num determinado formato gerada

por sistemas de informação geográfica da mesma entidade. Uma vez que as bases de dados geográficos utilizadas são proprietárias, estes sistemas são, regra geral, incompatíveis com outros sistemas de informação. Isto significa que não podemos ter uma gestão descentralizada da informação geográfica utilizada pelo sistema M-GIS (a não ser que todos os centros utilizem a mesma tecnologia). Por outro lado, muitos destes sistemas são orientados para determinados dispositivos, i.e., apenas funcionam em *PDAs* com o sistema operativo *Pocket PC*, por exemplo.

O objectivo deste trabalho é fornecer uma arquitectura para sistemas móveis de informação geográfica independente de tecnologias proprietárias (nomeadamente, da tecnologia da base de dados geográfica) de forma a propiciar a sua potencial integração com diversos sistemas de informação geográfica.

O sistema *M-GIS — Mobile Geographic Information System* [1] utiliza informação geográfica em formato GML (*Geography Markup Language*) proveniente de um WFS (*Web Feature Server*) para a construção de mapas pesquisáveis em formato SVG (*Scallable Vector Graphics*). A programação do lado do cliente foi feita em J2ME (*Java 2 Micro Edition*), mais concretamente usando o perfil MID (*Mobile Information Device*).

Os resultados obtidos demonstram que existem limitações ao nível da quantidade de informação que os clientes móveis conseguem, nesta data, suportar.

O resto deste artigo está estruturado da seguinte forma: a secção 2 apresenta, de forma resumida, alguns sistemas de informação geográfica para dispositivos móveis; a secção 3 apresenta algumas das tecnologias mais relevantes para o desenvolvimento deste projecto; a secção 4 apresenta a arquitectura do projecto M-GIS; na secção 5 são apresentados alguns detalhes de implementação, assim como alguns testes realizados sobre o sistema e respectivos resultados; por fim, na secção 6, apresentamos algumas conclusões que se podem retirar do trabalho desenvolvido e apontamos o caminho para trabalho futuro.

2 GIS para dispositivos móveis

Nesta secção serão introduzidos, de forma resumida, alguns sistemas de informação geográfica para dispositivos móveis existentes.

ArcPad

O sistema ArcPad [4] é um produto comercial da ESRI [5] para *PDAs*. É um sistema de informação geográfico completo com capacidade para adicionar dados a partir da Internet através de tecnologia *wireless*; "navegar" nos mapas: aproximar e deslocar a vista, estabelecer *bookmarks* e centrar na posição GPS actual; e pesquisar os dados: localizar e identificar objectos geográficos. O ArcPad funciona em *PDAs* com sistema operativo Windows CE/Pocket PC. O sistema suporta dados geográficos no formato *shapefile*[3] e pode ser usado como cliente de um servidor ArcIMS.

[3] Os ficheiros do tipo *shapefile* são um formato de informação geográfica criado pela ESRI e aceites como *de facto standard* pela indústria dos SIG.

Uma vez que este sistema apenas foi desenvolvido para a plataforma Windows CE/Pocket PC e implica a utilização de bases de dados geográficas proprietárias da ESRI, não vai de encontro aos objectivos propostos para o nosso trabalho.

GeoGIS

O GeoGIS [6] é um sistema de informação geográfico para *PDAs* com o sistema operativo Palm OS. Esta aplicação utiliza dados geográficos convertidos de ficheiros *shapefiles* criados com o sistema ArcView da ESRI. Este sistema tem, entre outras, as seguintes funcionalidades: criação de projectos com temas (ou camadas de mapas) únicos a cada projecto; aproximar, afastar e deslocar a vista sobre o mapa; pesquisar o mapa; adicionar ou apagar objectos do tema activo.

O GeoGIS utiliza dados convertidos do formato *shapefile* da ESRI e apenas pode ser usado em dispositivos com o sistema operativo Palm OS. Para além disso, este sistema não usa uma arquitectura cliente-servidor, i.e., os dados geográficos têm de ser carregados manualmente para o dispositivo. Por estas razões, o sistema GeoGIS não satisfaz os requisitos definidos neste projecto.

PocketGIS

O PocketGIS [17] é um sistema para dispositivos com o sistema operativo Windows CE. O sistema é composto por uma aplicação que corre no dispositivo móvel e por uma aplicação de *desktop*, o PocketGIS Connection, que serve para transferir os dados entre o PocketGIS e o *desktop* numa variedade de formatos: NTF, *Shapefile*, MIF (MapInfo), DXF, CSV, Raster (BMP e TIFF). A aplicação PocketGIS tem as seguintes funcionalidades: identificação de objectos do mapa; medição de distâncias; modificação da geometria de um objecto; alteração dos atributos de objectos. Para além disso, um sistema de GPS pode ser integrado com o PocketGIS.

O sistema PocketGIS suporta dados de diversos formatos através de um utilitário de *desktop* que converte os dados de vários formatos para o formato utilizado pelo PocketGIS (provavelmente um formato proprietário). No entanto, este sistema implica o carregamento manual da informação geográfica para o dispositivo. Para além disso, apenas funciona em dispositivos com o sistema operativo Windows CE, por isso não cumpre os requisitos do projecto.

3 Tecnologias Relevantes

Nesta secção são apresentadas algumas das tecnologias mais relevantes para o desenvolvimento deste projecto.

3.1 *Geography Markup Language*

O GML [13] é uma especificação desenvolvida pelo consórcio internacional OpenGIS. O grande objectivo do GML é fornecer uma plataforma neutra e aberta para

a definição de objectos e esquemas geo-espaciais. O GML não é uma linguagem rígida mas antes um sistema que suporta a definição de linguagens de descrição geográfica. Posto muito simplesmente, o GML consiste num conjunto de esquemas (*schemas*) XML que podem ser estendidos de forma a se adaptarem a situações específicas, mas mantendo sempre uma base comum.

O GML está desenhado de forma a suportar a interoperabilidade e fá-lo através da definição de elementos geométricos básicos (todos os sistemas que suportam GML usam os mesmos elementos geométricos), de um modelo de dados comum e de um mecanismo para criação e partilha de esquemas (*schemas*) aplicacionais. A maior parte das comunidades de informação facilita a interoperabilidade dos seus sistemas publicando os esquemas GML definidos por si. Esta norma é assim fundamental para satisfazer os objectivos deste trabalho relacionados com a interoperabilidade e independência em relação a formatos proprietários

3.2 *Web Feature Services* e *Web Map Services*

Web Feature Services (WFS) e *Web Map Services* (WFS) são duas especificações do consórcio OpenGIS [16].

O WMS [15] especifica o serviço de *Web Map*, ou seja, especifica a interface *Web* de um sistema que produz mapas (representações visuais de informação geográfica). A especificação define os tipos de pedidos e respostas que um serviço deste género deve suportar. A especificação WMS define três operações: *GetCapabilities*, que retorna meta-informação sobre o serviço; *GetMap*, que retorna uma imagem cujo conteúdo e dimensão são definidos pelo cliente; *GetFeatureInfo* (opcional), que retorna informação sobre objectos particulares mostrados no mapa.

O WFS [14] especifica o serviço de *Web Features*, ou seja, define a interface de um sistema que produz informação geográfica no formato GML. Os pedidos ao WFS podem ser codificados de duas formas: em XML ou em pares parâmetro-valor. A especificação WFS define vários tipos de pedidos, e.g.,: *DescribeFeatureType*, gera um esquema com a descrição dos tipos de objectos servidos pela implementação do WFS; *GetFeature*, permite obter um conjunto de objectos geográficos no formato GML; *GetCapabilities*, gera um documento XML que define as "capacidades" do WFS, por exemplo, que camadas de informação estão disponíveis.

Estas duas especificações têm por objectivo a interoperabilidade de sistemas de informação geográfica, uma vez que permitem aceder a informação geográfica de uma forma normalizada, residente em qualquer sistema que esteja conforme com a especificação. Daí também o interesse que estas normas têm para o nosso projecto.

3.3 *Scalable Vector Graphics*

O SVG (*Scalable Vector Graphics*) é uma linguagem para descrever gráficos vectoriais bidimensionais em XML. O SVG permite três tipos de objectos gráficos: formas gráficas vectoriais (e.g., polígonos), imagens e texto.

Os desenhos SVG podem ser interactivos e dinâmicos. Através de *scripting*, é possível manipular o DOM (*Document Object Model*) do SVG e ter acesso a todos os elementos, atributos e propriedades. Além disso, a norma define um conjunto de *event handlers* (e.g. `onmouseover` e `onclick`) que podem ser atribuídos a qualquer objecto gráfico do SVG.

Existem, neste momento, três perfis de SVG: o perfil *SVG Full* e os perfis móveis: *SVG Tiny* e *SVG Basic*. O perfil *Full* inclui todos os módulos da especificação SVG. Os perfis *Tiny* e *Basic* foram definidos tendo como alvo pequenos dispositivos móveis, com limitações de recursos. O perfil *Tiny* é orientado para dispositivos muito limitados enquanto que o *Basic* é orientado para dispositivos ou pouco mais potentes. O perfil *Tiny* é um subconjunto do perfil *Basic*, sendo este um subconjunto do SVG 1.1.

O perfil que nos interessa mais para o nosso projecto é, obviamente, o perfil *SVG Tiny*, uma vez que o nosso sistema é orientado para dispositivos muito limitados.

Sendo tanto o SVG como o GML linguagens XML, o primeiro é facilmente obtido através de aplicação de folhas de estilo XSLT, a partir do segundo.

4 Desenho do Sistema

A motivação principal deste projecto era desenvolver um sistema que permitisse visualizar informação geográfica, disponível num servidor em formato GML, em dispositivos móveis usando, na medida do possível, tecnologias e formatos abertos. O facto de o formato de entrada dos dados geográficos ser o GML iria permitir desenvolver um sistema independente da tecnologia de base de dados geográficos e, por isso mesmo, adaptável a vários sistemas.

A primeira versão do sistema utilizava ficheiros com dados GML como fonte de dados. No entanto esta solução era pouco flexível pelo que foi desenvolvida uma segunda versão que tem como fonte de dados um servidor WFS. A introdução de um *Web Feature Server* tornou o sistema mais flexível uma vez que podem ser adicionadas ou retiradas fontes de dados geográficos de uma forma transparente para o sistema M-GIS. Para além disso, o uso de um WFS permite configurar o sistema com fontes de dados geográficos em diversos formatos, incluindo formatos proprietários.

A arquitectura do sistema é apresentada na Figura 1. A informação geográfica é obtida com recurso a um WFS que, por sua vez, pode ser configurado de forma a obter os dados geográficos de diversas fontes, inclusive de outros WFS. O WFS devolve informação geográfica no formato GML. A informação geográfica em formato GML é transformada no formato gráfico SVG usando uma folha de transformação XSLT. O mapa, no formato SVG, é enviado ao cliente que o poderá manipular directamente.

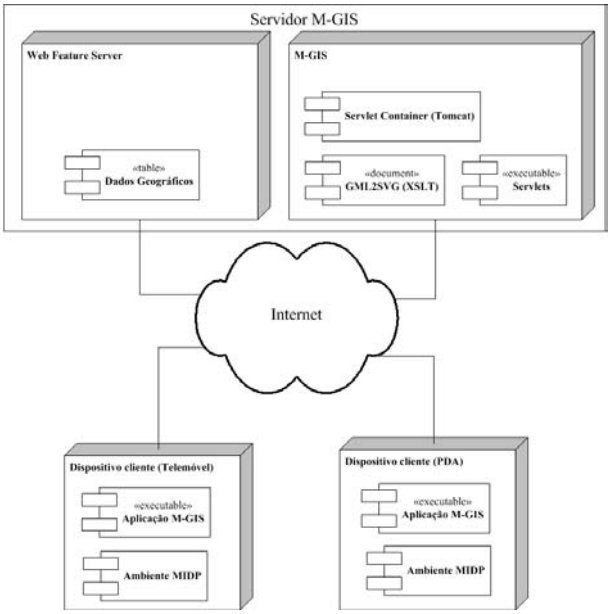


Figura 1. Arquitectura geral do sistema M-GIS

O Servidor M-GIS

O servidor responde a dois tipos de pedidos:

- Pedidos de listagem das camadas (*layers*) disponíveis no WFS.
- Pedido do conteúdo de uma área. Este pedido deve incluir a delimitação da área pedida.

A informação geográfica pode ser organizada por camadas, de forma a que o cliente apenas visualize o tipo de informação que necessitar. A configuração das camadas disponibilizadas é feita no WFS.

Quando o servidor recebe um pedido do conteúdo de uma área, esse pedido é redireccionado ao WFS (com algumas modificações de sintaxe). O WFS devolve a informação geográfica da área pedida, em formato GML, que o servidor M-GIS se encarregará de transformar para SVG, usando uma folha de transformação XSLT, e enviar ao cliente.

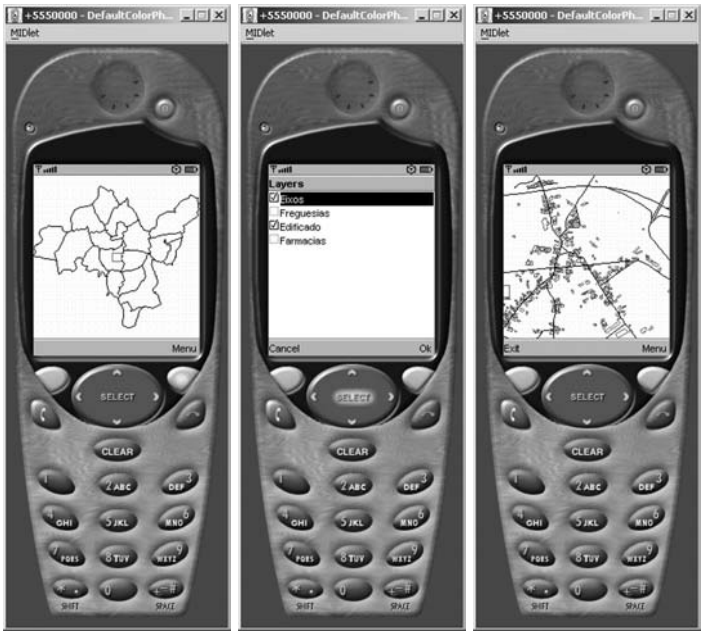
A comunicação entre o servidor e os clientes é feita sobre HTTP, sendo os pedidos dos clientes efectuados através do método HTTP GET codificados em pares parâmetro-valor no URL do pedido.

O Cliente M-GIS

O cliente M-GIS é uma aplicação Java[4] com as seguintes funcionalidades:

[4] A aplicação foi implementada usando a plataforma MIDP (*Mobile Information Device Profile*). Esta plataforma consiste, basicamente, num conjunto de APIs dese-

– Escolher a área a visualizar
– Escolher quais as camadas do mapa que se pretendem visualizar.
– Visualizar o mapa: deslocar, aproximar e afastar a vista.
– Pesquisar no mapa.
– Configurar o endereço do servidor que fornece os mapas.



(a) Ecrã inicial (b) Camadas disponíveis (c) O mapa

Figura 2. Sequência de interacção com o sistema m-GIS

A Figura 2 ilustra uma sequência de interacção do utilizador com o sistema M-GIS a partir de um telemóvel. A interacção resume-se a escolher a área a ser visualizada, as camadas de informação e, por fim, a navegação pelo mapa.

Nesta aplicação, o utilizador escolhe previamente a área que pretende visualizar sobre um mapa já presente no dispositivo. A escolha da área é feita manipulando um rectângulo gráfico no ecrã do dispositivo — Figura 2(a). O utilizador pode deslocar, aumentar ou diminuir o rectângulo de forma a escolher a área a visualizar. Depois de escolhida a área, a aplicação exibe uma lista

nhadas especificamente para dispositivos muito limitados. É esta a plataforma encontrada nos telemóveis com tecnologia Java.

das camadas de informação presentes no WFS. A lista das camadas disponibilizadas é obtida através de um pedido ao servidor M-GIS, que por sua vez fará o pedido ao WFS. Depois de escolhidas as camadas a visualizar (Figura 2(b)) é apresentado o mapa correspondente. A aplicação permite realizar operações de aproximação, afastamento e deslocamento da vista sobre o mapa. Permite também efectuar pesquisas e identificação de objectos no mapa. Todas estas operações são realizadas localmente no dispositivo, i.e., não existe comunicação com o servidor.

O núcleo central da aplicação do cliente é constituído por um interpretador e um visualizador de SVG desenvolvidos neste trabalho.

5 Implementação e Avaliação do Projecto

O servidor foi desenvolvido tendo por base o *servlet container Apache Tomcat* e consiste num conjunto de *servlets* e uma folha de transformação XSLT (transformação de GML para SVG Tiny).

O servidor é composto por duas *servlets*: `ListLayer` e `GetLayer`. A *servlet* `ListLayer` retorna ao cliente a lista de camadas (*layers*) de informação disponíveis no WFS. Esta lista é pedida directamente ao WFS, fazendo um pedido de *GetCapabilities*, que retorna, entre outras informações, as camadas configuradas no WFS. O resultado deste pedido (todos os pedidos e respostas ao WFS são feitos em XML) é interpretado e enviado ao cliente. A *servlet* `GetLayer` é responsável por obter do WFS a informação geográfica em GML representativa de uma determinada área (a área é definida em parâmetros da *servlet*), transformá-lo em SVG e enviar a resposta ao cliente. Os parâmetros aceites por esta *servlet* são os seguintes:

xmin A menor coordenada x do rectângulo que representa a área que se pretende visualizar.
xmax A maior coordenada x do rectângulo que representa a área que se pretende visualizar.
ymin A menor coordenada y do rectângulo que representa a área que se pretende visualizar.
ymax A maior coordenada y do rectângulo que representa a área que se pretende visualizar.
layers Uma lista de camadas que se pretende visualizar. Os nomes das camadas são separados por vírgulas.

Estes parâmetros são encapsulados num pedido do tipo *GetFeature* que é feito ao WFS. O resultado deste pedido ao WFS é um documento GML que será depois transformado em SVG e devolvido ao cliente. A transformação é feita recorrendo à API JAXP da Sun e a uma folha de transformação XSLT. Originalmente foi usado o processador de XML *Xerces*, uma vez que é este o processador usado pelo J2SDK1.4.x. No entanto, devido à existência de alguns *bugs* optou-se por usar o processador Saxon 6.5.2 [10]. Esta *servlet* mantém a folha de transformação XSLT em *cache* de forma a diminuir o tempo de interpretação do ficheiro XSLT.

Neste projecto foi usado o *deegree Web Feature Server* [2] como WFS.

Transformação GML para SVG

A transformação de documentos GML para documentos SVG é feita através de uma folha de transformação (XSLT).

A folha de transformação usada para transformar GML em SVG foi adaptada de um exemplo da *OS MasterMap* [8]. Foram introduzidas algumas modificações básicas, uma vez que a XSLT original se destinava a transformar GML em SVG Full e não em SVG Tiny. Uma dessas modificações é a conversão dos números para a gama suportada pela norma SVG Tiny. Para além disso foi modificada a forma e o tipo de estilos aplicados aos elementos SVG, mais uma vez devido às restrições impostas pela norma SVG Tiny, tais como a de não suportar estilos CSS.

A norma SVG Tiny suporta apenas números de vírgula fixa entre -32767.9999 a 32767.9999. No entanto a informação geográfica, contida nos ficheiros GML, pode usar valores arbitrariamente grandes para representar coordenadas de pontos no espaço. Isto obriga a que a transformação de GML para SVG efectue uma modificação de escala. Na folha de transformação utilizada essa alteração é feita recorrendo aos valores do elemento <*gml:boundedBy*>. Este elemento GML define uma caixa (*bounding box*) dentro da qual todos os pontos definidos no documento GML devem estar contidos. Assim podemos usar o maior valor absoluto das coordenadas dessa caixa, para definir um factor de escala a aplicar a todos os outros valores, de forma a que o resultado esteja dentro da gama pretendida.

Uma vez que queremos distinguir visualmente objectos geográficos diferentes é necessário aplicar diferentes estilos a esses objectos no documento SVG resultante da transformação. Para tal ser possível é necessário que o documento GML contenha informação sobre o tipo de objecto representado, i.e., é necessário que o documento GML indique se determinado objecto é uma estrada, um edifício, um jardim, etc. A norma GML não define elementos para representar objectos como os mencionados atrás, mas define regras para a construção de esquemas GML que definem esses elementos. Assim, foi usado um esquema GML[5] que define, entre outras coisas, elementos para representar edifícios, zonas verdes e vias. A folha de transformação reconhece os elementos definidos no esquema e aplica diferentes estilos conforme o tipo de objecto geográfico representado (e.g. os jardins são pintados a verde).

O Exemplo 1 mostra um exemplo de um documento GML típico, conforme o esquema desenvolvido. O resultado da transformação em SVG desse documento é apresentado no Exemplo 2.

Podemos ver que um objecto geográfico, por exemplo, uma rua, é representado em SVG usando um elemento <*g*>, sendo que o nome é codificado no elemento <*title*> e a geometria no elemento <*path*>.

O estilo aplicado aos objectos é diferenciado, agrupando objectos do mesmo tipo dentro de um elemento <*g*> e atribuindo a esse elemento o estilo definido

[5] O esquema GML usado foi adaptado de um já existente criado no âmbito de um outro projecto desenvolvido no INESC Porto ([12]).

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gml="http://www.opengis.net/gml"
  xsi:noNamespaceSchemaLocation="mgismap.xsd">
  <gml:boundedBy>
    <gml:Box>
      <gml:coord>
        <gml:X>-36391.6369611</gml:X><gml:Y>170507.5437507</gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>-36012.3401935</gml:X><gml:Y>170770.2912157</gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <MapaMember>
    <Edificado>
      <EdificadoMember>
        <Edificio fid="_1">
          <gml:name>Edificio</gml:name>
          <gml:coverage>
            <gml:Polygon>
              <gml:outerBoundaryIs>
                <gml:LinearRing>
                  <gml:coord>
                    <gml:X>-36263.089</gml:X>
                    <gml:Y>170724.85</gml:Y>
                  </gml:coord>
                  <!--  ...   -->
                  <gml:coord>
                    <gml:X>-36263.089</gml:X>
                    <gml:Y>170724.85</gml:Y>
                  </gml:coord>
                </gml:LinearRing>
              </gml:outerBoundaryIs>
            </gml:Polygon>
          </gml:coverage>
        </Edificio>
      </EdificadoMember>
    </Edificado>
  </MapaMember>
  <MapaMember>
    <EixosVia>
      <ViaMember>
        <Eixo fid="_237">
          <gml:name>Avenida do Lidador da Maia</gml:name>
          <!-- ... -->
        </Eixo>
      </ViaMember>
    </EixosVia>
  </MapaMember>
</Mapa>
```

Exemplo 1: Exemplo de um documento GML

```
<?xml version="1.0" encoding="UTF-8"?>
<svg id="svgAll" width="100%" height="100%"
   preserveAspectRatio="xMidYMid meet"
   viewBox="-6982.9544 -32767.9999 72.7808 50.4169">
<defs>
  <g id="pointSymbol" overflow="visible">
    <circle cx="0" cy="0" r="1.0" fill="#000000" stroke="#000000"/>
  </g>
</defs>
<g transform="matrix(1 0 0 -1 0 0)" fill="none"
   stroke-width="0.1" stroke="#000000">
  <g fill="#905050" stroke="#000000" stroke-width="1">
    <g id="_1">
      <title>Edificio</title>
      <path d="M-6958.2882,32759.2805z"/>
    </g>
  </g>
  <g fill="none" stroke="#000000" stroke-width="1">
    <g id="_237">
      <title>Avenida do Lidador da Maia</title>
      <path d="M-6985.7329,32804.05711159.0339,-46.9665"/>
    </g>
  </g>
</g>
</svg>
```

Exemplo 2: Resultado da transformação em SVG do documento do Exemplo 1

para o tipo de objecto. Podemos ver pelo Exemplo 2 que o edifício está inserido num elemento <*g*> com uma cor castanha (`fill=■#905050■`) enquanto que a rua não tem preenchimento (`fill=■none■`).

O elemento <*g*> de topo serve apenas para realizar uma adaptação do eixo das coordenadas y. No SVG, o eixo y, tem uma orientação de cima para baixo, enquanto que no GML usado, o eixo y tem uma orientação de baixo para cima. Podemos ver que o elemento <*g*> de topo define uma transformação que inverte o eixo das coordenadas y.

Praticamente todos os elementos de geometria do GML são transformados usando o elemento <*path*> do SVG. A excepção é o elemento <*Point*> do GML que é convertido para o elemento <*circle*> do SVG. De facto, este elemento, <*circle*>, é apenas definido uma vez no ficheiro SVG e todas as ocorrências de um ponto no documento são transformadas em referências para esse elemento. Isto é feito definindo o elemento <*circle*> dentro do elemento <*defs*> do SVG e atribuindo um "`id`" ao elemento. O elemento <*path*> é usado para representar quase todos os elementos geométricos porque permite optimizar o documento SVG resultante em termos de tamanho final.

Interpretação de SVG

Para realizar a interpretação do documento SVG no cliente é utilizado um interpretador to tipo *pull* fornecido pela biblioteca kXML [3]. Um exemplo do uso do kXML para realizar interpretação de XML pode ser encontrado em [7].

Uma vez que o kXML é um interpretador de XML, e não um interpretador de SVG, é necessário construir, em cima do kXML, algo que interprete os elementos e atributos SVG. O Exemplo 3 mostra parte de um documento SVG constituído por apenas um polígono. Um polígono é especificado através do elemento <*polygon*>, sendo os pontos que constituem o polígono definidos no atributo "`points`", como uma sequência de pares de valores numéricos. Neste caso, o kXML é capaz de nos fornecer, por exemplo, o valor do atributo "`points`". No entanto para termos acesso aos pontos que constituem o polígono é necessário interpretar esse valor (para o kXML, trata-se apenas de uma cadeia de caracteres).

```
<g id="_N3HE1s3th50">
    <title>Pedroucos</title>
    <polygon points="-7057.0669,30810.5546 -7055.2255,30829.6328
     -7052.8957,30837.5716 -7050.1427,30840.2181 -7046.4364,30843.9228
     -7036.6944,30849.7447 -7030.9762,30852.0734 -7027.9053,30852.9202
     -7026.4227,30857.2601 -7025.7875,30858.8479 -7023.6697,30858.5304
     -7018.7986,30856.8367 -7016.9983,30856.8367 -7014.2451,30857.2601
     -7009.6919,30857.8952 -7004.8207,30858.7421 -6999.9496,30859.483"/>
</g>
```

Exemplo 3: Parte de um documento SVG típico

O Exemplo 3, é um exemplo típico do conteúdo de um documento SVG. O que mostra que grande parte da informação está contida no valor dos atributos dos elementos que definem elementos gráficos. Devido a esta estrutura típica de um documento SVG, acontece que se torna ineficiente interpretar o SVG. A razão é, muito simplesmente, a seguinte: o documento SVG tem de ser lido praticamente duas vezes. Primeiro, o kXML tem de fazer a interpretação para devolver à aplicação os elementos e atributos; depois a aplicação tem de interpretar o conteúdo dos atributos (na maior parte dos casos extrair valores numéricos).

A alteração mais natural consiste em incluir no kXML funcionalidades de interpretação de documentos SVG. Ou seja, obrigar o kXML a interpretar alguns atributos e, em vez de devolver à aplicação uma cadeia de caracteres, devolver, por exemplo, um *array* de pontos. Desta forma a interpretação do ficheiro é feita de uma só vez. Esta alteração foi levada a cabo modificando-se o código fonte do kXML (o kXML é um projecto *open source*), de forma a interpretar os atributos "`points`" dos elementos <*polygon*> e <*polyline*> e o atributo "`d`" do elemento <*path*>. São estes os atributos responsáveis pela maior parte da informação nos nossos documentos SVG.

Testes e Resultados

Esta secção apresenta os resultados de algumas medições efectuadas sobre o sistema M-GIS. O sistema é analisado segundo vários eixos: memória consumida,

tamanho máximo do mapa visualizado e tempos de transformação e de interpretação dos documentos SVG.

Os testes foram efectuados usando um computador com processador AMD Athlon a 1 GHz e com 256 Mb de memória RAM. Neste computador foram instalados o servidor WFS e o servidor m-GIS. O emulador de telemóvel usado foi também executado nesta máquina. No caso do PDA, foi usado um HP Jornada 548 Pocket PC com 32Mb de memória e com a máquina virtual Personal Java [11] da Sun instalada. Neste caso não existe emulação. A MIDlet foi executada recorrendo à ferramenta ME4SE [9] que permite executar MIDlets em ambientes Java "normais". A ligação do PDA com o computador onde se encontrava o servidor foi feita através de USB.

Uma das características dos dispositivos MIDP é a quantidade de memória reservada para as aplicações Java. A quantidade de memória do dispositivo impõe um limite à quantidade de informação geográfica que pode ser visualizada pelo dispositivo, ou seja, limita o tamanho do mapa visualizado.

Tabela 1 Dimensões máximas dos mapas de acordo com a memória disponível

Memória (Kb)	Dimensão do mapa (m)	Camadas	Tamanho do documento SVG (Kb)
192	1033x1029	eixos	29.1
256	1549x1544	eixos	55.0
320	2324x2316	eixos	95.5
384	2840x2831	eixos	138.2
448	3098x3088	eixos	159.7
512	3615x3603	eixos	205.2
256	258x257	edificado	22.7
320	516x515	edificado	65.0
448	775x772	edificado	103.4
256	258x257	eixos + edificado	34.7
384	516x515	eixos + edificado	78.3
448	775x772	eixos + edificado	124.3

A Tabela 1 apresenta, para uma determinada quantidade de memória, o tamanho máximo que o dispositivo consegue visualizar. Estes valores foram obtidos com o emulador de telemóvel do WTK[6]. Apenas foram testados mapas com três tipos de informação: eixos de via, edificado e eixos de via mais edificado. Podemos ver que com 192Kb conseguimos visualizar mapas dos eixos de via de dimensão até cerca de 1x1 quilómetros. Para visualizar mapas de edificado precisamos de, pelo menos, 256Kb de memória.

Um dos factores mais limitativos do sistema M-GIS é o tempo de transformação e de interpretação dos documentos SVG. A Tabela 2 apresenta alguns tempos medidos com o emulador de telemóvel do WTK. A tabela apresenta a dimensão do mapa, as camadas presentes no mapa, o tempo que demora a transformação de GML para SVG, o tempo que o dispositivo demora a efectuar

[6] *Wireless Toolkit* – O *toolkit* de desenvolvimento de aplicações MIDP da Sun.

a interpretação do documento[7], o tempo que o mapa demora a ser desenhado no ecrã do dispositivo e o tamanho do documento SVG.

Tabela 2 Tempos de transformação e de interpretação

Dimensão mapa (m)	Camadas	Tempos (segundos)			Tam. SVG (Kb)
		XSLT	Interpretação	*Rendering*	
258x257	eixos	0.8	20.5	3.2	13.0
516x515	eixos	0.8	22.6	3.6	14.2
775x772	eixos	1.0	33.8	5.4	21.9
1033x1029	eixos	1.1	46.3	7.3	29.9
1291x1287	eixos	0.5	57.3	8.9	37.5
1549x1544	eixos	1.7	83.1	13.0	55.4
1807x1801	eixos	2.5	108.9	16.9	72.4
2066x2059	eixos	25.5	128.2	19.5	85.3
2324x2316	eixos	15.2	147.0	22.7	97.8
2582x2574	eixos	49.1	175.2	25.9	117.1
2840x2831	eixos	72.7	211.5	30.4	141.5
3098x3088	eixos	98.4	244.8	34.1	163.5
3357x3346	eixos	147.0	285.5	38.4	190.2
3615x3603	eixos	160.0	313.7	41.3	210.1
258x257	edificado	0.3	40.6	6.5	23.3
516x515	edificado	0.9	111.6	14.6	66.5
775x772	edificado	27.4	175.1	17.2	105.9
258x257	eixos+edificado	2.3	59.3	9.7	35.6
516x515	eixos+edificado	2.7	130.0	18.1	80.2
775x772	eixos+edificado	54.3	204.2	22.5	127.3

A Tabela 3 mostra alguns tempos medidos com um dispositivo real: um PDA HP Jornada.

Tabela 3 Tempos de de interpretação num dispositivo real: HP Jornada

Dimensão do mapa (m)	Camadas	Tempos (segundos)	
		Interpretação	*Rendering*
306x305	eixos+edificado	17.0	3.0
816x814	eixos+edificado	50.0	9.0

Os valores apresentados indicam que o sistema apenas responde de forma aceitável para mapas muito pequenos. Quando a área abrangida pelo mapa é grande, ou quando escolhemos visualizar várias camadas, o tempo de espera aumenta consideravelmente tornando o sistema muito pouco usável.

Basicamente, existem dois gargalos no sistema:

– A transformação através de XSLT. Quando o documento GML se torna demasiado grande, a transformação para SVG torna-se demasiado lenta. Existem vários factores que contribuem para isto. A transformação é feita através

[7] Este tempo inclui também o tempo de ligação ao servidor, ou seja, é medido desde o momento em que é feito o pedido ao servidor até a interpretação do documento estar concluída.

de XSLT em Java usando a API JAXP. No nosso caso é usado o processador de XML Saxon. A transformação através de XSLT requer a criação das árvores DOM tanto do documento a transformar como do documento XSLT. À medida que o documento a transformar aumenta, também aumenta o tempo de interpretação do documento e da criação da respectiva DOM. Para além disso, as árvores DOM requerem muita memória, o que, no nosso caso vai obrigar ao uso de memória virtual durante a transformação e consequente degradação do desempenho.
- A interpretação e *renderização* do SVG no dispositivo. Uma vez que estamos a lidar com dispositivos muito limitados a nível de processamento, é natural que o desempenho na interpretação e *renderização* dos documentos SVG seja fraco.

6 Conclusões e Trabalho Futuro

O objectivo principal deste projecto era obter uma resposta à pergunta: "Será viável desenvolver um sistema móvel de visualização de informação geográfica tendo por objectivo a interoperabilidade de sistemas GIS, isto é, recorrendo a normas e formatos abertos?"

Os resultados obtidos indicam que sim, mas com algumas limitações. As principais limitações do sistema estão relacionadas com a quantidade de informação que o cliente consegue processar. Estas limitações são de duas ordens: a primeira diz respeito à memória necessária para visualizar um determinado mapa; a segunda relaciona-se com o tempo necessário para processar um mapa. Obviamente que estas são limitações de alguns dispositivos actuais. É de esperar que os próximos dispositivos melhorem o desempenho do sistema no que diz respeito a estas restrições.

Uma outra limitação do sistema prende-se com o aspecto gráfico. As APIs gráficas do MIDP são muito básicas pelo que apenas se conseguem produzir mapas muito simples graficamente. Uma possível forma de resolver este problema seria recorrer às APIs de alguns fabricantes de telemóveis, que disponibilizam funções gráficas um pouco mais completas. A desvantagem seria a redução da portabilidade da aplicação.

As limitações apontadas nos parágrafos anteriores dizem respeito ao cliente, no entanto, existem também algumas limitações, no sistema M-GIS, do lado do servidor. Estas limitações referem-se à transformação de GML para SVG. Contudo, uma vez que as opções que podemos tomar para a implementação do servidor são mais alargadas, estas limitações não causam muita preocupação. Seria perfeitamente possível, por exemplo, implementar a transformação de GML para SVG sem usar XSLT; a transformação pode ser feita usando *Perl*, ou mesmo Java, directamente sobre o documento GML. Obviamente, esta solução diminuiria a flexibilidade do sistema, mas poderia minorar o problema da eficiência da transformação.

Na prática estas limitações significam que o sistema se torna pouco usável quando a quantidade de informação geográfica a transformar e transmitir

é elevada face às características actuais da tecnologia usada e à capacidade de processamento do dispositivo móvel.

Trabalho Futuro

O sistema desenvolvido pode evoluir principalmente segundo três eixos: arquitectura, desempenho e funcionalidades.

Arquitectura Em termos da arquitectura, podemos, por exemplo, pensar em adaptar o mapa resultante da transformação do GML em SVG às capacidades do dispositivo cliente. Deste modo os mapas gerados poderiam ser mais ou menos complexos dependendo da capacidade de processamento e/ou gráficas do dispositivo. Esta adaptação poderá ser realizada recorrendo a diferentes folhas de transformação XSLT (uma por cada tipo de dispositivo) e a um mecanismo de negociação entre o cliente e o servidor de forma a ser estabelecido qual o tipo de mapa suportado pelo cliente.

Desempenho Ao nível do desempenho, o sistema pode ser melhorado principalmente ao nível da transformação do GML para SVG. Para se saber a que nível actuar sobre esta transformação seria, no entanto, necessário um estudo mais aprofundado sobre as causas do baixo desempenho do sistema a esse nível. A solução tanto pode passar por reescrever a folha de transformação de uma forma mais eficiente, como usar extensões do processador XSLT, ou até mesmo implementar a transformação sem recorrer a XSLT. Do lado do cliente o desempenho da aplicação está limitado ao desempenho do próprio dispositivo pelo que nesta vertente existe menos que possa ser feito. Adicionalmente poderá evitar-se a transformação e transmissão de alguma informação anotando as camadas disponíveis com informação acerca das escalas mínima e máxima de visualização (por exemplo, a camada de edificações deve ser visível apenas acima da escala 1:5000).

Funcionalidades Quanto às funcionalidades do sistema, existem várias que podem ter interesse para o utilizador:

- **Localização actual** Exibir o mapa da zona onde o utilizador se encontra no momento. Um serviço deste tipo tem interesse, por exemplo, no caso de pessoas de visita a cidades desconhecidas. Isto implica o uso de dispositivos com capacidade para identificar a localização corrente, o que, no caso dos telemóveis pode ser feito através da informação das células da rede.
- **Pontos de Interesse** Lista de pontos de interesse na área visualizada pelo utilizador. Poderia ser acrescentado uma opção que listasse os pontos de interesse (possivelmente configurados pelo utilizador) da área visualizada, e.g., uma lista de monumentos, postos de bombeiros e de polícia, etc.
- **Pesquisa global** Neste momento a pesquisa é efectuada sobre a área pedida pelo utilizador. Poder-se-ia implementar uma nova funcionalidade que permitisse ao utilizador, por exemplo, pesquisar por uma determinada rua em todo o concelho. O utilizador poderia, depois, visualizar o mapa da área em que situa a rua pretendida.

Referências

1. Jorge C. S. Cardoso. M-GIS: Mobile Geographic Information System. Relatório do projecto, INESC Porto, 2003.
2. Deegree. Projecto Deegree, 2003. http://www.deegree.org.
3. Enhydra.org. Projecto kXML, 2003. http://kxml.enhydra.org.
4. ESRI. ArcPad, 2003. http://www.esri.com/software/arcpad/index.html.
5. ESRI. ESRI - *GIS and Mapping Software*, 2003. http://www.esri.com.
6. Geo InSight. GeoGIS, 2003. http://www.geoinsight.com/Products/Mobile/GeoGIS.cfm.
7. Jonathan Knudsen. *Parsing XML in J2ME[tm].* http://wireless.java.sun.com/midp/articles/parsingxml, Março 2002.
8. OS MasterMap. *Style and XML Examples*, 2003. http://www.ordnancesurvey.co.uk/os_mastermap/xml/.
9. ME4SE. ME4SE, 2003. http://www.me4se.org.
10. Michael Kay. Saxon, 2003. http://saxon.sourceforge.net/saxon6.5.2/.
11. Sun Microsystems. *Personal Java*, 2003. http://java.sun.com/products/personaljava/.
12. Anabela Soares Pinto Monteiro. Servidor de Mapas Vectoriais. Relatório do projecto, INESC Porto, 2002.
13. OGC. *Geography Markup Language (GML) 2.0*. Especificação, OGC, Fevereiro 2001.
14. OGC. *Web Feature Service Implementation Specification*, 2002. http://www.opengis.org.
15. OGC. *Web Map Service Implementation Specification*, 2002. http://www.opengis.org.
16. OGC. Consórcio *Open GIS*, 2003. http://www.opengis.org.
17. Pocket Systems Ltd. PocketGIS, 2003. http://www.pocket.co.uk.

Web Services na Informação Geográfica

Mário André Araújo and Jorge Gustavo Rocha

Universidade do Minho
maaraujo@mail.pt, jgr@di.uminho.pt

Resumo Os Web Services (WS) constituem-se como um novo paradigma de desenvolvimento de software. Estes exibem algumas propriedades fundamentais, como o encapsulamento e a composicionalidade, no desenvolvimento de software.
Desde que surgiu o conceito, os WS têm sido explorados para criar um nível computacional sobre a Informação Geográfica (IG). A IG tem características intrínsecas, como seja a diversidade, complexidade e volume, que se pretendem encapsular através de WS. Este trabalho tem vindo a ser desenvolvido sob os auspícios do consórcio *OpenGIS*, sendo de realçar o sucesso dos serviços *Web Map Service* (WMS) e *Web Feature Service* (WFS).
Estes serviços vieram operacionalizar a comunicação aplicação – aplicação, permitindo-nos trocar informação entre clientes e servidores, que respeitem a norma. No entanto, queremos ser mais ambiciosos e, por isso, podemos dizer que apenas se resolveu um problema sintáctico (as mensagens satisfazem a estrutura definida). Interpretar e usar correctamente os conteúdos carece da manipulação do nível semântico. Para ilustrar esta limitação recorremos a um exemplo que pretende demonstrar a ainda existente necessidade da intervenção humana para a interpretação da informação.
Explora-se uma possível solução que passa pela criação e manutenção de uma ontologia a adicionar à arquitectura dos WS. Os conceitos disponibilizados pelos servidores e manipulados pelos clientes são indexados a esta ontologia comum. Desta forma será possível proporcionar a interoperacionalidade [1]dos WS, isto é, substituindo ou acrescentando dinamicamente novos servidores de IG. Este dinamismo é necessário para que um determinado dispositivo móvel, possa ir sucessivamente encontrando e seleccionando servidores de informação relevante – relevante é algo conceptual – à medida que o seu utilizador se vai deslocando.

1 Introdução

Com o aparecimento das redes informáticas e consequente massificação do uso da Internet, os computadores deixam de ser apenas máquinas que processam informação para assumirem o papel de meio de comunicação. Através da Internet

[1] Capacidade de comunicar, executar programas, ou transferir dados entre diferentes unidades funcionais sem que o utilizador tenha que se preocupar com as características específicas de cada uma dessas unidades[9].

e com a utilização de um computador a informação pode ser trocada e partilhada por todos a um nível cada vez mais abrangente.

A evolução das tecnologias de telecomunicação e a procura pela mobilidade tiveram um impacto fundamental no aparecimento de novas plataformas e consequentemente novas necessidades de disponibilidade de informação. Pretetende-se que a informação seja disponibilizada a qualquer hora, em qualquer lugar e para qualquer plataforma. A portabilidade da informação passa a ser um requisito essencial no âmbito da sua distribuição através da Internet. É neste contexto que aparece o XML (*eXtensible Markup Language*), que se pretende assumir como norma no intercâmbio de informação estruturada.

Ainda assim, a Web limita-se a ser um meio de interacção entre um humano e informação que aparece na forma de texto e gráficos, aparecendo o humano sempre como actor indispensável nesta cadeia de partilha e interpretação de informação.

1.1 Objectivos

Neste artigo estamos interessados em investigar até que ponto os WS podem desempenhar um papel relevante na manipulação de IG. Por um lado, com base nas experiências que têm sido efectuadas com os serviços propostos pelo consórcio OpenGIS, os WS permitem-nos criar um nível de abstracção sobre o qual se pode manipular e partilhar IG, independentemente da sua especificidade própria. Por outro lado, até que ponto estes serviços podem ser concebidos para, autonomamente, procurarem e obterem informação relevante no contexto de soluções móveis. Como defenderemos, esta autonomia só é possível com base na descrição das características semânticas da IG.

1.2 Estrutura

Este artigo encontra-se dividido em cinco partes. Na segunda parte (secção 2) é feita uma abordagem à tecnologia dos WS, com especial ênfase para a normalização no seu desenvolvimento. Na terceira parte (secção 3) são apresentados os WS aplicados à IG, é feita uma abordagem às normas envolvidas e aos problemas relacionados com a interoperacionalidade. Na quarta parte (secção 4) é apresentado um exemplo prático com o intuito de identificar as necessidades e desafios no que diz respeito à interoperacionalidade em geo-web services.

2 Web Services

2.1 Definição

Os WS são aplicações Web com a capacidade para interagir entre si sem a intervenção do ser humano, permitindo a automatização de tarefas que só podiam ser feitas através da interacção dos humanos [5], ou seja, uma norma que define formas de interacção aplicação – aplicação recorrendo a formatos abertos. A

utilização de protocolos normalizados, baseados em XML, proporciona a capacidade de intercâmbio de informação entre aplicações em ambientes eminentemente heterogéneos.

Os WS aparecem portanto como uma plataforma independente para suporte ao desenvolvimento de aplicações distribuídas sobre Internet respondendo. A ideia fundamental centra-se em tornar possível a criação de aplicações modulares com capacidade de auto-descrição, para serem integradas na Internet e estarem acessíveis de e em toda a rede.

2.2 XML Web Services

Sendo um WS uma aplicação vocacionada para comunicar com outras, torna-se necessária a definição de normas que possibilitem essa interacção. Nesta secção vão ser apresentados os XML Web Services que são uma restrição aos WS mencionados na secção 2.1.

A arquitectura dos WS é apresentada de forma simplificada e ilustrada na figura 1. Esta prevê o envolvimento de três entidades distintas:

O Fornecedor publica no catálogo, através da *interface* disponibilizada para o efeito, a existência de um novo serviço.

O Cliente após a consulta do catálogo e se o serviço pretendido for encontrado, usufrui do mesmo directamente requisitando-o ao fornecedor.

O Catálogo informa o cliente fornecendo-lhe a descrição dos serviços e localização dos mesmos.

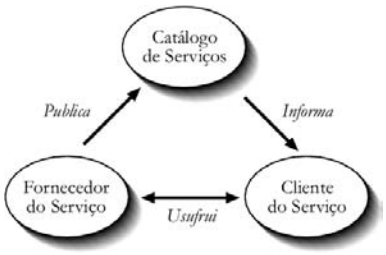


Figura 1. Arquitectura dos XML Web Services

A arquitectura apresentada na figura 1 necessita de protocolos que suportem a comunicação entre as três entidades mencionadas. Esses protocolos, definidos pelo W3C, são:

SOAP *Simple Object Access Protocol*, protocolo de comunicação, baseado em XML, que permite a troca de mensagens entre aplicações independentemente do protocolo de transporte utilizado, no entando, no contexto dos WS o protocolo de transporte utilizado é o HTTP. A troca de mensagens é feita num só sentido, não existindo a noção de estado.

WSDL *Web Service Definition Language*, como o próprio nome indica, trata-se de uma linguagem que permite fazer a descrição dos WS, em termos de métodos e respectivos parâmetros fornecidos pelo WS.

UDDI *Universal Description, Discovery and Integration*, um serviço de registo e descoberta de WS descritos em WSDL, utilizado para armazenar, fornecer e devolver informação acerca de WS existentes.

Desta forma o fornecedor de serviço pode ser desenvolvido numa qualquer linguagem de programação, deverá no entanto, definir a sua *interface* em WSDL. Posto isto, é necessário que seu o registo no UDDI seja efectuado. Um qualquer cliente pode então procurar o serviço no já referido UDDI, se o cliente desejar invocar o serviço, deverá fazê-lo através da troca de mensagens SOAP.

A desordem no desenvolvimento de WS põe em causa o seu grande objectivo – a interoperacionalidade – e mesmo apesar das recomendações do W3C (*World Wide Web Consortium*), os produtores de software começaram a seguir os seus próprios caminhos.

Em 2002, a *Web Services Interoperability Organization*[2](WS-I), uma organização que reúne os maiores produtores de *software*, surge com o principal objectivo de criar uma "linha" orientadora para o desenvolvimento de WS de maneira a manter a interoperacionalidade, publicando para isso um documento [2] que consiste numa série de recomendações a adoptar para o desenvolvimento de WS.

Só o tempo poderá dizer se estas recomendações serão de facto seguidas pelos produtores de software, ou se aparecerão outras. No entanto, o importante é que se chegue a um consenso para o desenvolvimento de WS.

3 Geo-Web Services

3.1 Web Services e Informação Geográfica

A IG tem vindo a ser coleccionada em formato digital há várias décadas e os Sistemas de Informação Geográfica (SIG) são componente essencial na gestão, manutenção, representação, armazenamento, análise e transformação dessa IG. A complexidade da IG aliada á heterogeneidade de organizações, sistemas e processos que a manipulam, dificultam a partilha da mesma que acaba por ficar tendencialmente confinada ao contexto em que foi produzida. Estas características causam dificuldades no que diz respeito à partilha e acesso à IG em ambientes distribuídos.

As tecnologias SIG tendem a caminhar no sentido de adquirir capacidades que permitam a sistemas independentes trocar informação entre si de forma transparente, apesar de internamente a armazenarem em formatos distintos e potencialmente incompatíveis [1].

O aparecimento dos WS trouxe grandes expectativas no âmbito da sua utilização como forma de valorização da IG em diversas vertentes. Aos WS podem

[2] `http://www.ws-i.org/`

ser confiadas diversas funções que vão desde a localização de IG relevante até ao encapsulamento de especificidades dessa IG, como sendo por exemplo o sistema de coordenadas em que esta se encontra. Uma valência que pode vir a ser conseguida, tem a ver com a partilha da informação no sentido da cooperatividade, isto é, aproveitando a bi-direccionalidade que os WS oferecem podem criar-se repositórios que permitam não só operações de consulta mas também de actualização da IG.

O consórcio *OpenGIS* [3], uma associação sem fins lucrativos, dedicada a promover novas abordagens, técnicas e comerciais, para a interoperacionalidade tem como objectivo primordial resolver os problemas de partilha de dados e transferir os sistemas SIG do seu ambiente actual, suportados maioritariamente por ambientes fechados e monolíticos, para ambientes suportados por tecnologias abertas e distribuídas.

Como documento orientador para o desenvolvimento de serviços que promovam a interoperacionalidade entre fontes de informação e tecnologias surge o *OpenGIS Reference Model* (ORM) [4] que apresenta as linhas orientadoras para a criação de uma plataforma de trabalho, sobre a qual os implementadores podem desenvolver aplicações que valorizem a interoperacionalidade da IG.

3.2 OpenGIS Web Services

O desejo da criação de um ambiente distribuído que promova a já mencionada interoperacionalidade entre tecnologias e fontes de informação começa antes mesmo da existência dos Web Services [6].

Os XML Web Services correspondem a um esforço de normalização posterior aos *OpenGIS* Web Services. Por isso, os Geo-WS não contemplam a auto-descrição através de um documento WSDL, nem os mecanismos de catalogação UDDI e também não suportam o encapsulamento das mensagem em SOAP.

As mensagens são trocadas no protocolo HTTP (através das operações GET e POST), com a vantagem de se poderem testar usando um navegador vulgar, como o *Netscape*. A auto-descrição é feita através de uma operação comum a todos os Geo-WS, `GetCapabilities`, que devolve um documento XML. Não existe a noção de catálogo para registo dos Geo-WS disponibilizados, semelhante ao UDDI.

Numa tentativa de criar um modelo conceptual que defina um conjunto de serviços que abranja todas as áreas da IG, o consórcio *OpenGIS* sugere a criação de um conjunto de WS, através dos quais os clientes trocam IG, de acordo com formatos e protocolos bem definidos. Alguns desses serviços já gozam de um certo consenso, mas outros ainda estão em discussão. Os dois serviços mais dessiminados são:

Web Mapping Service (WMS) Este serviço normaliza o processo de requerer mapas por parte do cliente. Os mapas são requeridos a um WMS, identificando um conjunto de camadas e respectivos parâmetros necessários à

[3] `http://www.opengis.org`

sua visualização. Na especificação destes serviços [8,7] são definidas três operações: `GetCapabilities` que retorna meta-informação do serviço, descrevendo-o e enumerando os parâmetros que este aceita; `GetMap` que retorna uma imagem do mapa de acordo com os parâmetros especificados no pedido; `GetFeatureInfo` trata-se de uma operação opcional que retorna informação acerca de entidades especificas apresentadas no mapa.

Web Feature Service (WFS) Este serviço foi concebido pelo consórcio *OpenGIS* para retornar informação geográfica discreta, pronta a ser manipulada. A informação retornada é codificada em *Geographic Markup Language* (GML), sendo o GML uma recomendação do mesmo consórcio. É, como a maioria dos formatos baseado em XML, preferencialmente utilizada para transmitir informação entre sistemas ou aplicações diferentes. Este serviço suporta operações de inserção, actualização, remoção, inquérito e descoberta de entidades geográficas. Em resposta aos pedidos efectuados um WFS devolve GML, proporcionando ao cliente o processamento dos dados. As operações definidas para este serviço são: `GetCapabilities` que retorna meta-informação do serviço, descrevendo-o e enumerando os parâmetros que este aceita; `DescribeFeatureType` que permite obter uma descrição da estrutura de cada uma das entidades disponibilizadas; `GetFeature` retorna a IG pretendida em GML; `Transaction` trata-se de um pedido que visa modificar o conteúdo do repositório de IG; `LockFeature` que permite bloquear o acesso a uma determinada entidade durante uma transacção.

Os outros serviços previstos incluem: Web Coverage Service (WCS), Sensor Collection Service (SCS), Gazetteer Service, Geocoder Service, Catalog Service, Geoparser Service, entre outros.

3.3 Terra Service: um serviço simultaneamente XWS e WMS

O serviço Terra Service, lançado em 1998 pela Microsoft, TerraServer.com, Compaq e USGS, é um bom exemplo de um WS bem documentado através de publicações, [3], ou no próprio sítio [4]. Com base em tecnologia Microsoft, tem sido também usado para promover as capacidades da plataforma Microsoft .NET para a concretização deste tipo de software, quer do lado do servidor quer do lado do cliente.

Usamos o Terra Server como um exemplo concreto de um serviço que disponibiliza duas *interfaces* distintas, que apesar de se tratarem ambas de WS, são implementadas de acordo com duas normas diferentes:

TerraServer como um XML Web Service Este WS oferece um conjunto de operações desenhadas sem qualquer critério de uniformização com outros serviços relacionados com IG. Para saber as operações disponibilizadas, pode consultar-se o próprio serviço através de um documento WSDL. Este serviço pode ser utilizado, por exemplo, através de um cliente desenvolvido para o efeito.

[4] `http://terraserver-usa.com/`

OpenGIS WMS Este WMS obedece à recomendação do *OpenGIS*, passando a disponibilizar uma operação `GetCapabilities`. Desta forma, utilizando um qualquer cliente, desenvolvido por uma qualquer entidade, desde que conforme com as recomendações do *OpenGIS*, poder-se-ia manipular e utilizar a IG fornecida pelo WMS.

No segundo caso, e porque se encontra desenvolvido com base em normas bem específicas, atinge-se a barreira da interoperacionalidade, no sentido em que se prevê que todas as entidades conheçam a sintaxe da informação. No entanto, apesar da informação poder ser entendida por todos, será que é bem entendida? Pretende-se introduzir aqui a questão do significado dos dados, ou seja, da sua semântica.

4 Desafios Semânticos

A utilização de WS para disponibilizar IG trará vantagens importantes, na medida em que vem unificar a forma de aceder à mesma, não só no acesso a mapas através dos WMS, mas também no acesso a entidades através dos WFS. Ao encapsular determinadas especificidades da IG, para o consumidor final, deixam de ter importância questões como o formato, a fonte, etc...

Apesar estas vantagens constituirem um avanço significativo, pretende-se chegar ainda mais longe, minimizando a intervenção humana ao ponto de permitir que a IG relevante seja encontrada e obtida automaticamente.

Na secção 4.1 aborda-se esta temática recorrendo a um exemplo, que pretende demonstrar o que já se conseguiu e quais os desafios para o futuro.

4.1 Enunciado do Problema

Tome-se como exemplo a utilização de dispositivos móveis para aceder a IG. Estes dispositivos precisam de obter informação sobre os locais onde se encontram, para isso utilizam-se WS específicos.

Imagine-se que um utente de um serviço móvel se encontra algures num país A, onde alugou um automóvel, pretende encontrar o melhor caminho para uma determinada localidade do país vizinho B.

Nesta situação, há alguns cenários que se podem traçar:

Dispositivo móvel com toda a informação previamente carregada Antes de viajar, o utente deverá carregar no dispositivo móvel toda a informação sobre os países de destino, o que em dispositivos de capacidade limitada seria inviável.

Acesso a um serviço através da rede GSM Neste caso, o utente não necessita de armazenar previamente a informação antes de viajar, no entanto há que ter em conta a necessidade de cobertura GSM e as limitações na transmissão de dados.

Acesso a Web Services locais Como temos vindo a mostrar ao longo deste artigo, os WS oferecem uma camada computacional bem definida. Esta camada permite que o dispositivo móvel possa descarregar informação de um ou mais WS sobre os países A e B[5].

Interessa-nos viabilizar este terceiro cenário, utilizando a tecnologia dos WS. Vamos imaginar que existe a informação geográfica representada graficamente na figura 2, assumindo que a aplicação do utente já encontrou e pode aceder à informação de dois WS: um com informação do país A e outro com informação do país B.

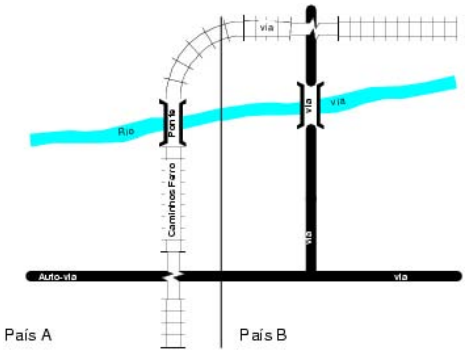


Figura 2. Mapa exemplo

Pretende-se então saber como pode a aplicação determinar que algo que é identificado como "via" pelo WS B, é a continuação da "Auto-via" identificada pelo WS A. Ou como sabe a aplicação que o significado de rio não é um via adequada para o veículo alugado do utente?

Só a interpretação humana pode ajudar a perceber que tipo de informação está disponível, eliminando assim duvidas e ambiguidades. Para que a interoperacionalidade entre geo-web services seja realmente atingida tem que ser resolvida esta questão.

Conclui-se que não é suficiente a descrição retornada pelas operações `GetCapabilities` ou `DescribeFeatureType`, e por essa razão ter-se-á que recorrer a um mecanismo que permita compreender o significado dos dados.

4.2 Inclusão de meta-informação

Para ultrapassar esta limitação dos geo-web services, onde falta informação sobre o significado dos dados, pode começar-se por utilizar um recurso já previsto no

[5] Uma das características da informação geográfica é que a mesma existe em mais abundância junto ao local a que se refere.

meta-documento[6] que define o formato do documento retornado pela operação `GetCapabilities`.

Assim, a informação sobre cada uma das entidades seria estendida com alguma informação semântica no campo `Keywords`.

```
<FeatureType>
  <Name>auto-via</Name>
  <Title>...</Title>
  <Abstract>...</Abstract>
  <Keywords>road</Keywords>
  <SRS>...</SRS>
  <LatLongBoundingBox .../>
</FeatureType>
```

Figura 3. Entidade do país A como **road**

```
<FeatureType>
  <Name>via</Name>
  <Title>...</Title>
  <Abstract>...</Abstract>
  <Keywords>road</Keywords>
  <SRS>...</SRS>
  <LatLongBoundingBox .../>
</FeatureType>
```

Figura 4. Entidade do país B também como **road**

Seria de esperar que as outras entidades também aparecessem marcadas com o respectivo significado (p.e. *railway*, *river*, *bridge*, etc). No entanto, se falamos em semântica pode bem colocar-se a questão do significado das *Keywords* referidas anteriormente.

Ontologias, precisam-se! Para que estas *Keywords* funcionem, temos que estabelecer um vocabulário onde ocorram palavras cujo significado está bem entendido, e desta feita criar relações entre os termos desse vocabulário que permitam inferir, por exemplo, que um automóvel pode circular numa ponte rodoviária. Na verdade queremos estabelecer relações entre os termos auto-estrada, estrada, itinerário principal, itenerário complementar, estrada nacional, estrada municipal, etc, que permitam inferir que em todos eles pode circular um veículo automóvel. De igual modo, saber que existem outras vias de comunicação, adequadas a outros meios...

[6] Vamos usar o termo meta-documento para referir um *schema*.

Para tal, poder-se ia recorrer a um *thesaurus*, com a capacidade de relacionar todos os conceitos capturados pela IG. Numa perspectiva mais abrangente poderia falar-se em ontologia, em vez de *thesaurus*. Para que isso seja possível, é necessário resolver questões que se levantam fundamentalmente a dois níveis:

Metafísico Como conceptualizamos o mundo real e de onde nos vem a percepção sobre o mesmo?

Ontológico Que fenómenos têm significado e qual é esse significado?

Na secção 4.2, ilustramos a vantagem de se usar um *thesaurus* bem definido, que se pode utilizar em qualquer geo-web service.

Meta-informação sobre o local Resta ainda mais um atributo *Keywords* que se pode associar ao serviço (e não a cada uma das entidades). Embora o âmbito da IG disponibilizada esteja devidamente especificado através do atributo <LatLongBoundingBox>, este atributo pode usar-se para dar a localização da informação, através do nome do local[7].

Neste caso, vamos recorrer a um thesaurus conhecido, que é o *Getty Thesaurus of Geographic Names* disponível em `http://www.getty.edu/research/conducting_research/vocabularies/tgn/`.

A vantagem de usar este vocabulário garante que todos podem entender o que significa o termo utilizado para descrever a localização. No nosso caso, podemos especificar que a informação se refere à cidade de Braga, utilizando a hierarquia em que a cidade (e não o distrito) aparece no referido thesaurus, da seguinte forma:

```
<Service>
  <Name>GeoServer do Lab de SIG do DI da UMINHO</Name>
  <Title>Teste e Implementacao de OpenGIS Web Services</Title>
  <Abstract>
    Test server. Contains some data from Braga.
  </Abstract>
  <Keywords>TGN/World/Europe/Portugal/Braga/Braga</Keywords>
  <OnlineResource>
      http://siglab.di.uminho.pt:8888/geoserver
  </OnlineResource>
  <Fees>NONE</Fees>
  <AccessConstraints>NONE</AccessConstraints>
</Service>
```

Utilizamos o prefixo TGN/ para lembrar que este conceito se refere a um termo do referido thesaurus.

[7] Pode-se sempre recorrer aos serviços complementares, Geocoding e Gazetteer, para converter nomes para coordenadas e vice-versa.

4.3 Recurso a meta-informação externa

A informação geográfica disponibilizada pelos dois web services do exemplo referido na secção 4.1, poderá já estar catalogada. A meta-informação desses recursos também nos parece ser um local interessante para registar informação semântica.

Neste artigo, vamo-nos referir aos atributos existentes na norma de meta-informação geográfica ISO 19155, [10], para caracterizar a semântica dos dados. Note-se que esta informação se refere, tipicamente, a um *dataset*. Um *dataset* poderá conter informação sobre muitas entidades diferentes (rios, rede viária, limites administrativos, hidrografia, altimetria, etc). Neste contexto estamos interessados em meta-informação associada apenas a um tipo de entidade, a informação do mesmo tipo.

Na referida norma, as características semânticas são registadas através de `descriptiveKeywords`, do tipo `MD_Keywords`.

`descritiveKeywords` é uma lista de termos à qual se pode associar a proveniência (o vocabulário). O recurso a um vocabulário fechado, com significado bem definido, é a forma mais conveniente para caracterizar a semântica.

Um extracto do documento de meta-informação, utilizando `descritiveKeywords`, poderá ser:

```
<descriptiveKeywords>
  <keyword>World/Europe/Portugal/Braga/Braga</keyword>
  <type>
    <MD_KeywordTypeCode_CodeList>
      place
    </MD_KeywordTypeCode_CodeList>
  </type>
  <thesaurusName>
    <title>Getty Thesaurus of Geographic Names</title>
    <edition>Version 3.0-Web</edition>
  </thesaurusName>
</descriptiveKeywords>
<descriptiveKeywords>
  <keyword>road</keyword>
  <type>
    <MD_KeywordTypeCode_CodeList>
      theme
    </MD_KeywordTypeCode_CodeList>
  </type>
  <thesaurusName>
    <title>
      New Ontology of Geographic Information Concepts
    </title>
    <edition>Not (yet) available</edition>
  </thesaurusName>
</descriptiveKeywords>
(...)
<spatialRepresentationType>
  <MD_SpatialRepresentationTypeCode_CodeList>
```

```
    vector
  </MD_SpatialRepresentationTypeCode_CodeList>
</spatialRepresentationType>
(...)
<topicCategory>
  <MD_TopicCategoryCode_CodeList>
    transportation
  </MD_TopicCategoryCode_CodeList>
</topicCategory>
```

5 Conclusões e Trabalho Futuro

Os WS vêem trazer um contributo fundamental à IG, na medida em que permitem criar sobre a mesma, simultaneamente um nível de encapsulamento e oferecem uma forma uniformizada de aceder a essa informação.

Neste artigo são identificados dois problemas.

Em primeiro lugar, dado que estes Geo-Web Services surgiram ao mesmo tempo que os próprios Web Services, seguiram uma orientação que não é conforme à recomendação XML Web Services, actualizada pelo consórcio WS-I, [2]. Os Geo-Web Services não usam SOAP, não usam WSDL nem o UDDI. Utilizam o protocolo HTTP e definem operações específicas sobre o próprio serviço.

O segundo problema, que só faz sentido tentar resolver após ter sido criada esta plataforma de Geo-WS, está relacionado com o suporte à interoperacionalidade dos serviços com base na localização e que precisam de recorrer a novas fontes de dados geográficos (a descobrir e a explorar em tempo real), com a mínima intervenção humana. Argumentamos que esta interoperacionalidade só é possível atingir se incluirmos mecanismos que caracterizem a semântica da informação. Apontamos dois caminhos para isso. Uma possibilidade passa pela inclusão de meta-informação nos próprios Geo-Web Services, explorando o atributo `<Keyword>` já existente. Outra possibilidade passa pela existência de meta-informação externa associada à(s) entidade(s) que se pretendam conhecer. Sugere-se a utilização da norma ISO 19115 para descrever externamente a informação geográfica, e exemplifica-se a utilização dos atributos <descriptiveKeywords>. Em ambos os casos, a solução peca pela falta de uma ontologia que nos defina os termos e respectivas relações que representam conceitos da informação geográfica. Uma vez que estão envolvidas questões metafísicas e ontológicas, não será fácil produzir uma tal ontologia. Naturalmente que a urgência e necessidade de criação da mesma justifica a tentativa de a desenvolver a mesma em determinados domínios ou no âmbito de determinadas comunidades de utilizadores, onde seja mais fácil obter consensos.

Referências

1. A. Aguiar, A. Sousa, A. Rocha, A. Pires, J. Hespanha, L. Silva, and L. Galiza. Simat – relatório da especificação de requisitos. Technical report, INESC, October 1998.

2. K. Ballinger. Basic profile version 1.0a. http://www.ws-i.org/ProfilesBasic/2003-08/BasicProfile-1.0a.HTML, August 2003.
3. T. Barclay, J. Gray, S. Ekblad, E. Strand, and J. Richter. Terraservice.net: An introduction to web services. Technical Report MS-TR-2002-53, Microsoft Research, June 2002.
4. K. Buehler, S. Bacharach, C. Reed, C. Kottman, C. Heazel, J. Davidson, Y. Bisher, H. Niedzwiadek, and J. Evans. Opengis reference model. http://www.opengis.org/info/orm, April 2002. Last revision in March 2003.
5. J. Chung. Web-services computing: Advancing software interoperability. *Computer*, October 2003.
6. OpenGIS Consortium Technical Committee. The opengis guide, third edition.
7. OpenGIS. Web feature service implementation specification. http://www.opengis.org/.../02-058.pdf.
8. OpenGIS. Web map service implementation specification. http://www.opengis.org/...
9. J. Rocha. *Informação Geográfica: Meta-Informação, Codificação e Visualização*. PhD thesis, UM, 2004.
10. B. 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.

SVG WebSlide & SVG WebChart: aplicações com gráficos vectoriais escaláveis em XML

Helder Filipe P.C. Ferreira

Faculdade de Engenharia Universidade do Porto
FEUP / Portugal
hfilipe@fe.up.pt

Resumo A maior parte das linguagens XML apenas exprime informação textual, com deficientes ou inexistentes capacidades de representação gráfica. Por esse motivo a W3C[1] desenvolveu o SVG[2] (Scalar Vector Graphics), e tornou "gráfico" o XML. As potencialidades gráficas, a expansibilidade e flexibilidade inerente a qualquer linguagem XML, e a possibilidade de publicação directa na web, fizeram do SVG a linguagem ideal para criação e publicação de conteúdos com qualidade gráfica excelente e imutável, independente do tamanho da janela do browser ou da resolução do monitor.
Este artigo relata os resultados de um trabalho em desenvolvimento[3], que visou a criação de duas aplicações práticas de SVG: para criação automatizada de apresentações ou slides on-line (SVG WebSlide), e para a criação automatizada de relatórios on-line com gráficos de dados estatísticos (SVG WebChart). Ambas as aplicações motivaram a criação de duas novas linguagens XML: Presentation Markup Language como XML para slides de apresentações; e Workbook Markup Language como XML para gráficos de dados.
Palavras Chave: XML, SVG, slides, gráficos estatísticos, PresentationML, WorkbookML

1 Introdução

SVG significa Scalable Vector Graphics (Gráficos Vectoriais Escaláveis):

- **Gráficos:** A maioria das linguagens XML apenas exprime informação textual, com deficientes ou inexistentes, capacidades de representação gráfica. Por esse motivo a W3C desenvolveu o SVG, e tornou "gráfico" o XML.
- **Vectoriais:** Actualmente existem dois tipos de gráficos: raster e vectorial. Os gráficos do tipo raster baseiam-se nos pixels para representar a imagem, enquanto que os gráficos vectoriais são compostos por polígonos, linhas, pontos e curvas. Estes últimos podem também incluir imagens do tipo raster. Uma das grandes vantagens dos gráficos vectoriais é o facto de estes não sofrerem efeito de aliasing.

– **Escaláveis:** Escalamento é o acto de aumentar ou diminuir de forma uniforme. Em termos gráficos isto significa que um objecto SVG não se encontra limitado ao tamanho fixo de um pixel. Isto é, podemos aumentar ou diminuir uniformemente um objecto SVG sem que este perca definição gráfica. Uma das outras vantagens do SVG é o facto de podermos reutilizar objectos SVG dentro de um novo objecto SVG, escalando-os.

O conjunto destes três conceitos define as capacidades do SVG e justifica a sua criação como a tecnologia de gráficos para a World Wide Web.

A aplicação directa do SVG é a representação gráfica de algo. No entanto, desta representação podem surgir diversas ideias, tais como:

– Desenhos e animações em páginas web;
– Exemplos interactivos educativos;
– Slides de apresentações;
– Pesquisa virtual de edifícios e interiores de andares;
– Mapas geográficos;
– Representação gráfica de tabelas de dados estatísticos;
– Entre outros...

O objectivo deste trabalho é uma proposta de aplicações simples em SVG, com algumas sugestões ou propostas de linguagens de marcação para determinadas áreas de aplicação. Foram desenvolvidas duas aplicações práticas do SVG, visando:

– a criação automatizada de slides de apresentações (SVG WebSlide)
– e a criação automatizada de representações gráficas de tabelas unidimensionais de dados estatísticos (SVG WebChart).

Para ambas as aplicações foram desenvolvidas novas linguagens XML que compõem documentos que são processados via XSLT, usando o processador Saxon[4]. Toda a documentação, aplicações SVG e demonstrações on-line, encontram-se disponíveis em: `http://lpf-esi.fe.up.pt/~ped`.

2 SVG WebSlide: criação de slides

Uma das aplicações práticas que surgiu muito naturalmente, foi a elaboração de slides em SVG. As potencialidades gráficas e a possibilidade de publicação directa na Internet, são duas características que criam condições fantásticas para a construção de slides e a sua publicação imediata na web, com características de imagens vectoriais. Assim nasceu o SVG WebSlide.

Com esta aplicação pretende-se automatizar o processo de criação de uma apresentação dentro de moldes personalizáveis por parte dos utilizadores, através do uso de CSS[5], em que o primeiro e último slide da apresentação são gerados automaticamente, assim como um sistema de navegação entre slides.

2.1 Presentation Markup Language

A criação de slides em SVG, usando a própria linguagem do SVG, é um pouco entediante e complexa, obrigando o utilizador a ter conhecimentos algo profundos sobre determinados objectos SVG, propriedades e definições. Sendo assim, e para facilitar a construção de slides, optou-se pela criação de uma nova linguagem XML mais simples e intuitiva.

Foi realizada uma pesquisa na Internet, procurando saber se já existia uma linguagem XML própria para slides. De facto existe e chama-se SlideML[6]. No entanto, esta linguagem encontra-se ainda em desenvolvimento (não existe sequer uma DTD), e a sua definição é demasiado complexa para a aplicação simples que se procurava implementar. Sendo assim optou-se por criar uma nova linguagem XML para slides, a que se deu o nome de PresentationML. O seu conjunto de etiquetas de marcação consiste no seguinte:

2.2 Conversão do PresentationML em SVG

A transformação de PresentationML para SVG é feita através de uma stylesheet (WebSlide.XSL). Esta funciona recorrendo a chamadas de templates pré-definidos, ou aplicando templates para os elementos identificados. Quando um elemento de PresentationML é detectado, este é substituído por um conjunto de marcações SVG. O utilizador não necessita de se preocupar com o slide de rosto e o último de agradecimento, uma vez que estes são gerados automaticamente. No processo de conversão existem 3 aspectos interessantes que merecem ser mencionados:

- a necessidade de se criarem vários documentos SVG (um para cada slide);
- um mecanismo de navegação entre os slides;
- posicionamento dos elementos gráficos no slide.

Para a criação de vários documentos, para cada slide é definido um novo ficheiro de saída dentro da pasta onde estes são guardados, com o nome "slide" + posição do nó slide + ".svg". Deste modo, são sequencialmente criados ficheiros slideX.svg.

Para o mecanismo de navegação foram definidas algumas variáveis que capturavam as posições dos nós slide. Uma vez que o nome dos slides se baseava na posição, seria apenas necessário adicionar 1 à posição corrente para avançar, e subtrair 1 para recuar na apresentação.

O posicionamento dos elementos gráficos é realizado de modo diferente consoante o elemento em causa é texto ou uma imagem/forma geométrica. O texto em SVG utiliza atributos x, y, dx e dy que permitem posicionar o texto no slide de modo absoluto ou em relação ao último posicionamento de texto. As imagens ou formas geométricas são posicionadas de modo absoluto usando os atributos x e y.

A versão integral desta stylesheet pode ser vista em: `http://lpf-esi.fe.up.pt/~ped/down.html`.

```
1  Elemento: presentation : Delimita um conjunto de slides.
2  Atributos:
3  title: título da apresentação.
4  subtitle: subtítulo da apresentação.
5  author: autor(es) da apresentação.
6  date: data em que foi feita, ou vai ser apresentada.
7  organization: instituição ou organização a que o(s) autor(es) pertence(m).
8  dir: directório no qual irão ser criados os slides em SVG.

9  Elemento: slide : Define um slide.
10 Atributos:
11 subtitle: título do slide.

12 Elemento: tline : Linha de texto.
13 Elemento: blist : Lista de itens.
14 Elemento: bitem : Item de uma lista.

15 Elemento: link : Endereço URL/URI.
16 Atributos:
17 url: endereço.

18 Elemento: image : imagem do tipo JPG ou PNG ou SVG.
19 Atributos:
20 x: posição no eixo das abcissas.
21 y: posição no eixo das ordenadas.
22 width: largura da imagem.
23 height: altura da imagem.
24 url: endereço da imagem.

25 Elemento: code : Excerto de um programa ou rotina de código.
26 Elemento: br : Nova linha de texto.

27 Elemento: paint : Pinta o texto com uma determinada cor.
28 Atributos:
29 color: cor em hexadecimal ou em rgb.

30 Elemento: space : Espaçamento ou afastamento.
31 Atributos:
32 n: posição de afastamento em relação à margem esquerda.

33 Elemento: b : Negrito. i : Itálico. u : Sublinhado.
```

Tabela 1. Conjunto de etiquetas de marcação *PresentationML*.

```
<!-- PRESENTATION : elemento raíz do documento -->
<!ELEMENT presentation (slide)+>
    <!ATTLIST presentation title CDATA #REQUIRED>
    <!ATTLIST presentation date CDATA #REQUIRED>
    <!ATTLIST presentation author CDATA #REQUIRED>
    <!ATTLIST presentation subtitle CDATA #IMPLIED>
    <!ATTLIST presentation organization CDATA #IMPLIED>
    <!ATTLIST presentation dir CDATA #IMPLIED>


<!-- SLIDE : slide -->
<!ELEMENT slide (tline | blist | bitem | link | image | code | br | paint |
                 space | b | i | u)+>
    <!ATTLIST slide subtitle CDATA #REQUIRED>

<!-- TLINE : linha de texto -->
<!ELEMENT tline ( #PCDATA | link | code | br | paint | space | b | i | u )*>
<!-- BLIST : lista de itens -->
<!ELEMENT blist (bitem)+>
<!-- BITEM : item de uma lista -->
<!ELEMENT bitem ( #PCDATA | link | code | br | paint | space | b | i | u)*>

<!-- LINK : endereço URI / URL -->
<!ELEMENT link (#PCDATA)>
    <!ATTLIST link url CDATA #REQUIRED>

<!-- IMAGE : imagem -->
<!ELEMENT image EMPTY>
    <!ATTLIST image x CDATA #REQUIRED>
    <!ATTLIST image y CDATA #REQUIRED>
    <!ATTLIST image width CDATA #REQUIRED>
    <!ATTLIST image height CDATA #REQUIRED>
    <!ATTLIST image url CDATA #REQUIRED>

<!-- CODE : rotinas de código -->
<!ELEMENT code ( #PCDATA | br | paint | space | b | i | u)*>
<!-- BR : nova linha de texto -->
<!ELEMENT br EMPTY>

<!-- PAINT : pinta texto com uma determinada cor -->
<!ELEMENT paint ( #PCDATA | link | code | br | paint | space | b | i | u)*>
    <!ATTLIST paint color CDATA #REQUIRED>

<!-- SPACE : espaçamento -->
<!ELEMENT space EMPTY>
    <!ATTLIST space n CDATA #REQUIRED>

<!-- B : negrito -->
<!ELEMENT b (#PCDATA | link | code | br | paint | space | i | u)*>
<!-- I : itálico -->
<!ELEMENT i (#PCDATA | link | code | br | paint | space | b | u)*>
<!-- U : sublinhado -->
<!ELEMENT u (#PCDATA | link | code | br | paint | space | b | i)*>
```

Tabela 2. *Document Type Definition* do *PresentationML*

2.3 Exemplos de apresentações criadas pelo SVG WebSlide

Vejamos um exemplo de uma apresentação com um único slide em PresentationML:

```
<presentation title="Exemplo do WebSlide"
              author="Helder Ferreira"
                date="13-Nov-2003"
       organization="PED / MEI / FEUP">
 <slide subtitle="Slide criado por este programa">
  <tline>Exemplo do <paint color="navy"><b>SVG</b></paint>
         <i><paint color="orange">WebSlide</paint></i></tline>
  <blist>
    <bitem>
       É um aplicativo desenvolvido em
       <b>XHTML+CSS+XSL+<paint color="red">SVG</paint>+PERL</b>.
    </bitem>
  </blist>
  <tline>Mais info:
    <link url="http://www.fe.up.pt/~hfilipe">
      http://lpf-esi.fe.up.pt/~ped
    </link>
  </tline>
 </slide>
</presentation>
```

Tabela 3. Exemplo de um documento *PresentationML*

A figura seguinte mostra o slide que foi gerado a partir do documento da tabela 3.

Figura 1. Slide gerado pelo SVG WebSlide

Outros exemplos podem ser vistos na página web do SVG WebSlide, ou mesmo testados, usando a demonstração on-line em: `http://lpf-esi.fe.up.pt/~ped`.

3 SVG WebChart: criação de gráficos de tabelas de dados

O SVG WebChart é uma aplicação cujo objectivo é a geração automatizada de gráficos em SVG, a partir de tabelas de dados, definidos através de uma linguagem XML (WorkbookML), baseada no XML do Microsoft Excel. Esta aplicação apenas suporta tabelas de dados no formato descrição, dado. Isto é, tabelas do tipo:

Venda de Livros num ano Categoria Valor Policial 1.051.234 Romance 2.345.678 Poesia 345.893 Escolar 5.235.623 Tabela 4: Exemplo de tabela passível de ser representada pelo SVG WebChart.

Venda de Livros num ano	
Categoria	**Valor**
Policial	1 051 234
Romance	2 345 678
Poesia	345 893
Escolar	5 235 623

Tabela 4. Exemplo de tabela passivel de ser representada pelo SVG WebChart

Isto deve-se ao facto de ser necessária uma grande complexidade nos cálculos matemáticos que calculam o posicionamento dos elementos gráficos, legendas, etc. Uma vez que o objectivo do trabalho era apenas concretizar representações simples de gráficos estatísticos, optou-se pelo uso de tabelas com apenas uma série de valores.

O WebChart suporta diferentes tipos de gráficos: Barras, Pontos ou Linhas. E suporta diferentes tipos de direcções dos gráficos: Vertical (0^o de orientação) ou Horizontal (90^o de orientação). Nota: Assume-se que a posição natural de um gráfico de dados é a posição vertical.

3.1 WorkBook Markup Language

A criação de gráficos em SVG, usando a própria linguagem do SVG, é algo complexa porque exige um elevado número de cálculos, escalamentos, translações e rotações de objectos SVG.

Para não obrigar o utilizador a ter conhecimentos profundos sobre determinados objectos SVG, propriedades e definições, criou-se o SVG WebChart, e uma linguagem XML mais simples e intuitiva.

Uma vez que a folha de cálculo actualmente mais usada é o Microsoft Excel, optou-se por estudar o ficheiro XML gerado pelo mesmo quando se cria uma

tabela de dados. No entanto, o documento XML gerado pelo Excel é mais vocacionado para descrever visualmente a área de trabalho da aplicação e por isso contém demasiadas etiquetas de marcação desnecessárias, tendo em conta o que se pretendia neste trabalho. Sendo assim, optou-se por criar uma nova linguagem XML para gráficos, baseada no XML do Microsoft Excel, a que se deu o nome de WorkbookML. O seu conjunto de etiquetas de marcação consiste no seguinte:

```
1  Elemento: Workbook : Delimita um conjunto de gráficos.
2  Atributos:
3  * name: título do conjunto de gráficos.
4  * subtitle: subtítulo do conjunto de gráficos.
5  * logo: eventual logotipo de um relatório estatístico.
6  * logow: largura do logotipo.
7  * logoh: altura do logotipo.
8  * dir: directório no qual irão ser criados os gráficos em SVG.

9  Elemento: DocumentProperties : Define meta-informação do documento de gráficos.
10 Elemento: Author : Autor do documento.
11 Elemento: Created : Data/Hora de criação do documento.
12 Elemento: Company : Instituição/Organização à qual o autor pertence.
13 Elemento: Version : Versão do documento.

14 Elemento: Worksheet : Definição de uma folha de dados estatísticos.
15 Atributos:
16 * name: título do gráfico de dados.
17 * orientation: orientação do gráfico (vertical -- 0°, horizontal -- 90°).
18 * type: tipo de gráfico: (barras - bars, pontos - points e linhas -- lines).

19 Elemento: Table : Define um gráfico/tabela de dados.
20 Elemento: Caption : Legendas dos eixos do gráfico.
21 Elemento: Row : Linha da tabela de dados.
22 Elemento: Cell : Coluna da tabela de dados.

23 Elemento: Data : Dados da tabela.
24 Atributos:
25 * type: tipo de dado (string ou number).
```

Tabela 5. Conjunto de etiquetas de marcação *WorkbookML*

3.2 Conversão de WorkbookML em SVG

A transformação de WorkbookML para SVG é feita através de uma stylesheet (WebChart.XSL).

Uma vez que o sistema de coordenadas do SVG é complexo, assim como os cálculos necessários para construir o gráfico, optou-se por desenhá-lo na horizontal. Posteriormente, aplicam-se operações de translação e rotação para colocar o gráfico desenhado no local pretendido (centro do slide). Seja qual for a orientação e o tipo de gráfico, é sempre necessário determinar:

– a largura de cada barra que forma o gráfico (mesmo nos gráficos de pontos, pois estes também são formados por rectângulos, mas com uma área menor).

```
<!-- WORKBOOK : elemento raíz do documento -->
<!ELEMENT Workbook (DocumentProperties, Worksheet+)>
        <!ATTLIST Workbook name CDATA #REQUIRED>
        <!ATTLIST Workbook subtitle CDATA #REQUIRED>
        <!ATTLIST Workbook logo CDATA #IMPLIED>
        <!ATTLIST Workbook logow CDATA #IMPLIED>
        <!ATTLIST Workbook logoh CDATA #IMPLIED>
        <!ATTLIST Workbook dir CDATA #IMPLIED>

<!-- DOCUMENTPROPERTIES : meta-informação do documento -->
<!ELEMENT DocumentProperties (Author, Created, Company, Version)>
<!-- AUTHOR : autor do documento -->
<!ELEMENT Author (#PCDATA)>
<!-- CREATED : data/hora da criação do documento -->
<!ELEMENT Created (#PCDATA)>
<!-- COMPANY : instituição ou organização à qual o autor pertence -->
<!ELEMENT Company (#PCDATA)>
<!-- VERSION : versão do documento -->
<!ELEMENT Version (#PCDATA)>

<!-- WORKSHEET : folha de dados estatísticos -->
<!ELEMENT Worksheet (Table)>
        <!ATTLIST Worksheet name CDATA #REQUIRED>
        <!ATTLIST Worksheet orientation (0 | 90) #REQUIRED>
        <!ATTLIST Worksheet type (bars | lines | points) #REQUIRED>

<!-- TABLE : tabela de dados -->
<!ELEMENT Table (Caption, Row+)>
<!-- CAPTION : legenda dos eixos -->
<!ELEMENT Caption (Cell)+>
<!-- ROW : linha da tabela de dados -->
<!ELEMENT Row (Cell)+>
<!-- CELL : coluna da tabela de dados -->
<!ELEMENT Cell (Data)>


<!-- DATA : dados -->
<!ELEMENT Data (#PCDATA)>
        <!ATTLIST Data type (Number | String) #REQUIRED>
```

– a barra com valor mais elevado (para que se possam escalar todos os objectos SVG, principalmente os eixos e o texto);

A versão integral desta stylesheet pode ser vista em: `http://lpf-esi.fe.up.pt/~ped/down.html`.

3.3 Exemplos de apresentações criadas pelo SVG WebChart

Um exemplo de um documento WorkbookML bem-formado e válido seria:

```
<Workbook name="Relatório Anual do Restaurante «Tripas à Moda do Porto»"
          subtitle="Ano 2003">
   <DocumentProperties>
      <Author>Helder Filipe P.C. Ferreira</Author>
      <Created>27-12-2003</Created>
      <Company>PED / MEI / FEUP</Company>
      <Version>1.0</Version>
   </DocumentProperties>
   <Worksheet name="Pratos de Carne do Restaurante «Tripas à Moda do Porto»"
              orientation="0" type="bars">
      <Table>
         <Caption>
            <Cell><Data type="String">Pratos de Carne</Data></Cell>
            <Cell><Data type="String">N° de refeições servidas</Data></Cell>
         </Caption>
         <Row>
            <Cell><Data type="String">Vitela Assada</Data></Cell>
            <Cell><Data type="Number">10456</Data></Cell>
         </Row>
         <Row>
            <Cell><Data type="String">Tripas à moda do Porto</Data></Cell>
            <Cell><Data type="Number">98020</Data></Cell>
         </Row>
         <Row>
            <Cell><Data type="String">Cozido à Portuguesa</Data></Cell>
            <Cell><Data type="Number">56000</Data></Cell>
         </Row>
         <Row>
            <Cell><Data type="String">Lombo Assado</Data></Cell>
            <Cell><Data type="Number">12345</Data></Cell>
         </Row>
         <Row>
            <Cell><Data type="String">Prego no Prato</Data></Cell>
            <Cell><Data type="Number">34023</Data></Cell>
         </Row>
      </Table>
   </Worksheet>
</Workbook>
```

A figura seguinte mostra o gráfico que foi gerado a partir do documento da tabela 7.

Outros exemplos podem ser vistos na página web do SVG WebChart, ou mesmo testados, usando a demonstração on-line em: `http://lpf-esi.fe.up.pt/~ped`.

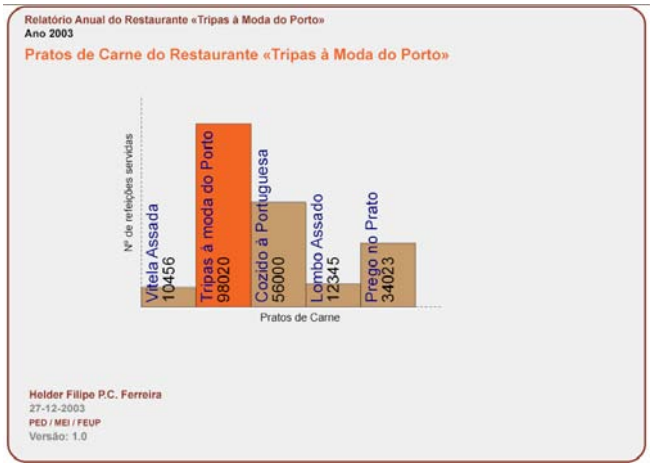


Figura 2. Gráfico gerado pelo SVG WebChart

4 Modo de funcionamento das aplicações

Foram desenvolvidos dois programas de linha de comando, que realizam todas as operações necessárias à transformação de PresentationML ou WorkbookML em SVG. Estes programas foram desenvolvidos em Perl [7], e recorrem ao processador XSLT, Saxon. Para se usar o Saxon é necessário possuir o Microsoft Java Virtual Machine [8] instalado. Para que se possam visualizar os slides criados, é recomendado o plugin da Adobe - SVG Viewer, e o uso do browser Internet Explorer. As sintaxes de comando são da forma: webslide.exe exemplo.xml e webchart.exe -help . Cada um dos programas realiza a seguinte sequência de operações:

- Leitura do documento PresentationML ou WorkbookML.
- Testa a presença do elemento raíz: Presentation ou Workbook.
- Usa o módulo de Perl, XML::Parser[9] para validar o documento XML.
- Pesquisa pelo atributo dir do elemento Presentation ou Workbook, de modo a descobrir qual a pasta onde os slides devem ser criados. Se o atributo não tiver sido colocado ou estiver em branco, ele assume por defeito que a pasta denomina-se “slides”, no caso do WebSlide, ou “charts”, no caso do WebChart.
- Cria a pasta para guardar os ficheiros criados.
- Modifica todos os endereços para ficheiros externos, de modo a que estes incluam a referência para o directório onde vão ser criados os slides.
- Copia o ficheiro CSS e imagens que são usadas na apresentação para esse directório.
- Utiliza o Saxon para realizar a transformação com o WebSlide.XSL ou o Webchart.XSL[10].
- Cria os slides e retorna sucesso ao autor.

5 Conclusões e Perspectivas Futuras

O desenvolvimento destas duas aplicações veio:

- Facilitar a criação de apresentações on-line, com uma qualidade vectorial, e por essa razão legíveis e acessíveis em qualquer resolução ou tamanho;
- Simplificar a criação de apresentações em SVG e permitir a reutilização de slides entre apresentações, uma vez que basta um simples copy & paste de uma apresentação para outra.
- Facilitar a criação de gráficos de dados estatísticos, embora ainda limitados na dimensionalidade das tabelas.
- Estabelecer uma proposta de linguagens de marcação XML para a criação de slides e de gráficos, que até ao momento, ou eram inexistentes ou demasiado complexas.
- Fornecer meios de transformação SVG de distribuição gratuita.

Actualmente estas aplicações encontram-se na versão 1.0, e com algumas limitações, principalmente o SVG WebChart. Planeia-se no futuro, melhorar as capacidades de ambas as aplicações. Para o SVG WebSlide, prevê-se um melhoramento na quantidade de objectos a suportar, assim como um aumento no grau de personalização dos slides. Para o SVG WebChart, prevê-se um aumento do tipo de gráficos a suportar, assim como o suporte para tabelas de dados multidimensionais.

Referências

[1] World Wide Web Consortium (W3C) - `http://www.w3c.org`.
[2] Scalable Vector Graphics (SVG) - `http://www.w3.org/Graphics/SVG`.
[3] SVG WebSlide e SVG WebChart - `http://lpf-esi.fe.up.pt/~ped`.
[4] Saxon - `http://saxon.soundforge.net/`.
[5] Cascading Style Sheet (CSS) - `http://www.w3.org/Style/CSS/`.
[6] SlideML - `http://www.slideml.org`.
[7] Practical Extraction and Report Language (Perl) - `http://www.perl.org` e `http://www.perl.com`.
[8] Microsoft Java Virtual Machine - `http://java-virtual-machine.net/download.html`.
[9] XML::Parser - `http://search.cpan.org/~msergeant/XML-Parser-2.34/Parser.pm`.
[10] Stylesheets que transformam PresentationML ou WorkbookML em SVG - `http://lpf-esi.fe.up.pt/~ped/down.html`.

Parte IV

Desenvolvimento de Aplicações e Intercâmbio de Informação

Desenvolvimento Estruturado de Aplicações Web com Cocoon

Márcio Ferreira, João Pias, Célio Abreu, and Rui Alberto Golçalves

PT inovação — {jpias,celio,rui-l-goncalves}@ptinovacao.pt
Telbit — mferreira@telbit.pt

Resumo Pretende-se com este documento apresentar os principais conceitos utilizados na framework de publicação Apache Cocoon. Não é pretendido explanar de forma detalhada a framework, mas sim apresentar as metodologias subjacentes à tecnologia.

1 Introdução

Inicialmente a WWW[1] foi criada como um meio de publicação de conteúdos estáticos. No entanto o conteúdo que é devolvido por um servidor aplicacional não necessita de ser obrigatoriamente estático, poderá ser o resultado da execução de um programa, produzindo respostas de forma dinâmica.

O Apache Cocoon é uma framework de publicação, desenvolvida na linguagem Java, baseado na Servlet API, podendo ser instalado em qualquer servlet container, elevando a utilização das tecnologias XML[2] e XSLT[3] para um novo nível no desenvolvimento de aplicações Web. As aplicações desenvolvidas nesta framework são de grande performance e escalabilidade, pois baseiam-se no encadeamento de eventos SAX[4] e mecanismos de cache, diminuindo assim os tempos de resposta. Os custos de desenvolvimento e manutenção são também mais reduzidos por existir uma separação clara entre lógica e apresentação.

Esta framework é baseada em componentes, sendo por isso bastante modular. Como esses componentes geram e/ou consomem eventos SAX, podem ser encadeados numa sequência para produzir os resultados pretendidos. Uma cadeia de componentes é designada por **pipeline**. O conceito de pipeline é semelhante ao conceito já existente nos sistemas unix, mas nos pipelines do cocoon são passados eventos SAX. Existe um conjunto de componentes genéricos prontos a ser utilizados num pipeline, no entanto o programador é livre de criar os seus próprios componentes para tarefas mais específicas.

2 Sitemap

O sitemap é o ficheiro de configuração principal do cocoon, pois é onde se define o fluxo de uma determinada aplicação. Conforme a figura1 que se segue,

[1] World Wide Web
[2] eXtended Markup Language
[3] eXtensible Stylesheet Language
[4] Simple Api for XML

o sitemap está dividido em duas secções, componentes e pipelines, dos quais passaremos a falar. Definem-se pipelines para cada tipo de operação que se pretenda realizar, o fluxo dentro de um pipeline é bem definido e é determinado pelos matchers, estes componentes permitem encaminhar o fluxo mediante as necessidades da aplicação. No exemplo apresentado apenas existe um pipeline com um matcher, desta forma sempre que exista um pedido, por exemplo, http://servidor/cocoon/sendmail, o matcher encaminha o processamento para o bloco correspondente, permitindo assim que a tarefa para a qual foi desenvolvida seja realizada.

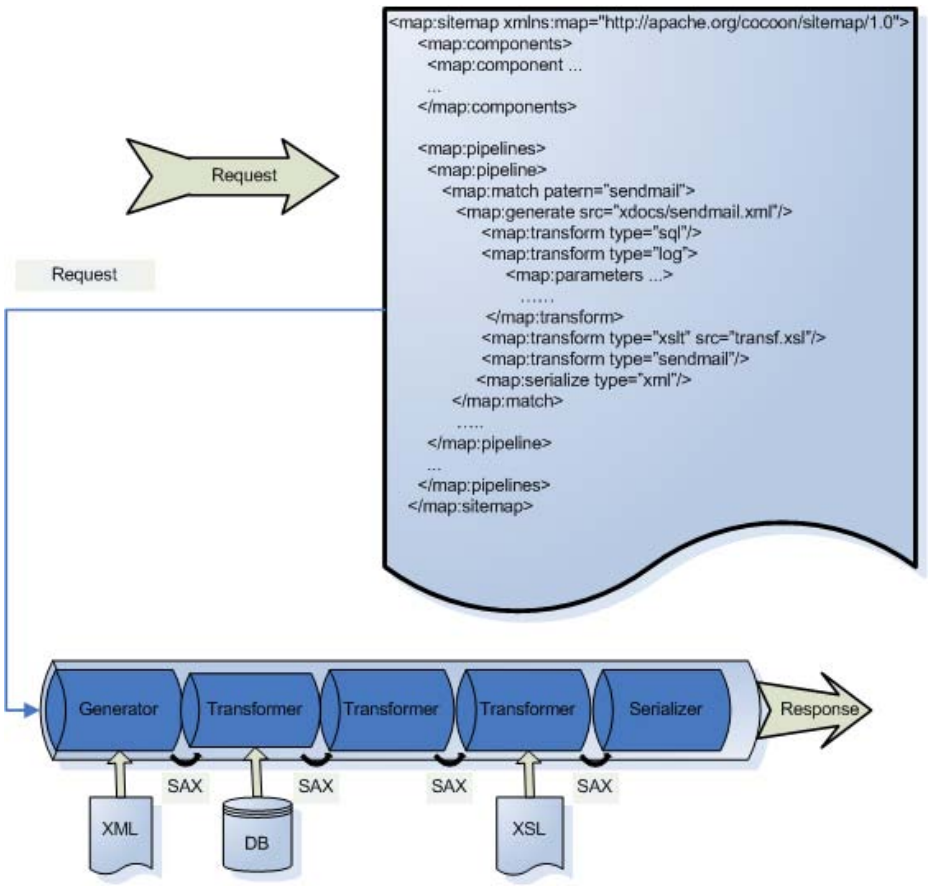


Figura 1. Funcionamento de um Pipeline

2.1 Componentes

Um componente é um bloco de software que pode ser combinado com outros de forma a realizar a tarefa que se pretende. Na secção dos componentes são declarados todos os componentes necessários para a construção dos pipelines. São geridos pelo Avalon Excalibur Component Manager, o qual é responsável pela gestão das pools, configuração e parametrização dos mesmos.

Generators Este componente aceita qualquer coisa à entrada e gera uma sequência válida de eventos SAX, que são encaminhados para o próximo componente existente no pipeline. Este tipo de componente é utilizado na inicialização de um pipeline de forma isolada ou conjuntamente com outros generators para produzir um documento que é o resultado da agregação de cada um dos documentos produzidos pelos generators isoladamente.

Transformers Componente opcional num pipeline que pode ser usado depois de um generator ou um transformer. Recebe como input eventos SAX e lança novos eventos SAX para o próximo componente no pipeline.

Um transformer não precisa de processar o documento inteiro, apenas precisa de focar a informação necessária á tarefa para a qual foi desenvolvido. Se o documento contém elementos pertencentes ao namespace conhecido pelo transformer corrente, este avalia essa informação e age para obter conteúdos adicionais ou para manipular o documento de outras formas. Por exemplo se o generator receber o seguinte documento XML:

```
<document>
 <sql:execute-query xmlns:sql="http://apache.org/
                                  cocoon/SQL/2.0">
   <sql:use-connection>
     conf.conn@cocoon.xconf
   </sql:use-connection>
   <sql:query>
     select id, name from users_table
   <sql:query>
 </sql:execute-query>
</document>
```

que especifica um statement SQL para uma base de dados, usando o SQL Transformer, extrai a query SQL, executa-a e insere o resultado no documento. É possível que um componente gere comandos para o componente que se segue no pipeline. Por exemplo, o SQL Transformer fez uma query a base de dados e o seu resultado pode ser usado, usando uma transformaąão (XSLT), para formar um comando para o próximo transformer na cadeia, como por exemplo o mail transformer, para enviar um e-mail ao administrador se algo correu mal.

Serializers Todos os pipelines tem obrigatoriamente que terminar com um serializer. Recebe como input eventos SAX e serializa-os num determinado formato, dependendo do tipo de serializer.

Matchers A função de um matcher é avaliar se o URI[5] que está a ser pedido deverá ser o resultado da execução do fluxo que matcher encapsula. Um matcher age como uma fechadura para um pipeline, só deixa passar se a chave o abrir. O sitemap é percorrido usando o URI que foi pedido como chave, essa chave é testada em cada um dos matchers até que seja encontrado um match para a respectiva chave. Uma vez que seja encontrado, é executado o respectivo pipeline. Por essa razão é importante a ordem pela qual se colocam os pipelines no sitemap. Matchers mais específicos devem ser colocados no inicio do sitemap. Se o URI não fizer match em qualquer pipeline é retornado um erro para o cliente.

Exemplo da utilização de um matcher:

```
<map:match pattern="today-*/*.xml"/>
  ....
</map:match>
```

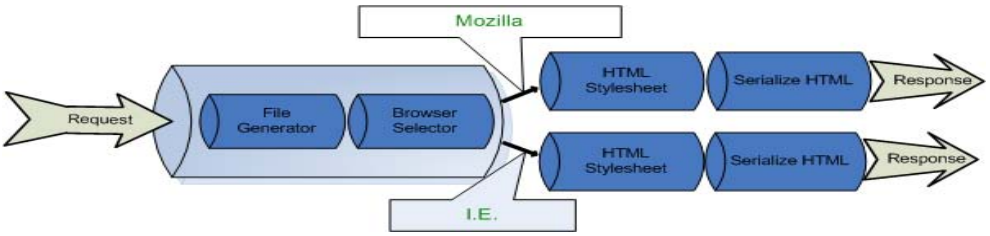


Figura 2. Funcionamento de um Selector

Selectors Tal como os matchers os selectors são componentes do sitemap que podem ser usados para determinar o fluxo atravéz do sitemap. Essas decisões poderão ser tomados com base em informações quer do cliente, quer do sistema.

O exemplo mais comum de um selector é o browser selector, que permite optar por fluxos distintos dependendo do tipo de cliente.

```
<map:generate src="calls.xml">
<map:select type="browser">
  <map:when test="explorer">
    <map:transform src="">
```

[5] Uniform Resource Identifier

```
      <map:serialize/>
    </map:when>
    <map:when test="mozilla">
      <map:transform src=""/>
      <map:serialize/>
    </map:when>
    <map:otherwise>
      <map:transform src=""/>
      <map:serialize/>
    <map:otherwise>
  </map:select>
```

Note-se que o ficheiro lido não é dependente do browser, é o mesmo para todos os clientes. Isto permite uma verdadeira separação entre informação e apresentação, dando ao sitemap a mesma flexibilidade que temos nas linguagens de programação estruturada.

Actions Os componentes até agora apresentados partilham uma funcionalidade comum, influenciam o resultado do pedido. No entanto quando se está a desenvolver uma aplicação é necessário efectuar tarefas que não influenciam directamente o documento, como por exemplo se quiseremos, na construção da nossa aplicação adicionar informação à sessão ou mesmo consultar alguma informação da mesma, é aqui que entram os actions. Vários actions são disponibilizados pelo cocoon para acesso a informação dos pedidos, como por exemplo actions que permitem verificar informacão existente nos cookies para autorizar acesso a conteudos num portal. Um action pode também controlar o fluxo e fornecer informação a outro pipeline.

```
<map:act type="session-isvalid">
  <map:generate type="file" src="file.xml"/>
  <map:transform src="rules.xsl" type="xslt"/>
  <map:serialize type="html"/>
</map:act>
<map:read src="timeout.html"/>
```

O exemplo anterior define um action chamado session-isvalid que dá indicação do estado da sessão. Caso a sessão seja válida, então é retornado o HTML correspondente a transformação do documento file.xml com a stylesheet rules.xsl, senão será retornado o documento timeout.html.

Readers Já se falou de componentes para processar XML ou para controlar o fluxo de processamento, no entanto as aplicações necessitam frequentemente de documentos que não são XML-based, como imagens, filmes, javascript. Estes

documentos devem ser servidos para o cliente tal como são, sem qualquer tipo de manipulação. Os readers são usados para estes casos, em que o documento já se encontra no formato pretendido, apenas lê a fonte e passa a informação directamente à aplicação. Pode ser visto como a combinação de um generator especial e um serializer.

Exemplo típico da utilização de um reader:

```
<map:match pattern="*/*.gif">
  <map:read src="{1}/{2}.gif"/>
</map:match>
```

2.2 Pipelines

É considerado um pipeline uma sequência de componentes entre os quais fluem eventos SAX. Exemplo de definição de um pipeline num sitemap:

```
<map:pipelines>
  <map:pipeline>
    <map:match pattern="*.csv">
      <map:read type="xsp"
                src="generators/xsp/{1}.xsp" mime-type="text/csv"/>
    </map:match>

    <map:match pattern="*.xsp">
      <map:generate src="generators/xsp/{1}.xsp" type="xsp"/>
      <map:serialize/>
    </map:match>

    <map:match pattern="*.html">
      <map:read src="static/html/{1}.html" mime-type="text/html"/>
    </map:match>
  </map:pipeline>

  ...
</map:pipelines>
```

Um sitemap é composto por um conjunto de pipelines, cada pipeline contém um conjunto de matchers e cada matcher encapsula um fluxo de eventos.

Tratamento de Erros Durante o processamento de um pipeline poderão ocorrer excepções que impossibilitem a execução correcta do pedido. O mecanismo de error handling oferecido pelo cocoon permite que seja executado um fluxo distinto dependendo do tipo de erro.

```
<map:handle-errors>
  <map:select type="exception">
    <map:when test="connection-refused">
      <map:read src="connection_refused.html"
                mime-type="text/html"/>
    </map:when>
  </map:select>
</map:handle-errors>
```

Mecanismos Para Debug No caso de o output retornado não ser o que se esperava o que fazer? O cocoon disponibiliza mecanismos que permitem efectuar o debug de forma simples.

– Usando o LogTransformer

```
...
  <map:transform type="sql"/>
  <map:transform type="log">
    <map:parameter name="logfile" value="logfile.log"/>
    <map:parameter name="append" value="no"/>
  </map:transform>
...
```

Neste exemplo, todos os eventos gerados pelo SQLTransformer serão registados no ficheiro definido no parâmetro logfile, o parâmetro append, define se deve ser criado um novo ficheiro ou se deverá ser concatenado ao existente. Se não for definido nenhum parâmetro o output será enviado para o standard output.

– Views
Uma view é um pipeline alternativo para um documento, começa como o original mas termina de modo diferente, ou seja, não é definido um generator mas podem-se definir transformers e um serializer. O atributo from-position permite definir o inicio do novo pipeline.

```
<map:view name="debug"
          from-label="debug">
  <map:serialize type="xml"/>
</map:view>
```

Para que seja executada uma determinada view o utilizador deve efectuar o request passando na query string do URL cocoon-view=debug.

2.3 Subsitemaps

No desenvolvimento de uma aplicação podem estar envolvidas várias pessoas, executando tarefas distintas o que torna a gestão do sitemap complicada. Para facilitar a edição e manutenção do sitemap o Cocoon disponibiliza o conceito de subsitemaps. Um subsitemap é igual a um sitemap normal mas é incluido por outro.

```
<map:match pattern="faq/*">
 <map:mount check-reload="yes" src="faq/sitemap.xmap"
   reload-mathod="synchron"/>
</map:match>
```

O atributo src define a localização do sitemap, o check-reload define se as alterações devem ser reflectidas e o reload-method especifica se a regeneração do sitemap deve ser síncrona ou assíncrona. Os componentes declarados num sitemap são herdados pelos subsitemaps.

3 Conclusão

Com o Cocoon é possível construir aplicações sem que haja preocupações com o problema do crescimento. Pelo facto de o cocoon permitir um desenvolvimento bastante modular, a adição de novas funcionalidades apenas implica a adição de novos pipelines, quer seja num sitemap existente ou num novo sitemap. Esta modularidade e separação de funcionalidades permite que o processo de desenvolvimento seja efectuado de forma paralela entre as equipas de desenvolvimento, sem que posteriormente venham a existir problemas de integração.

Os mecanismos de cache existentes no cocoon permitem aumentar o desempenho das aplicações por ele suportadas.

Devido à diversidade de componentes oferecidos pela framework, dificilmente será necessário recorrer ao desenvolvimento de componentes específicos, no entanto essa possibilidade é também deixada em aberto para casos mais particulares.

Para além de todas as vantagens já enumeradas, o cocoon herda todas as vantagens das tecnologias que lhe estão subjacentes.

4 Referêcias

[AA03] The Apache Avalon Project. http://avalon.apache.org/, 1997-2003

[CA03] The Apache Cocoon Project. http://cocoon.apache.org/2.1/, 1999-2003

[HM01] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*, O'Reilly, Janeiro 2001.

[Wl01] Heather Williamson. *XML: the complete reference*, Osborne/McGraw-Hill, 2001.

[MC02] Matthew Langham and Carsten Ziegeler. *Cocoon: Building XML Applications*, New Riders, Jun 2002.

Interoperabilidade entre Sistemas de Informação Universitários

Sérgio Sobral Nunes and Gabriel David

MGI/FEUP — mgi01016@fe.up.pt
FEUP/INESC-Porto — gtd@fe.up.pt

Resumo Neste artigo discute-se a interoperabilidade entre sistemas de informação universitários. Depois de uma revisão das principais iniciativas existentes na área, é apresentada uma análise aos tipos de interoperabilidade existentes num contexto universitário. São definidas 3 tipologias: agregação, concentração e difusão de informação. No contexto da agregação de informação é apresentada uma proposta de especificação em XML Schema para o registo académico do aluno. O trabalho apresentado faz parte de um projecto mais abrangente, que visa dotar a UP de infraestruturas que permitam a integração das diferentes instituições ao nível dos sistemas de informação.

Palavras-chave: Sistemas de Informação Universitários, Interoperabilidade, Agregação de Informação, XML Schema, Registo Académico do Aluno.

1 Introdução

Os sistemas de informação têm fronteiras que delimitam o seu âmbito. No caso dos sistemas de informação universitários, o contexto abrangido é vasto e resulta de uma organização complexa e fortemente descentralizada [15].

No entanto, tal como a própria organização, os sistemas de informação necessitam de interagir com outros sistemas externos. É essencial a possibilidade de troca de registos de alunos entre instituições, a consulta de informação institucional, a disponibilização de currículos ou estatísticas de desempenho. A estes processos de partilha e reutilização de informação e procedimentos entre sistemas, está associado o termo "interoperabilidade". Beynon-Davies define interoperabilidade como "uma medida do grau segundo o qual os sistemas de informação são capazes de se coordenar e colaborar" [2].

A declaração de Bolonha [11], assinada em 1999 pelos ministros da educação dos países da UE, veio reafirmar a importância estratégica de uma aposta na interoperabilidade entre os sistemas de informação das instituições de educação europeias. Dos objectivos traçados, destacam-se a aposta na promoção da mobilidade de alunos, docentes, investigadores e outro pessoal, e a aposta na cooperação entre as instituições.

Neste artigo, descreve-se o trabalho efectuado nesta área tendo por base o estudo dos mecanismos de interoperabilidade associados à partilha de informação

académica relativa aos alunos. Na secção seguinte, enumeram-se as principais iniciativas no âmbito da interoperabilidade entre sistemas de informação no domínio da educação. Na secção 3, apresenta-se uma abordagem à estruturação dos diferentes tipos de interacção identificados. Na secção 4, e no contexto da agregação de informação, apresenta-se uma proposta para a representação do Registo Académico do Aluno usando XML Schema. No final (secção 5), apresentam-se as conclusões sobre o trabalho já desenvolvido e descrevem-se os planos para o trabalho futuro.

2 Iniciativas Existentes no Contexto Académico

São várias as iniciativas a nível mundial que exploram a colaboração entre instituições do ensino superior [1]. Nesta secção, faz-se uma breve apresentação de algumas das principais iniciativas existentes.

2.1 ANSI ASC X12A

O ACS X12 é um comité, certificado pela ANSI em 1979, que visa o desenvolvimento de normas para a implementação de transacções electrónicas entre empresas de uma mesma indústria. As normas definidas pelo ACS X12 são utilizadas para a definição da sintaxe, estrutura e ordenação das mensagens utilizadas nas referidas transacções e são designadas por *Electronic Data Interchange* (EDI).

O subcomité X12A [19] tem como âmbito de trabalho a administração escolar. As iniciativas levadas a cabo pelo X12A compreendem todos os níveis de escolaridade, desde o primário ao universitário, bem como casos especiais (privado, profissional ou outro). As normas já desenvolvidas abrangem a área administrativa, o registo académico dos alunos, informações relativas aos recursos humanos, a ajuda financeira e a gestão curricular. São os formatos mais usados nos EUA para a transferência de informação, maioritariamente administrativa, entre instituições académicas.

2.2 Postsecondary Electronic Standards Council

A *Postsecondary Electronic Standards Council* (PESC) [6] é uma organização dos EUA, constituída por cerca de 50 membros (instituições académicas e empresas comerciais), e que visa a promoção do uso de normas electrónicas na educação e a respectiva implementação. Destacam-se as actividades desenvolvidas no âmbito do grupo *XML Forum* [8], que se posiciona como o grupo responsável pelas normas XML na área da educação. Um dos principais objectivos do PESC é o suporte à transição de tecnologias e mecanismos baseados em EDI para XML.

Nos documentos já publicados por este grupo, inclui-se a Especificação Técnica em XML para o Ensino Superior [9] e um conjunto de livros brancos sobre uso de chaves públicas, identificadores de estudantes e XML.

Algumas das normas já existentes e aplicadas no meio académico, em particular as que são definidas pelo ACS X12, estão a ser usadas como ponto de

partida para o trabalho desenvolvido pelo *XML Forum*. No conjunto de tarefas agendadas, está incluída a conversão para XML de algumas das especificações em X12. Recentemente, foi publicada a versão 1.0 do *XML Postsecondary Transcript Schema* [7].

2.3 Sistema de Codificación Académica Normalizado en Red

O *Sistema de Codificación Académica Normalizado en Red* (SCANet) [17] é um projecto iniciado em 2001, na Universidade Espanhola de Lérida, que visa o desenvolvimento de normas para a representação e transferência de informação no contexto da gestão académica. O projecto tem como objectivo "tornar mais fáceis as relações entre as universidades, destas com os estudantes e docentes, trazendo valor acrescentado aos processos de gestão académica".

O projecto está estruturado em 3 fases: definição de códigos de identificação únicos, especificação de requisitos para transferência de conteúdos electrónicos e harmonização das formas e procedimentos.

Actualmente o projecto conta com a participação de cerca de 50 universidades espanholas e com a colaboração do Governo Central Espanhol na elaboração de regulamentos administrativos.

2.4 Internet2 Middleware Initiative

A *Internet2 Middleware Initiative* (I2-MI) [14] é um projecto enquadrado na Iniciativa Norte Americana Internet2, que visa a promoção e desenvolvimento de serviços e infra-estruturas que permitam a integração entre aplicações no contexto universitário e comunidades relacionadas. Esta iniciativa abrange áreas como segurança, partilha de informação, autenticação e uso de chaves públicas.

Dentre as várias iniciativas desenvolvidos no contexto da I2-MI, destacam-se os projectos eduPerson, eduOrg [10] e DoDHE [13]. Os dois primeiros têm como objectivo a definição de objectos LDAP para a representação de indivíduos e organizações no contexto académico. O projecto *Directory of Directories for Higher Education* (DoDHE) insere-se no estudo de tecnologias que suportem a integração inter-institucional de pesquisas em directórios.

2.5 Schools Interoperability Framework

O *Schools Interoperability Framework* (SIF) [18] é um consórcio empresarial dos EUA que, desde 1997, procura desenvolver uma especificação para a integração de aplicações de âmbito educacional. A especificação publicada permite, por exemplo, a interoperabilidade entre aplicações relacionadas com a gestão de matrículas de alunos, a gestão da cantina ou a emissão de cartões. A especificação desenvolvida baseia-se na norma XML do W3C e a versão mais recente disponível é a 1.1. Este trabalho está direccionado para o ensino pré-universitário e tem uma forte adesão por parte dos principais fabricantes de aplicações.

A nível europeu, o projecto OASIS - *Open Architecture and Schools in Society* [16], enquadrado no programa de apoio às Tecnologias de Informação na

Sociedade (IST), procura adaptar o trabalho desenvolvido pelo SIF ao contexto europeu [3].

3 Interoperabilidade entre Sistemas de Informação Universitários

Nesta secção, apresenta-se uma abordagem para a estruturação dos padrões de interacção encontrados no contexto dos sistemas de informação universitários. Com base nos processos de partilha de informação identificados e na natureza das instituições de ensino superior, definiram-se dois grandes eixos de interoperabilidade: horizontal e vertical (Figura 1).

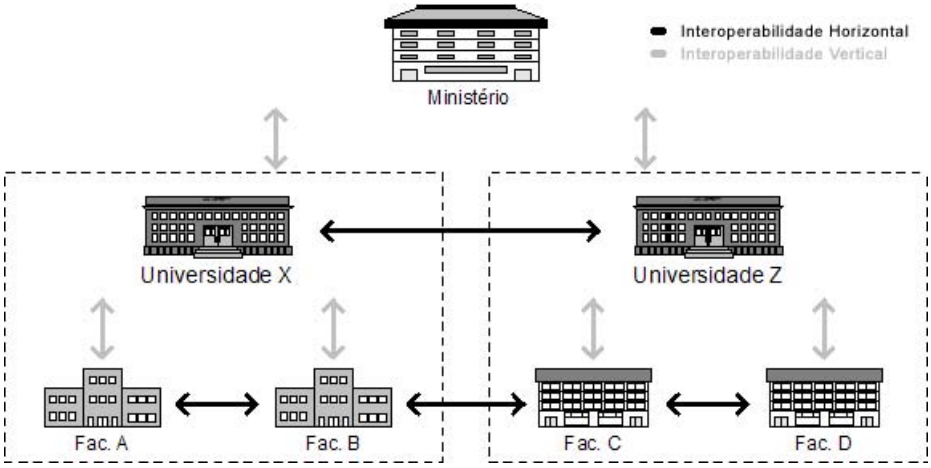


Figura 1. Interoperabilidade Vertical e Horizontal

A designação **Interoperabilidade Horizontal** identifica os fluxos entre as instituições académicas no mesmo nível administrativo. Insere-se neste âmbito a cooperação entre instituições, nomeadamente na implementação de cursos ou projectos multi-disciplinares. Neste contexto, a partilha de dados de uma forma automática constitui um factor importante; Um cenário concreto é o dos cursos partilhados por várias instituições, em que a informação sobre os conteúdos das disciplinas deve existir nos diversos SI.

A **Interoperabilidade Vertical** corresponde aos fluxos existentes entre as instituições académicas em níveis administrativos diferentes. Referem-se, a título exemplificativo, as trocas de informação entre uma faculdade e os orgãos centrais da universidade, ou entre um ministério e a reitoria de uma universidade.

No meio académico, os fluxos existentes segundo o eixo vertical são mais frequentes do que aqueles entre instituições ao mesmo nível. Este padrão resulta da

natureza hierárquica, mas descentralizada, das estruturas académicas. O poder está disperso por unidades independentes, que recorrem a orgãos centrais com funções administrativas e coordenadoras [15].

As interacções estudadas foram agrupadas segundo 3 padrões:

Agregação: existe nos cenários em que informação dispersa é resumida num ponto central. Exemplo: compilação de estatísticas relativas a cursos e disciplinas.

Concentração: é semelhante à agregação, mas neste caso pretende-se replicar centralmente informação dispersa por várias instituições e não resumi-la através de indicadores estatísticos. Neste caso, é necessário considerar a coordenação das várias entidades envolvidas. Exemplo: reunião, na Reitoria, de informação relativa aos recursos humanos da Universidade.

Difusão: neste caso, as interacções ocorrem no sentido contrário, ou seja, da Reitoria para as faculdades. Exemplo: difusão de notícias, produzidas na Reitoria, por toda a Universidade.

Cada um destes tipos de interoperabilidade coloca diferentes problemas ao nível da definição da arquitectura e implementação da solução, quer pelas tecnologias envolvidas, quer pelos procedimentos de transferência induzidos em cada um dos cenários (p.e. sincronização a pedido *versus* sincronização automática e imediata).

4 Registo Académico do Aluno

Nesta secção, e no contexto da interoperabilidade horizontal, apresenta-se uma proposta para a representação do Registo Académico do Aluno usando XML Schema.

4.1 Apresentação do Contexto

O Registo Académico do Aluno reúne informação sobre todas as actividades académicas de um aluno, ao longo do seu percurso no ensino superior. Este registo acompanha sempre o aluno e é usado, por exemplo, para enviar informação sobre a sua situação, em caso de transferência entre instituições académicas. Normalmente, este registo não existe sob a forma de documento único; está disperso por vários documentos existentes em diferentes serviços.

A existência de um documento electrónico, escrito de acordo com regras bem definidas e que inclua todos os dados considerados essenciais, permite normalizar e automatizar o processo de transferência de informação. A normalização acontece porque existe uma especificação para o registo académico do aluno que funciona como "gramática" e permite o entendimento entre parceiros diferentes. É possível automatizar o processo porque, devido à existência deste documento "bem definido", os dados podem ser processados sem intervenção humana. A automatização pressupõe uma normalização prévia.

4.2 Cenários de Aplicação

Troca de registos entre faculdades De forma análoga à que se verifica com o registo académico do aluno em papel, as escolas podem solicitar o envio de registos de alunos em formato electrónico. Este pedido pode efectuar-se durante o processo de transferência do aluno e o envio do documento electrónico pode ser feito em suporte físico (disco) ou por transferência electrónica (Internet). O registo académico do aluno pode ser gerado de forma automática por um sistema de informação ou produzido manualmente, de acordo com a especificação.

Troca de registos no contexto de programas de mobilidade internacional Durante um processo de intercâmbio enquadrado nos programas de mobilidade (p.e. ERASMUS), existe necessidade de trocar informações sobre os alunos entre as faculdades envolvidas (necessariamente de países diferentes). Esta informação, um subconjunto daquela que é necessária para o registo académico completo do aluno, pode ser produzida de forma semelhante.

4.3 XML Schema

Optou-se por usar a linguagem *XML Schema*, publicada pelo W3C [4], para a representação dos dados. Face à principal alternativa - *Document Type Definition* (DTD), a opção é justificada com base nos seguintes argumentos:

- O XML Schema permite usar *namespaces*, o que possibilita uma melhor estruturação dos documentos.
- O XML Schema oferece uma gama de tipos de dados base mais abrangente do que a que é oferecida pelos DTD. Com XML Schema é ainda possível definir tipos próprios.
- O XML Schema permite um maior controlo na validação dos documentos.
- O XML Schema tem um mecanismo de reutilização de dados, baseado em conceitos orientados a objectos (OO) mais sofisticado do que o que é oferecido pelos DTD.
- Os documentos em XML Schema são documentos XML, enquanto que os DTD são expressos usando outra sintaxe (EBNF). Quando se usam DTD, é necessário dominar duas linguagens distintas.
- O XML Schema é usado e suportado por várias organizações.

A mesma opção foi tomada no contexto de outras iniciativas, como é o caso no PESC [9].

4.4 Especificação para o Registo Académico do Aluno

O esquema proposto pelo PESC para a representação de informação sobre estudantes [7] foi usado como ponto de partida para o trabalho aqui apresentado. Esta norma destina-se prioritariamente à partilha de dados e é dirigida a um número significativo de instituições. Assim, procurou-se relevar a semântica da informação e eliminar ambiguidades ao nível dos tipos de dados usados.

Seguiu-se o paradigma *"Venetian Blind Design"*, apresentado em [5] e utilizado em [9], que foca o desenvolvimento na utilização de tipos de dados reutilizáveis.

A raiz da especificação é o elemento `StudentRecord`, que representa um Registo Académico de Aluno (Figura 2). Este elemento é composto por dois subelementos obrigatórios: `DocumentInfo` e `Student`. O elemento `Note`, que pode não existir ou existir várias vezes, representa uma anotação.

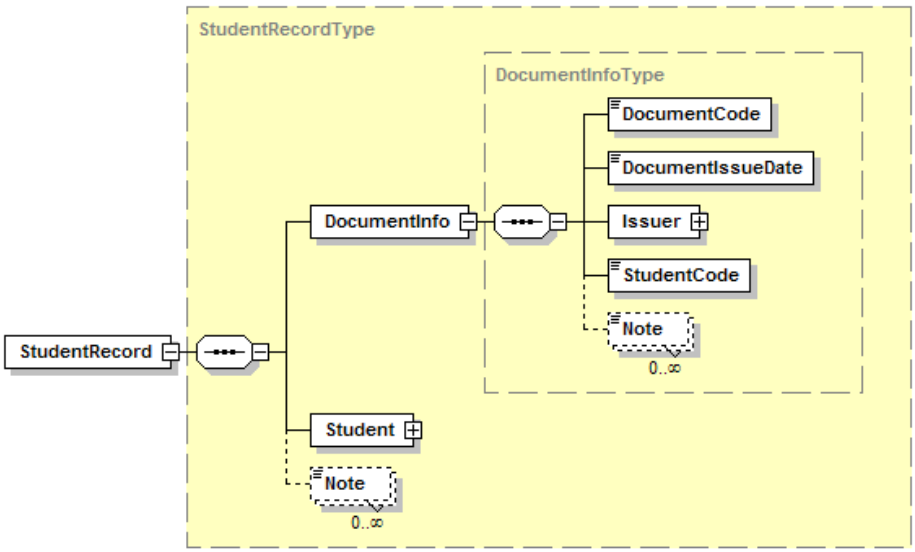


Figura 2. Elementos `StudentRecordType` e `DocumentInfoType`

O elemento `DocumentInfo` é do tipo `DocumentInfoType` e reúne informação sobre o documento (Figura 2). Este elemento é composto por 4 subelementos obrigatórios: referência interna do documento, data de criação, entidade responsável pelo documento e número do aluno definido pela entidade emissora.

Reunindo a informação da instituição (`Issuer`) e o identificador interno de cada aluno (`StudentCode`), é possível identificar de forma universal e inequívoca o aluno.

O elemento `Student` é do tipo `StudentType` (Figura 3), que define a estrutura para a informação relativa a um estudante, e inclui a informação base sobre a pessoa (elemento do tipo `PersonType`), a filiação do aluno, o conjunto de provas de admissão efectuadas para candidatura ao ensino superior, um conjunto de graus académicos que o aluno possui ou no qual está matriculado, o registo de saúde e a situação militar do aluno.

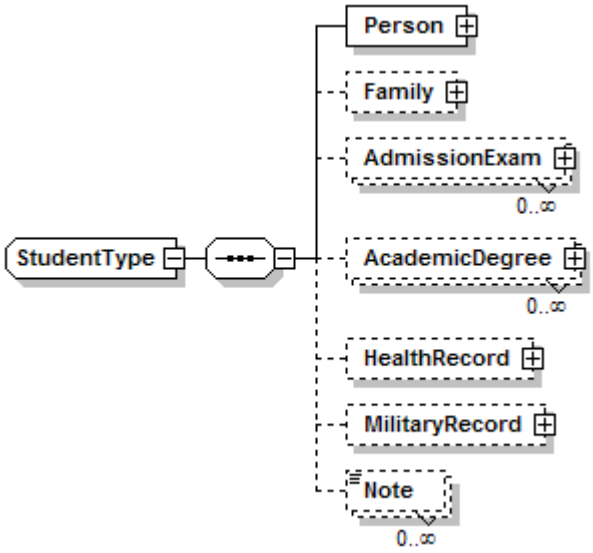


Figura 3. Elemento `StudentType`

O elemento `AcademicDegreeType` (Figura 4) representa uma inscrição num grau académico e inclui:

- `Degree`: Informação sobre o grau académico, do tipo `DegreeType`.
- `StudentCode`: Código do aluno na instituição responsável pelo grau.
- `DegreeAdmissionDate`: Data de matrícula no grau académico.
- `DegreeStudentStatus`: Estado da matrícula no grau académico (a frequentar, permutado, concluído, etc).
- `DegreeConclusionDate`: Data de conclusão do grau académico. A não existência deste subelemento significa que o grau não foi terminado.
- `DegreeAcademicGradeAverage`: No caso do curso estar concluído, representa a média final do aluno.
- `DegreeECTSGradeAverage`: A média final do aluno na escala *Sistema Europeu de Transferência e Acumulação de Créditos* (ECTS) [12].
- `AcademicSession`: Representa uma sessão académica do tipo `AcademicSessionType`. Cada sessão académica corresponde a um período lectivo.
- `AcademicAward`: Informação sobre prémio(s) académico(s) obtido(s).
- `ExtraActivity`: Informação sobre actividade(s) extra-curriculare(s) de relevância.

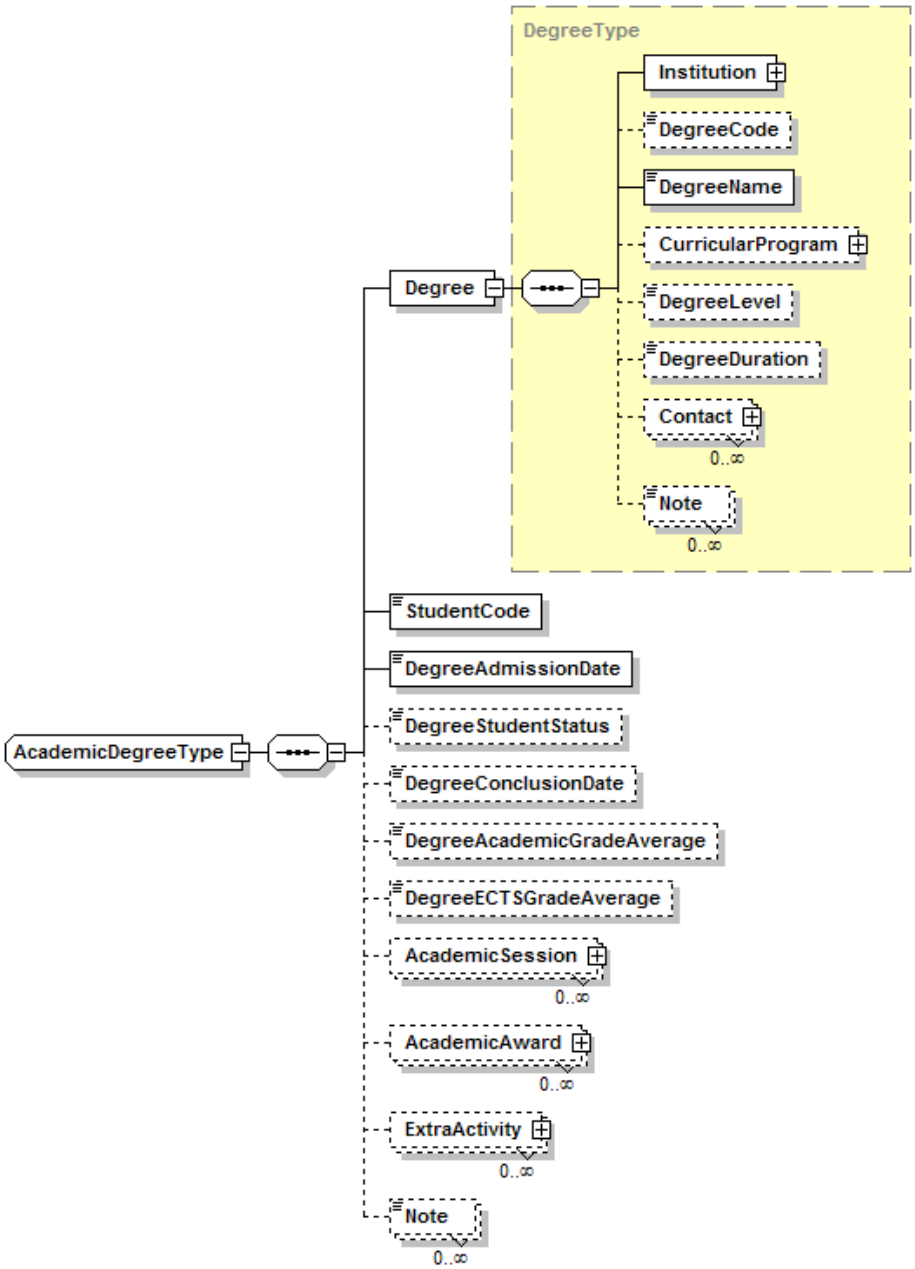


Figura 4. Elementos `AcademicDegreeType` e `DegreeType`

O `DegreeType` (Figura 4) reúne informação sobre o grau académico e inclui:

- `Institution`: Informação sobre a instituição responsável pelo curso.
- `DegreeCode`: Código do grau académico, atribuído pela instituição.
- `DegreeName`: Nome do grau académico.
- `CurricularProgram`: Referência para o plano de estudos. O código de cada plano é definido pela instituição.
- `DegreeLevel`: Nível académico deste grau, os valores possíveis são: bacharelato, licenciatura, pós-graduação, mestrado e doutoramento. Quando nenhum destes valores for adequado, o elemento não deve ser incluído.
- `DegreeDuration`: Duração oficial deste grau académico.
- `Contact`: Conjunto de contactos do curso: morada, página Web, endereço de correio electrónico, etc.

Como já foi referido, cada período lectivo do aluno é representado pelo elemento `AcademicSessionType` (Figura 5). Este elemento contém:

- `AcademicSessionStartDate`: Data de início do período lectivo.
- `AcademicSessionEndDate`: Data de conclusão do período lectivo.
- `Course`: Conjunto de disciplinas em que o aluno se inscreveu durante o período lectivo. Cada disciplina é do tipo `CourseType`.
- `Thesis`: Informação sobre uma eventual dissertação realizada no âmbito deste período e que inclui: título da dissertação, nome do(s) supervisor(es) e avaliação.

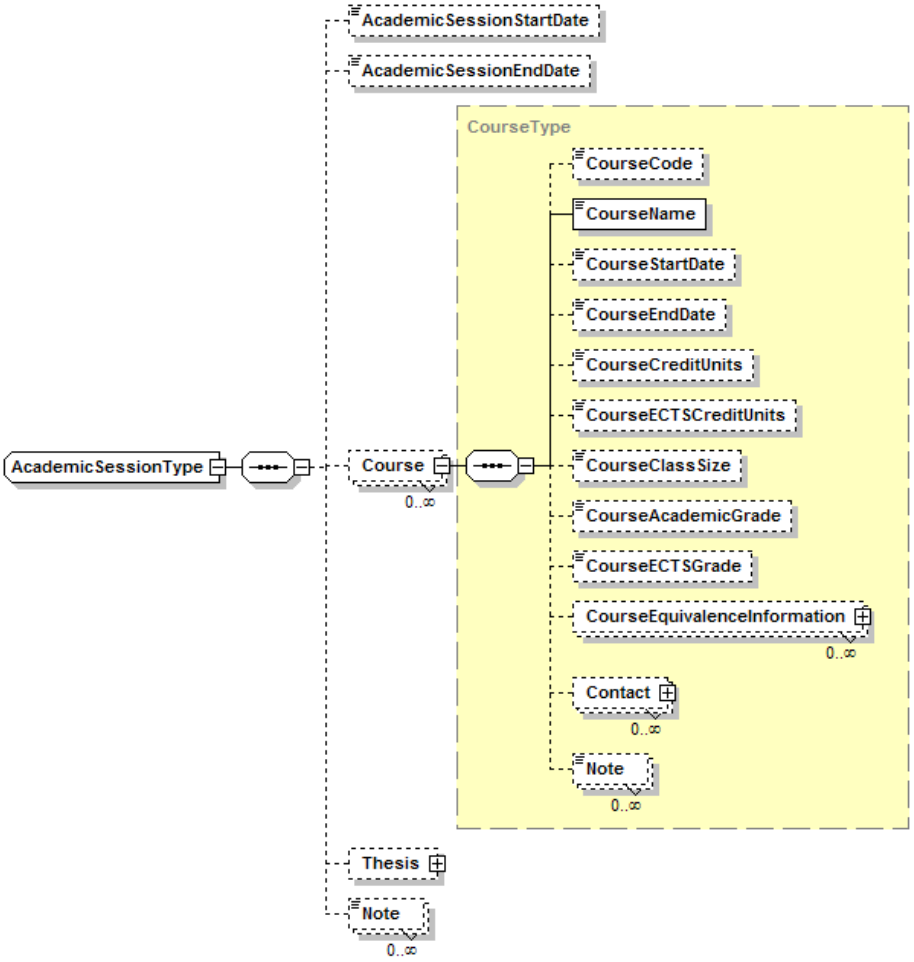


Figura 5. Elementos `AcademicSessionType` e `CourseType`

Cada disciplina é representada pelo elemento `CourseType` (Figura 5) e inclui: código atribuído pela instituição, nome, data de início, data de conclusão, unidades de crédito, unidades ECTS, número de alunos inscritos, classificação do aluno segundo a escala da instituição, classificação do aluno na escala ECTS, contactos e, caso exista, informação sobre a atribuição de equivalência.

4.5 Implementação

No protótipo já desenvolvido para a Faculdade de Engenharia da UP, a arquitectura usada é a que se apresenta na Figura 6.

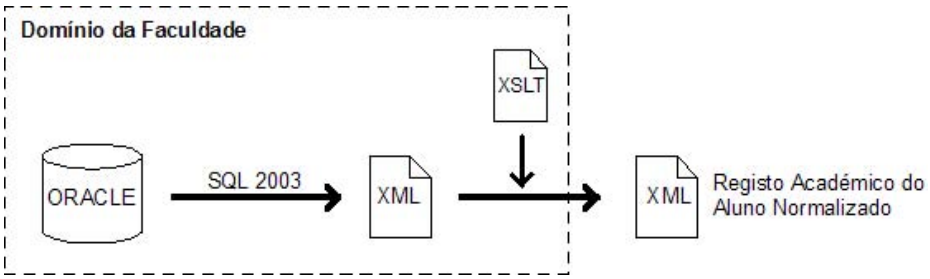


Figura 6. Arquitectura da solução implementada na FEUP

O sistema de informação da FEUP é construído sobre o SGBD Oracle. Utilizando as funções disponíveis para a produção de XML a partir de interrogações SQL à base de dados, foi possível extrair um documento XML com os dados necessários. Aplicando a esta saída uma transformação XSL produziu-se o documento final, segundo a norma especificada.

5 Conclusões e Trabalho Futuro

Neste artigo, apresentou-se uma norma direccionada para o problema concreto de agregação de informação em sistemas de informação universitários. A construção de um protótipo permitiu validar, numa primeira instância, a abordagem seguida.

Estão associados à criação desta norma para a representação do registo académico do aluno, o estudo e a definição de procedimentos relacionados com a autenticidade, a integridade e a confidencialidade dos dados. Esta fase de estudo está actualmente em curso.

Este trabalho enquadra-se num projecto mais abrangente que é o desenvolvimento de mecanismos de suporte à interoperabilidade, nos seus 2 eixos e 3 vertentes, entre os diversos sistemas de informação existentes na Universidade do Porto.

Nos planos de trabalho futuro inclui-se a análise e implementação dos casos de estudo relacionados com a concentração de informação relativa aos recursos humanos da UP e a disseminação de notícias publicadas pela Reitoria.

Referências

1. Paul Bacsich, Andy Heath, Paul Lefrere, Paul Miller, and Kevin Riley. The Standards Fora for Online Education. *D-Lib Magazine*, 5(12), 1999. `http://www.dlib.org/dlib/december99/12miller.html`.
2. Paul Beynon-Davies. *Information Systems - An Introduction to Informatics in Organizations*. Palgrave, Bath, 2002.

3. CEN/ISSS. Interoperability frameworks for exchange of information between diverse management systems. *Learning Technology eBrochure*, (5), jan 2003. http://www.cenorm.be/cenorm/businessdomains/businessdomains/information%societystandardizationsystem/elearning/learning+technologies+workshop/ebrochur%enumber5.pdf.
4. World Wide Web Consortium. XML Schema Part 0: Primer. Technical report, World Wide Web Consortium, 2001. W3C Recommendation – http://www.w3.org/TR/xmlschema-0/.
5. Roger L. Costello. Global versus Local. http://xfront.com/GlobalVersusLocal.html.
6. Postsecondary Electronic Standards Council. PESC - home page, 2003. http://www.standardscouncil.org.
7. Postsecondary Electronic Standards Council. PESC - XML Postsecondary Transcript Schema, 2003. http://www.standardscouncil.org/XMLPostTranscript.asp.
8. Postsecondary Electronic Standards Council. XML Forum for Education, 2003. http://www.pescxml.org.
9. Postsecondary Electronic Standards Council. XML Technical Specification for Higher Education. Technical report, Postsecondary Electronic Standards Council, may 2003. http://standardscouncil.org/docs/XML%20Forum%20Tech%20Specification%20%v%202.5.doc.
10. EDUCAUSE and Internet2 eduPerson Task Force. eduPerson Object Class, 2003. http://www.educause.edu/eduperson/.
11. União Europeia. Declaração de Bolonha, 1999. http://www.bologna-berlin2003.de/pdf/bologna_declaration.pdf.
12. União Europeia. Sistema Europeu de Transferência e Acumulação de Créditos de Curso (ECTS), 2003. http://europa.eu.int/comm/education/programmes/socrates/ects_pt.html.
13. Internet2/MACE. Directory of Directories for Higher Education, 2003. http://middleware.internet2.edu/dodhe/.
14. Internet2/MI. Internet2 Middleware, 2003. http://middleware.internet2.edu.
15. Henry Mintzberg. *Estrutura e Dinâmica das Organizações*. Publicações Dom Quixote, Lisboa, 1995.
16. OASIS. Open Architecture and Schools in Society, 2003. http://oasis.cnice.mecd.es.
17. SCANet. SCANet, 2003. http://www.scanet.udl.es.
18. SIF. Schools Interoperability Framework, 2003. http://www.sifinfo.org.
19. ASC X12. X12A - Education Administration Purpose and Scope, 2003. http://www.x12.org/x12org/subcommittees/sc_home.cfm?strSC=A.

Administração e monitorização de uma solução Disaster Recovery

André Vila Cova, José Ruivo, and Tomás Pereira

PT Inovação

http://www.ptinovacao.pt

Resumo O Portal NGIN Disaster Recovery surge no contexto de uma solução Disaster Recovery como módulo de monitorização e administração de todo o sistema. Essa monitorização deve ser intuitiva e permitir ao administrador do sistema uma análise gráfica eficaz. A navegação na ferramenta de monitorização deve ser efectuada a partir de um diagrama genérico, representativo do sistema e para cada servidor deverá ser possível monitorizar estatísticas actuais ou anteriores. O Portal deverá comunicar com os outros módulos da solução para garantir homogeneidade e possibilitar a administração de toda a solução.
Com este artigo pretende-se apresentar um caso prático de utilização de XML e tecnologias associadas, nomeadamente Scalable Vector Graphics (geração de gráficos e diagramas) e Web Services (comunicação entre módulos e recolha de dados dos servidores). Saliente-se que esta ferramenta é actualmente utilizada por uma operadora de telecomunicações internacional.
Área de Aplicação: Disaster Recovery
Palavras-chave: Disaster Recovery, Portal, XML,SVG, Web Services, JSP

1 Introdução

A solução NGIN da PT Inovação, para redes fixas ou móveis, é um sistema IN (Inteligent Network) com elevado grau de complexidade e que disponibiliza serviços sofisticados (como telefonia pré-paga, por exemplo) numa plataforma com hardware e software redundante. O núcleo do sistema é uma base de dados em tempo real cujo conteúdo é o contexto de cliente como utilizador do serviço (saldo, ciclo de vida, etc). A solução NGIN Disaster Recovery surge no contexto destas plataformas e o Portal NGIN Disaster Recovery como módulo de administração e monitorização.

O Portal NGIN Disaster Recovery deverá ser capaz de mostrar toda a informação sobre um determinado servidor do sistema através de gráficos e valores numéricos, funcionando como um registo on-line de toda a actividade. Deverá ser possível aceder a qualquer registo existente desse servidor através de um histórico. Será fundamental para o Portal a implementação de gráficos usando uma das várias alternativas para geração dinâmica recorrendo aos valores existentes na base de dados.

O portal mostrará o processo de replicação em funcionamento e permitirá obter conclusões sobre o mesmo, nomeadamente os recursos despendidos.

A tecnologia de suporte do portal é Java nomeadamente Java Server Pages (JSP)[1] que é uma especificação da Sun Microsystems que combina Java e HTML para fornecer conteúdo dinâmico para páginas Web. Todas as especificações e alternativas são influenciadas pela tecnologia Java.

2 Armazenamento de informação

O processo de armazenamento de informação dos servidores na base de dados é crítico. Numa arquitectura multicomputacional a recolha centralizada de informação estatística envolve mecanismos de comunicação eficientes. Os Web Services foram a alternativa proposta para comunicação entre um conjunto de servidores com o serviço de recolha de informação activo e um servidor cliente onde existe a instância da base de dados central. Para a implementação de Web Services foi utilizada uma implementação SOAP (Simple Object Access Protocol) da Apache — Axis. Neste passo torna-se requisito a instalação do XML Axis.

É simples e eficiente o desenvolvimento de um Web Service. É suficiente criar um serviço (neste caso em Java) que seja responsável pela execução local dos comandos. Este serviço deverá existir em todos os servidores do sistema. Deverá implementar métodos distintos para os diferentes comandos a executar. Neste serviço específico todos os métodos vão retornar uma sequência de caracteres alfanuméricos. O passo seguinte será desenvolver um cliente que invocará o serviço e o método pretendido que englobará os parâmetros correspondentes. O cliente invoca o serviço que devolverá uma resposta, que será armazenada num ficheiro. Esse ficheiro será importado posteriormente para a base de dados onde será registada a informação num formato específico (Figura 1).

Utilizando o Axis, para este caso específico, não temos de nos preocupar com a WSDL (Web Service Description Language) - linguagem para a descrição de serviços de rede como uma colecção de pontos terminais capazes de trocar mensagens, nem com a estrutura da mensagem SOAP, que servirá de comunicação entre o serviço e o cliente (baseada em XML).

3 Geração de gráficos e diagramas

Para a implementação de gráficos e diagramas com informação existente na base de dados foi usada a tecnologia Scalable Vector Graphics (SVG) , um formato de ficheiro que descreve um gráfico vectorial em XML. O SVG possui uma série de vantagens relativamente a formatos gráficos em uso como JPEG ou GIF. Por ter um formato de texto simples, os ficheiros SVG são legíveis e, geralmente, menores do que os outros formatos de imagens gráficas. As imagens SVG possuem recursos de zoom e são escaláveis, ou seja, os utilizadores podem aproximar uma área

[1] JSP é parte do Java 2 Enterprise Edition (J2EE)

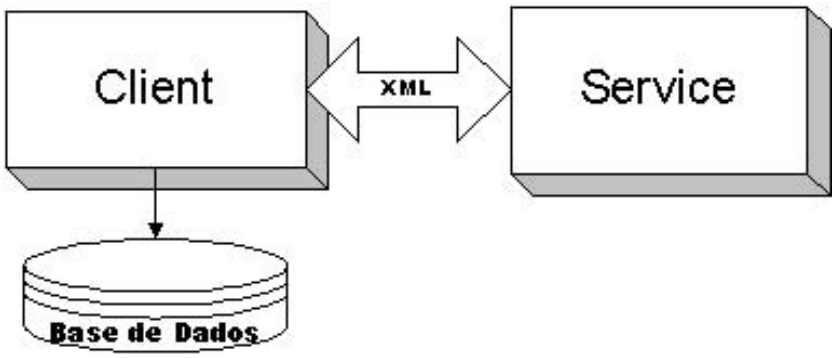


Figura 1. Web Service - Carregamento de tabelas

em particular de um gráfico, como por exemplo um mapa, e não ter nenhuma degradação de imagem. Por serem escaláveis, as imagens do SVG podem ser impressas em alta qualidade e em qualquer resolução. Textos dentro de uma imagem baseada em SVG, como o nome de uma cidade num mapa, podem ser tanto seleccionados ou procurados. Finalmente, o SVG suporta scripting e animação, o que permite gráficos dinâmicos e interactivos.

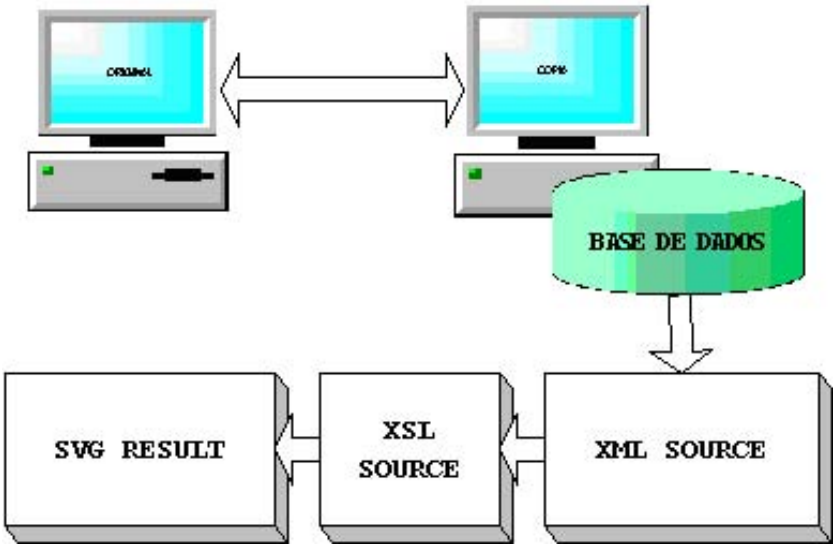


Figura 2. Estratégia de Implementação

Depois de existir a informação na base de dados é necessário arranjar uma estratégia de geração dinâmica de gráficos e diagramas (Figura 2).

3.1 Documento XML fonte

O primeiro passo consiste em gerar um documento XML de acordo com o schema específico.

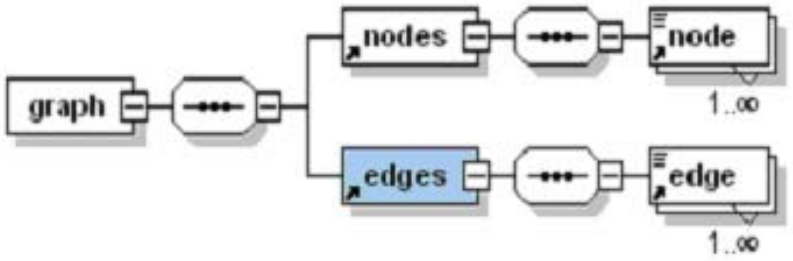


Figura 3. Schema do diagrama

- node — define um nodo no diagrama - representa o servidor (se o atributo type tiver o valor active significa que o servidor está activo, o atributo name define o nome do servidor e os atributos x e y representam as coordenadas de localização do servidor no diagrama).
- edge — define a ligação (o atributo from representa o servidor origem e o atributo to define o servidor destino).

- item — representa cada um dos pontos do gráfico
- label — define o label do ponto
- value — valor associado ao ponto

Existem várias alternativas para gerar um documento XML através de valores existentes na base de dados: SQL Transformer (Apache Cocoon), Oracle XML DB (para uma solução dependente de Oracle tem imensas funcionalidades) e através da escrita para ficheiro do resultado de uma query associada aos respectivos elementos XML.

3.2 Transformação XSL

A implementação da stylesheet (XSL) deverá estar de acordo com o resultado SVG pretendido. As stylesheets de geração de gráficos devem ser capazes de gerar dois tipos de gráficos: linhas e colunas. Esses gráficos devem suportar

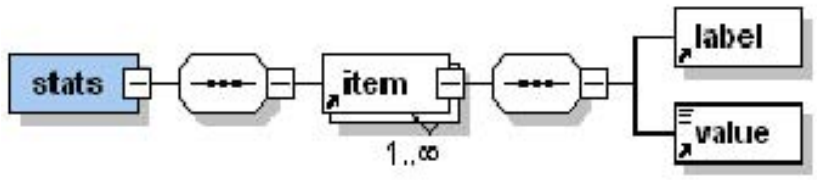


Figura 4. Schema do gráfico

valores negativos e grandes oscilações. O valor máximo a representar limita a construção de todo o gráfico.

A stylesheet para geração de diagramas deve ser capaz de representar servidores e respectivas ligações em estrutura circular e de acordo com um sistema de pontos pré-definidos.

Para aplicar uma transformação XSL ao documento XML fonte são necessários quatros objectos: um objecto que implementa a interface `javax.xml.transform.Source` para o documento XML fonte; um objecto que implementa a interface `javax.xml.transform.Result` que espera pelo resultado da transformação; um objecto que implementa a interface `javax.xml.transform.Source` para o documento XSL; um objecto da classe `javax.xml.transform.Transformer` que faz a transformação. O objecto Transformer do package `javax.xml.transform` tem um método designado transform(...) que recebe como argumentos os objectos genéricos Source (XML file) e Result(SVG file). O objecto transformer espera por uma referência ao objecto que representa o documento XSL, que usa para transformar o objecto XML Source no objecto Result.

Uma alternativa interessante é disponibilizada no pacote Oracle XML DB. A função XMLTRANSFORM aplica uma transformação XSL a um documento XML. Essa XSL pode bem ser o conteúdo de uma coluna de uma tabela do tipo XMLTYPE. Esta alternativa garante a total dependência de ferramentas Oracle e é interessante no ponto de vista da solução por não gerar nenhum ficheiro XML (documento XML on-fly).

3.3 Resultados SVG

As imagens seguintes representam um diagrama representativo da arquitectura (Figura 5) e um gráfico de linhas (Figura 6) gerado dinamicamente a partir de valores existentes na base de dados.

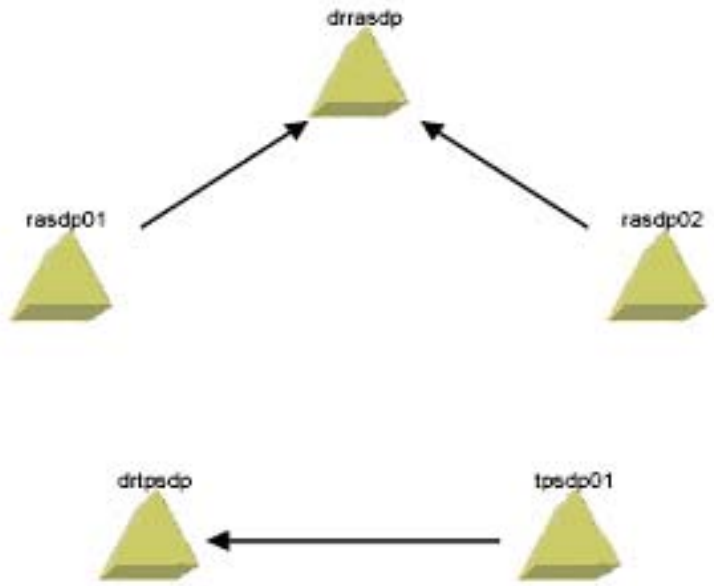


Figura 5. Diagrama representativo dos servidores

3.4 Integração com HTML

Para integração no portal o ficheiro SVG é embebido em HTML através da tag `<object>`:

```
<object  type="image/svg+xml"
         data="documento.svg"
        width="332" height="227" align="top">
   <img src="imagem.jpg" width="332" height="227"
        alt="Drawing on the Net" align="top" />
</object>
```

No exemplo o documento SVG é especificado no atributo data. Caso o browser não suporte formato SVG é colocada a imagem especificada na tag ¡img¿. Para isso é necessário ter o documento em formato SVG convertido para formato de imagem comparativa (PNG, JPEG, GIF,...). Existem várias alternativas sendo aconselhada a ferramenta SVG Rasterizer[2].

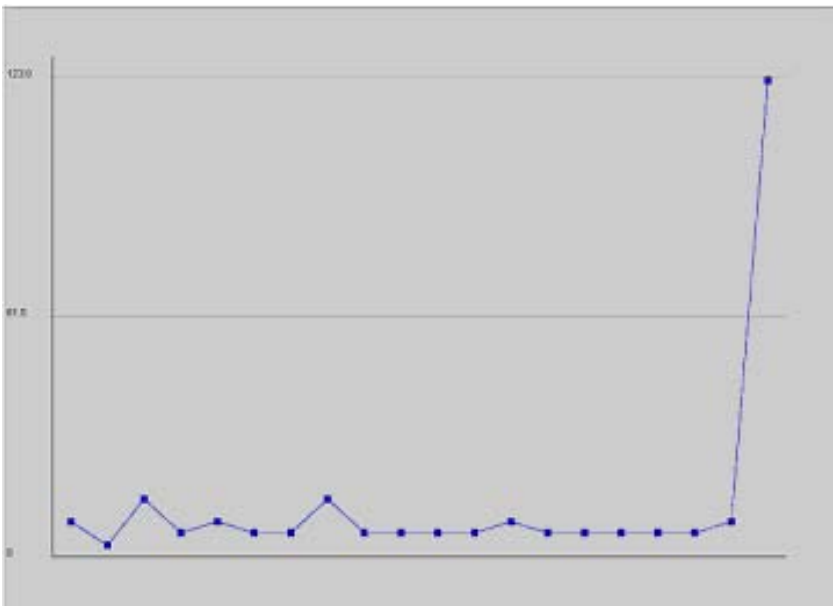


Figura 6. Gráfico de linhas

4 Ferramenta de administração

O Portal como ferramenta de administração envia pedidos para execução de tarefas a uma outra aplicação da solução designada Watchdog (responsável pela vigilância dos servidores e execução de comandos) (Figura 7). A alternativa para a comunicação entre ambas as aplicações é baseada também em Web Services, nomeadamente na implementação SOAP da Apache — Axis. É possível a comunicação entre uma aplicação desenvolvida em Java (Java Server Pages) e uma aplicação implementada na linguagem C++.

[2] integrada na distribuição Batik

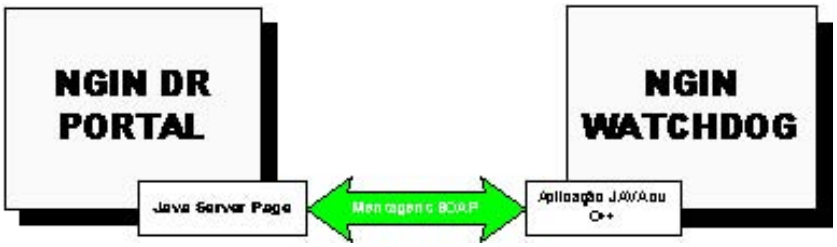


Figura 7. Comunicação Portal/WatchDog

5 Conclusão

A integração de tecnologia XML para a implementação do portal Disaster Recovery assim como na definição de camadas de comunicação entre aplicações e servidores garante a eficácia, intuitividade e interactividade da solução.

SVG apresenta inúmeras vantagens relativamente aos outros formatos de imagens gráficas, nomeadamente, capacidade de zooming, tamanho, resolução e interactividade.

A heterogeneidade dos Web Services revelou-se importante na construção de mecanismos de execução de "comandos remotos" e na comunicação entre o portal e a aplicação WatchDog. A utilização da implementação SOAP da Apache (Axis) facilitou a implementação dos Web Services, retirando complexidade na construção das mensagens SOAP e na definição de uma linguagem descritiva do serviço (WSDL).

Bibliografia

[1] Eisenberg, J., — SVG Essentials, 1st Ed, O'Reilly, 2002.
[2] Duffey, K., Goyal, V., Husted, T., LavandowskaL., Narayana, S., Perrumal, K., Walnes, J., Huss, R., Kunnumpurath, M. — Professional JSP Site Design, 1st edition, Wrox Press Inc, November 2001.
[3] Ramalho,J.,Henriques,P. — XML & XSL da teoria à prática , 1st ed, FCA,2002.
[4] Meyer, E., — Cascading Style Sheets , 1st Ed,O'Reilly,2000.
[5] McLaughlin,B., — Java & XML , 2nd Ed,O'Reilly,2001.

Parte V

XML e Metainformação

Aquisição e Armazenamento de Metainformação no Contexto de um Arquivo

Miguel Ferreira
mferreira@dsi.uminho.pt
DSI/UM and José Carlos Ramalho
jcr@di.uminho.pt
DI/UM

No Institute Given

Resumo Neste artigo, apresentam-se as várias etapas que levaram à construção dum repositório de metainformação no contexto de um arquivo histórico digital.
O XML surge naturalmente neste projecto pois a comunidade arquivística internacional adoptou há já algum tempo a linguagem de anotação EAD (*"Encoded Archival Description"*) para a descrição de metainformação arquivística e todo o intercâmbio de informação entre os vários actores (arquivos, tribunais, notários, paróquias, e outros) é feito em XML segundo a norma EAD.
Apesar do EAD existir no domínio arquivístico há alguns anos, este projecto encerrou alguns desafios interessantes. Em primeiro lugar foi preciso verificar de que maneira se poderia aplicar o EAD à realidade portuguesa. Em segundo lugar, o EAD segue uma estrutura hierárquica, natural no XML e natural também na descrição arquivística, mas o modelo a implementar teria de ser relacional. Assim houve que criar um modelo relacional que reflectisse a hierarquia da informação que seria armazenada.
Ao nível da aplicação de gestão de metainformação houve também que resolver o mesmo problema, pretendia-se manipular hierarquicamente informação que estava armazenada relacionalmente.
Estes e outros pontos satélites, como sendo a aquisição de metainformação e a transformação dos sistemas legados existentes, são discutidos no artigo.

1 Introdução

Quem já visitou, e se deteve pacientemente no serviço de referência de um Arquivo Histórico, sabe a confusão com que normalmente se depara. Múltiplos índices, livros de listagens, inventários, catálogos e guias de transferência que foram sendo elaborados ao longo do tempo, fruto de valioso trabalho mas, apesar disso, multiformes e sem coerência colectiva. Para pôr um fim definitivo a este cenário, foi desenvolvido no Arquivo Distrital do Porto (ADP) o projecto DigitArq, um projecto com múltiplas frentes, mas com um objectivo fundamental: servir de primeira abordagem à edificação de um *Arquivo Digital*.

Uma das componentes basilares deste projecto consistiu na conversão de materiais descritivos existentes somente em papel para formatos digitais normalizados baseados em normas internacionais. Assim, após a digitalização dos materiais, recorreu-se ao auxílio de software de reconhecimento óptico de caracteres (OCR) de forma a obter texto bruto capaz de ser tratado digitalmente. Após esta fase, o contributo de técnicos especializados em arquivística foi fundamental, de forma a corrigir alguns erros residuais da fase de digitalização, e para auxiliar em todo o processo de anotação do texto resultante. Uma vez o texto anotado, a sua conversão para formatos XML normalizados foi trivial. Diversos scripts transformadores foram desenvolvidos, que quando aplicados aos diferentes documentos resultaram em XML baseado na norma EAD (Encoded Archival Description).

Outra das componentes deste projecto consistiu na retro-conversão de bases de dados, agora obsoletas ou desadequadas, para um modelo de dados baseado na norma EAD. A título de exemplo, podemos mencionar a existência de materiais digitais armazenados em documentos Word, Excel, Access, XML, Arqbase/ISIS, etc.

Após as conversões descritas foi detectada uma quantidade assinalável de material em formato EAD. Tornou-se, pois, essencial, armazenar as várias centenas de ficheiros resultantes da migração num repositório de informação centralizado que permitisse, não só, instalar organizadamente esses ficheiros, como também, permitir o acesso à informação neles contida. Uma ferramenta que satisfizesse esses requisitos, mas que também, assistisse o operador na produção e manutenção de novas descrições arquivísticas tornou-se assim necessária. Nesse sentido, foi criada uma aplicação de descrição arquivística que reúne um número considerável de funcionalidades com o objectivo de facilitar e apoiar a produção de descrições normalizadas.

Ao longo deste artigo descreve-se com mais detalhe as várias fases do projecto. Na próxima secção (sec. 2), tenta caracterizar-se o contexto em o XML surge neste projecto. Na secção seguinte (sec. 3), discutem-se alguns requisitos dos futuros utilizadores que tiveram um forte impacto na solução proposta. A secção 4 é onde se apresenta o modelo desenvolvido para suportar o repositório de metainformação. Nesta secção, também se apresenta a interface criada para manipular o modelo de dados. Na secção 5 discute-se a implementação da aplicação e para terminar o artigo (sec. 6), apresentam-se algumas conclusões e pontos ainda por resolver.

Sobre o repositório de dados, foi também desenvolvido um motor de pesquisa acessível via *Web*. Uma ferramenta de trabalho indispensável aos utentes do Arquivo, especialmente àqueles que residem além-fronteiras e que não possuem disponibilidade para se deslocarem fisicamente às instalações do Arquivo.

2 O XML ao longo do projecto

Um dos principais requisitos do projecto DigitArq, para além da recuperação e centralização de toda a informação que se encontrava distribuída por diversas

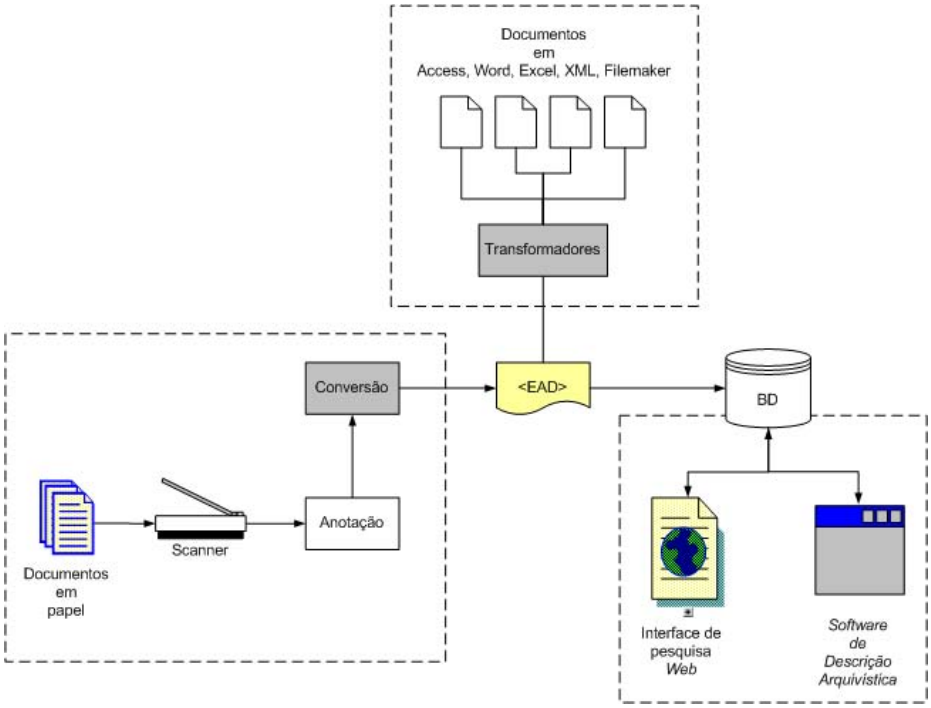


Figura 1. Diferentes fases do projecto.

bases de dados, assentava na necessidade dessa mesma informação ter de ser trocada com outros repositórios nacionais e internacionais. A norma Encoded Archival Description (EAD) foi escolhida para desempenhar essa função por se basear em XML, cujas vantagens neste contexto são amplamente conhecidas, e por se estar a tornar numa norma *de facto* no meio arquivístico para armazenamento e estruturação de informação. Além disso, assegura a criação de descrições consistentes, apropriadas e auto-explicativas e possibilita a partilha de dados de autoridade que tornam possível a integração de descrições de diferentes arquivos num sistema unificado de informação.

A informação arquivística é portadora de algumas características particulares. Para começar, encontra-se normalmente organizada hierarquicamente, efectuando uma descrição do mais geral para o mais particular. Assim, um documento alojado num Arquivo nunca se encontra descrito isoladamente (como acontece com um livro de uma biblioteca). Existe sempre uma relação entre o documento descrito e a entidade que o produziu, bem como uma descrição de todas as divisões e subdivisões dessa mesma entidade. Podemos, portanto, considerar uma descrição arquivística como uma árvore cujos nós descrevem diferentes partes de uma mesma organização. Ao nível da raiz possuímos a descrição do fundo (a organização que gerou o documento) e nos restantes níveis, a descrição das diferentes partes que o compõem. Ao nível das folhas são descritos os documen-

tos simples, ou seja, aqueles que não podem ser progressivamente subdivididos em mais níveis de descrição.

As vantagens do uso de XML neste contexto são óbvias, pois o armazenamento hierárquico da informação está automaticamente assegurado. No entanto, o XML possui um conjunto de características que o tornam difícil de manusear:

1. *O XML é baixo nível*
   Quer queiramos quer não, o XML é demasiado baixo nível para poder ser manipulado directamente por um utilizador. Os DTDs crescem de uma forma assustadora, de maneira que, é difícil encontrar uma norma internacional que não possua, pelo menos, várias centenas de elementos. Assim, é do interesse de todos, que o XML se mantenha, a toda a hora, escondido do utilizador comum, recorrendo a interfaces gráficas amigáveis que impossibilitem a ocorrência de erros na sua sintaxe.
2. *O XML é difícil de armazenar*
   Existe, hoje em dia, uma panóplia de opções no que diz respeito ao armazenamento de XML. O mercado das bases de dados com suporte para XML está em constante crescimento. Não obstante, podemos distinguir três estratégias fundamentais para o armazenamento de XML:
   (a) Sistemas XML nativos
   (b) Extensões XML (a sistemas RDBMS e OODBMS já existentes)
   (c) Sistemas XML virtuais (baseados em middleware entre as aplicações e o DBMS)

No entanto, não existe consenso no que diz respeito à melhor estratégia a seguir. Uma análise detalhada dos requisitos da aplicação terá que ser efectuada de forma a determinar qual das aproximações se adequa mais eficazmente ao cumprimento desses mesmos requisitos. Um estudo comparativo das diversas alternativas, realizado por R. Nunes e M. Silva [Nun03] conclui que, as bases de dados XML nativas, embora muito rápidas na recuperação da informação, consomem demasiado tempo durante a indexação de documentos de tamanho considerável e o uso de modelos relacionais, dotados de extensões, apenas são adequados quando a estrutura do XML é previamente conhecida e bem definida por um DTD.

3 Interface gráfica para descrição arquivística

A informação produzida por um Arquivo baseia-se, na sua essência, em meta-informação sobre os documentos albergados e as organizações que os produziram. Descrições intermédias entre ambos permitem catalogar os documentos no contexto da organização que os produziu, como por exemplo, identificar qual o serviço ou sucursal que gerou um documento específico.

Segundo as boas regras da arquivística, cada registo, ou nó, dessa árvore de descrição encontra-se identificado por um código de referência. Assim, um qualquer registo pode ser univocamente identificado pela concatenação das respectivas referências, desde o nível raiz até ao documento em causa. Por exemplo, a

referência completa EMP-BM/L/001/00001, poderá ser interpretada como pertencendo ao fundo EMP-BM (Empresa Banco do Minho), subfundo L (sucursal de Lisboa), série 001 (correspondência recebida) documento 00001 (a referência do documento propriamente dito).

Todos os sistemas de descrição arquivística que analisamos baseavam-se neste conceito de referências completas para identificar a posição da árvore onde o registo deveria estar pendurado. As interfaces gráficas de introdução de dados eram, assim, baseadas num único formulário onde todos os campos de meta-informação podiam ser introduzidos e onde um dos campos a preencher consistia na referência completa do registo. A referência do registo servia, assim, tanto para identificar o registo como para o situar na árvore de descrição.

Acontece, no entanto, que a ocorrência de erros aquando da introdução das respectivas referências (pelos operadores) é frequente, fazendo com que um registo salte para uma posição da árvore completamente distinta daquela que era pretendida, ou pior ainda, que o registo nem sequer possua significado na árvore em questão. Para além disso, o esforço mental que um operador necessita de fazer para visualizar a estrutura de uma organização é tremendamente desgastante, sendo este um dos principais causadores dos frequentes enganos.

De forma a minimizar a ocorrência de erros que deterioram francamente a qualidade das descrições, propusemos a realização de uma interface gráfica que representasse visualmente a árvore de descrição eliminando a necessidade de introdução de longas referências e impedindo o operador de cometer erros aquando da sua introdução. Assim, a interface gráfica está dividida em duas áreas distintas, uma constituída por uma árvore representativa do fundo em que se está a trabalhar e uma outra onde são apresentados os campos que podemos preencher para descrever o registo seleccionado (figura 2).

Em teoria arquivística, cada registo encontra-se posicionado num nível de descrição, uma espécie de catalogação que caracteriza o tipo de registo. No contexto do Arquivo Distrital do Porto foram identificados dez níveis de descrição distintos: Fundo, Subfundo, Secção, Subsecção, Subsubsecção, Série, Subsérie, Unidade de instalação, Documento composto e Documento simples. A interface gráfica seria, também, capaz de garantir a coerência da descrição, impedindo o utilizador de desrespeitar a lógica hierárquica inerente, e.g., a interface não deveria permitir a criação de uma Secção debaixo de uma Subsecção, pois isso violaria a lógica hierárquica subjacente. Além disso, o software deveria ser capaz de detectar incoerências e omissões na descrição elaborada. Existem campos que são considerados obrigatórios, como a Referência, o Título ou as Datas extremas. O programa deveria alertar o utilizador quando algum destes campos não fosse preenchido, ou a informação nele contida fosse incoerente, e.g., uma data final superior a uma data inicial.

Para além dos cuidados a ter com a qualidade da descrição, o software deviria ser possuidor de algumas características adicionais, como por exemplo, permitir a inferência de informação de níveis inferiores para níveis de topo. As datas extremas associadas a um registo de nível fundo deverão ser, respectivamente, a data inicial do primeiro documento produzido pela organização e a data final

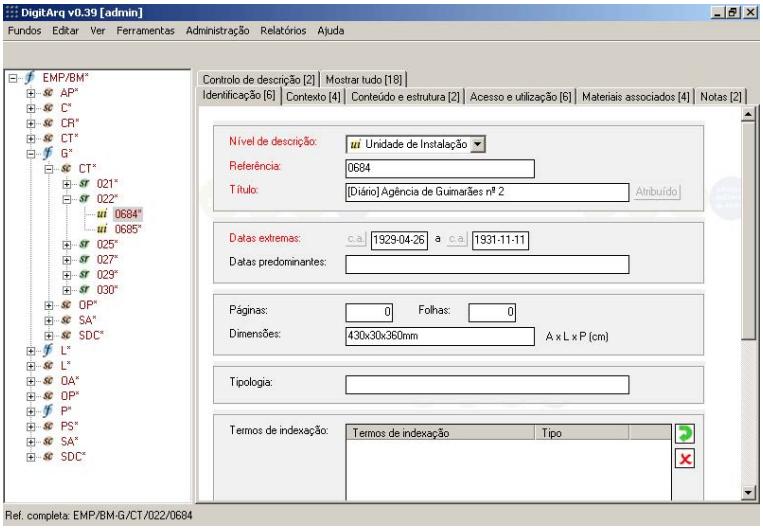


Figura 2. Interface do software de descrição arquivística.

o documento mais antigo. Assim, através de métodos de inferência deveria ser possível transportar para níveis superiores informação relativa aos documentos folha da árvore.

O software de descrição deveria, também, ser capaz de permitir que vários operadores pudessem trabalhar simultaneamente sobre um mesmo fundo, podendo observar em tempo real as alterações efectuadas pelos outros operadores, obtendo assim uma visão geral do todo o trabalho concretizado pela equipa de descrição.

4 Modelo de dados

No projecto DigitArq foi adoptada uma abordagem *top-down*, tomando como ponto de partida a interface gráfica que desejávamos desenvolver e analisando todos os requisitos que a mesma deveria suportar. A partir daí, construímos um modelo de dados capaz de suportar todas exigências previamente definidas. Após uma análise superficial da norma EAD e da constatação da sua complexidade passamos à fase que normalmente antecede a utilização de uma qualquer norma baseada em XML - a *adaptação da norma*. Esta, consiste na análise de requisitos do sistema a desenvolver e no teste de adequabilidade da norma a utilizar, recorrendo, por vezes, à adaptação de alguns elementos para suportar a informação pretendida e à rejeição de elementos desnecessários. A fase de adaptação da norma é fundamental ao desenvolvimento de qualquer projecto baseado em XML, pois transforma uma pesada norma internacional numa estrutura de tratamento simples e de tamanho manejável adequada às necessidades específicas dum determinado projecto.

4.1 Adaptação do EAD

Esta secção pretende servir de guia para a codificação de auxiliares de pesquisa em EAD/XML. Foi nossa intenção excluir deste documento uma introdução à norma EAD. Existe, na Internet, bastante documentação sobre este assunto [oAA02]. Assume-se, portanto, que o leitor conhece a norma EAD e que está a par das suas ambiguidades. A norma é bastante liberal na maior parte das suas componentes. Assim, algumas decisões tiveram que ser tomadas de forma a conseguir codificar a informação extraída, das diversas bases de dados.

De seguida são descritas algumas das decisões tomadas no âmbito do projecto DigitArq.

Datas extremas Um dos primeiros problemas detectados na norma EAD consistia na inexistência de elementos capazes de acolher as datas extremas dos registos retro-convertidos. Por norma, um documento é portador de duas datas extremas, a data de criação inicial e final. Em alguns casos essas datas poderão coincidir. Um caso típico de utilização de datas extremas ocorre geralmente quando descrevemos livros. Um livro de baptismos, por exemplo, possui, ao longo das suas páginas, centenas de registos, fazendo com que as datas extremas do livro sejam, respectivamente, as datas do primeiro, e do último baptismo registados.

A norma EAD apenas contempla a existência de um elemento para representar datas extremas, sendo da responsabilidade do utilizador assegurar a coerência do seu formato. Existem diversas soluções para este problema, no entanto, todas estas recaem sobre dois modelos fundamentais: utilizar uma sintaxe bem definida para as datas que permita distinguir a data inicial da data final, ou utilizar um atributo para indicar a qual das datas o elemento descritivo se refere.

No primeiro caso, se quiséssemos representar as datas extremas da crise cerealífera que assolou o nosso país no século XV, poderíamos fazê-lo recorrendo à notação `<unitdate>1436/1441<unitdate>`.

No segundo caso, a representação seria assegurada pelos elementos `<unidate datechar='inicial'>` 1436 `</unitdate>` e `<unidate datechar='final'>` 1441 `</unitdate>`, onde o atributo `datechar` indica a que data o elemento `unitdate` pretende descrever.

Em ambos os casos, a sintaxe escolhida para representar as datas deverá ser bem definida, pois é fundamental que se possam efectuar pesquisas sobre as datas dos registos, tendo obviamente em consideração os intervalos definidos.

Outro problema, que acompanha a codificação de datas, tem que ver com a representação de informação incompleta, ou seja, como representar as datas para as quais não conhecemos, por exemplo, o dia ou o mês. O formato escolhido deve permitir determinar se uma qualquer data pertence a um intervalo definido por duas datas incompletas.

No que diz respeito às datas, as decisões tomadas podem ser resumidas da seguinte forma:

– *Utilização de atributos para identificar a extremidade da data*
Evita a necessidade de processamento adicional para separar as datas na

eventualidade de ser necessário efectuar qualquer tipo de operação com as mesmas.
- *As datas são sempre representadas no formato YYYY-MM-DD*
  Garante a uniformidade das datas e simplifica todo o tipo de cálculos a efectuar com as mesmas.
- *Informação incompleta*
  A informação incompleta é representada por uma cadeia de zeros com o mesmo comprimento do elemento em causa, e.g., na data 1436-04-00 depreende-se que o dia é desconhecido.

Numbered vs Unnumbered Components Em EAD cada nível de descrição pode ser representado de duas formas: utilizando os elementos `<c1>`, `<c2>`, ...,`<c12>`, cujo valor numérico associado representa a profundidade do elemento na árvore, ou utilizando um elemento não numerado simplesmente representado por `<c>`.

No âmbito do nosso projecto não sentimos necessidade de utilizar os elementos numerados, pois a profundidade da hierarquia é automaticamente descrita pela disposição dos diversos elementos no documento XML. Optamos assim, pela utilização do elemento `<c>` uma vez que o tratamento numérico da profundidade, para além de supérfluo, acarreta processamento adicional.

Nos exemplos que se seguem, ambas as representações são perfeitamente equivalentes:

```
<c01>
    <c02>
        <c03>
            ...
        </c03>
    </c02>
    <c02>
        ...
    </c02>
</c01>
```

Exemplo 1: Anotação usando *Numbered Components*.

```
<c>
    <c>
        <c>
            ...
        </c>
    </c>
    <c>
```

```
      ...
    </c>
</c>
```

Exemplo 2: Anotação usando *Unnumbered Components*.

É de notar que o elemento `<c>` (*component*) representa um nó da árvore de descrição documental. Pendurados neste elemento, encontram-se todos os elementos que carregam informação útil produzida pelo arquivista, e.g., o código de referência e o título do elemento descritivo.

Um registo cujo código de referência é EMP-BM e cujo título é Banco do Minho podem ser codificados em EAD da seguinte forma:

```
...
<c>
    ...
    <unitid>EMP-BM</unitid>
    <unittitle>Banco do Minho<unittile>
    ...
</c>
...
```

Atributo otherlevel Associado a cada elemento `<c>` (*component*) existe um atributo `level` que indica qual o nível de descrição arquivística que o elemento representa. No seio do nosso projecto foram definidos os níveis de descrição: Fundo, Subfundo, Secção, Subsecção, Subsubsecção, Série, Subsérie, Unidade de instalação, Documento composto e Documento simples. O atributo `level` do elemento *component* descrito pela norma EAD/XML define os níveis de descrição: class, collection, fonds, subfonds, series, subseries, file, item, recordgrp , subgrp e otherlevel.

Como podemos constatar, os níveis de descrição definidos na norma não satisfazem as necessidades do Arquivo Distrital do Porto. Assim, foi utilizado o atributo opcional `otherlevel` onde pudemos descrever o nosso nível de descrição, bastando para isso identificar o nível de descrição como sendo `otherlevel`.

Exemplo: `<c level='otherlevel' otherlevel='Fundo'> ...</c>`

Elementos mistos O EAD contempla a utilização de alguns elementos especiais para anotar secções de texto no interior de outros elementos que se designam

por elementos de conteúdo misto: os chamados elementos flutuantes. Assim, é possível, por exemplo, na descrição do Âmbito e Conteúdo, de um qualquer registo, assinalar que um determinado conjunto de palavras diz respeito ao nome de uma pessoa.

```
<scopecontent>
  O fundador, <person>Manuel Ferreira</person>, constituiu ...
</scopecontent>
```

A utilidade dos elementos *flutuantes* neste contexto é amplamente reconhecida, no entanto, a sua implementação do ponto de vista da interface gráfica acarreta alguns problemas. Como poderemos permitir a anotação de texto, sem que as etiquetas sejam visualizadas? Usando um sistema de cores para diferenciar o texto anotado? Seria uma hipótese, no entanto, se encararmos o problema do ponto de vista relacional veremos que é demasiado complexo de tratar. Assim, todos os elementos *flutuantes* foram substituídos por termos de indexação. Por outras palavras, acabaram-se com os conteúdos mistos (estes são normalmente o problema quando se pretende guardar documentos num modelo relacional).

Termos de indexação A linguagem natural é utilizada correntemente em todas as actividades da vida quotidiana. De acordo com o contexto, o nível da língua será familiar, erudito, técnico, poético, literário, diplomático, administrativo, etc. Vários termos são sinónimos, antónimos; são específicos ou genéricos. Esta diversidade ilustra a riqueza e a subtileza da língua. Geralmente, os títulos das obras são redigidos em linguagem natural, o que facilita a sua compreensão pelos leitores.

Na linguagem natural, pode-se escolher um termo, ou um conjunto de termos para se fazer uma descrição, uma representação, uma ilustração ou uma síntese de um assunto, de uma ideia, de um texto, de uma situação.... etc. O termo assim escolhido torna-se uma palavra-chave. Esta palavra-chave permite-nos identificar o que é importante ou o que é preciso reter para compreender o essencial de um assunto. Consoante o caso, a palavra-chave tem um sentido geral ou específico.

A utilização da palavra-chave ou de um conjunto de palavras-chave, é uma estratégia eficaz para delimitar um assunto de pesquisa. Também nesta perspectiva, quanto mais a palavra-chave for precisa e pertinente mais fácil será o início da pesquisa da informação. Além disso, a utilização da palavra-chave torna-se essencial para fazer a ligação com as linguagens especializadas dos diversos domínios do conhecimento humano. A linguagem controlada utiliza termos escolhidos, precisos e unívocos com a finalidade de evitar as interpretações de sentido, permitindo assim a indexação precisa dos registos [dRdBE03].

Assim, associado a cada registo, existe uma lista de palavras-chave assentes em linguagem controlada, que permitem identificar os principais tópicos abordados. Como a linguagem é bem definida, é possível efectuar pesquisas eficazes sobre a meta-informação existente.

No projecto DigitArq os termos de indexação foram hierarquizados a dois níveis. O primeiro identifica uma série de categorias às quais podemos adicionar

termos concretos. Esta opção reduz a ambiguidade inerente a algumas palavras ou expressões. Por exemplo, será que "Elias Garcia"diz respeito ao nome de uma rua ou ao nome de uma escola secundária de Almada? Ao descrevermos o termo como "Toponímia/Elias Garcia"concluímos imediatamente que se trata do nome de uma rua. Em EAD o exemplo representar-se-ia da seguinte forma:

```
...
<controlaccess>
    <list>
        <defitem>
            <label>Toponímia</label>
            <item>Elias Garcia</item>
        </defitem>
    </list>
</controlaccess>
...
```

A definição dos termos a utilizar fica a cargo do administrador do sistema, ou seja, alguém no interior do Arquivo que possui autoridade para gerir a lista de termos de indexação.

4.2 XML vs SQL

As diversas fases de conversão resultaram em milhares de registos migrados de diversas bases de dados. O novo sistema de informação deveria garantir que toda a informação convertida fosse incorporada e tornada acessível ao público geral. Tornou-se, assim, fundamental a criação de um repositório central para que fosse possível gerir toda essa informação, bem como, permitir a realização de pesquisas através de uma interface Web. Para a realização dessa tarefa foi utilizada uma base de dados relacional.

A escolha de uma base de dados relacional recaiu, em parte, nas condicionantes orçamentais para a aquisição de uma base de dados XML nativa e nas restrições temporais vigentes, que impediam que demasiado tempo fosse dispendido em instalação e configuração.

Foi necessário desenvolver um modelo de dados capaz de funcionar sobre uma base de dados relacional que conseguisse reflectir, dentro do possível, a informação contida nos ficheiros EAD/XML resultantes das diversas conversões. A transição de um dos modelos para o outro (XML/Relacional) deveria ser simples e transparente. Foi então desenvolvida, uma camada intermédia de software para funcionar entre a aplicação de gestão, e a base de dados.

A camada intermédia (ou *middleware*) é descrita por uma classe abstracta onde são definidos nove métodos fundamentais:

- `Sub Upload()`
  Actualiza a informação do nível de armazenamento com a informação residente em memória.

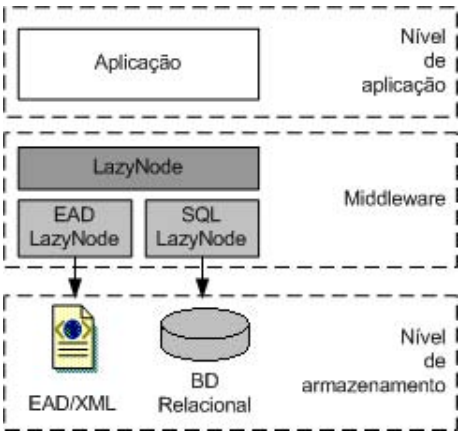


Figura 3. Diagrama de blocos.

- `Sub Download()`
  Actualiza a memória com a informação do nível de armazenamento. Este método permite refrescar os dados em memória de modo a garantir que a interface apresenta a versão mais recente do registo.
- `Function Children() As LazyNodeCollection`
  Retorna uma colecção com os nós filhos. Se não existirem filhos retorna uma colecção vazia.
- `Sub RemoveChild(ByVal child As LazyNode)`
  Remove um filho do nó actual.
- `Sub AppendChild(ByVal child As LazyNode)`
  Adiciona um novo filho ao nó actual.
- `Function HasChildren() As Boolean`
  Retorna verdadeiro se o nó actual tiver filhos, e falso em caso contrário.
- `Function CreateNode() As LazyNode`
  Cria um novo nó. O novo nó terá que ser adicionado como filho ao nó pretendido.
- `Function Clone() As LazyNode`
  Cria uma cópia do nó actual.
- `Function Parent() As LazyNode`
  Retorna o pai do nó actual. Caso não exista (o nó actual é a raiz) retorna o valor nulo.

Para além destes métodos, que cada implementação concreta da classe abstracta é obrigada implementar, cada nó carrega consigo um conjunto de propriedades que permitem conservar em memória os valores de cada campo de informação que o nosso sistema é capaz de manipular.

A classe abstracta foi baptizada de LazyNode (nó preguiçoso), uma vez que, a informação apenas é transportada do nível físico para a memória, a pedido. Desta forma não sobrecarregamos a memória do sistema com informação indesejada.

Foram desenvolvidas duas implementações da classe LazyNode: SQLLazyNode e EADLazyNode. Cada uma das implementações limita-se a implementar os nove métodos herdados.

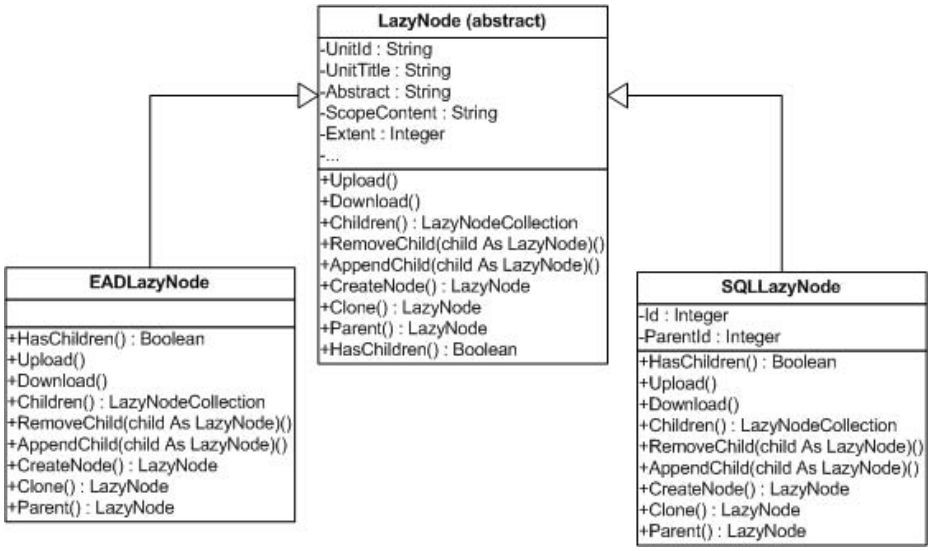


Figura 4. Diagrama de classes do LazyNode.

4.3 EADLazyNode

A classe EADLazyNode opera sobre documentos EAD/XML fazendo uso do DOM (W3C Document Object Model) como base de suporte para a manipulação dos ficheiros XML. Assim, todas as operações descritas anteriormente são implementadas com métodos disponibilizados pelo DOM.

Existe uma relação directa entre cada propriedade e a sua representação em XML (tabela 1).

Propriedade	**Elemento EAD/XML**
Otherlevel	@otherlevel
Unitid	did/unitid
CountryCode	did/unitid/@countrycode
RepositoryCode	did/unitid/@repositorycode
...	...

Tabela 1. Relação entre propriedades do LazyNode e elementos do EAD/XML

As operações de leitura sobre um documento XML limitam-se a retornar o valor do elemento indicado pelo XPath. Caso o elemento não exista, é retornado o valor nulo.

Antes de uma operação de escrita é verificada a existência do XPath correspondente à propriedade que se pretende escrever. Caso não exista, o caminho é criado e o valor do respectivo elemento é actualizado.

5 SQLLazyNode

A implementação do SQLLazyNode permite-nos manipular a informação quando esta se encontra armazenada numa base de dados relacional. Cada nó da árvore de descrição documental (*component* `<c>`) é descrito, no contexto relacional, por um registo que contém uma coluna por cada propriedade definida. Para além das propriedades, foi necessário adicionar alguns campos de controlo: Id, ParentId e HasChildren.

O modelo hierárquico inerente à árvore de descrição documental é assegurado por uma relação circular, onde o campo ParentId de um registo aponta para o Id do registo hierarquicamente superior (figura 5). Um registo raiz possui um apontador nulo (ParentId = NULL), ou seja, não possui pai.

Foi utilizado um campo adicional de controlo designado *HasChildren* (boleano) para indicar se um determinado nó da árvore possui, ou não, filhos. Assim, é possível obter esta informação sem sobrecarregar a base de dados com uma consulta demasiado pesada.

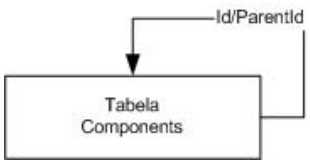


Figura 5. Diagrama entidade-relação.

As operações obrigatórias são implementadas de uma forma simples sobre o modelo relacional:

- `Sub Upload()`

```
UPDATE Components
SET {campos a actualizar}={valor}
WHERE ID={Id do nó actual}
```

- `Sub Download()`

```
SELECT *
FROM Components
```

– `Function Children() As LazyNodeCollection`

```
SELECT Id
FROM Components
WHERE ParentId={Id do nó actual}
```

– `Sub RemoveChild(ByVal child As LazyNode)`

```
For Each Node In Child.Children
    RemoveChild(Node)
Next
Dim SQLChildNode As SQLLazyNode = CType(Child, SQLLazyNode)

DELETE FROM Components
WHERE ID = {Child.Id}
If Children.Count = 0 Then
    UpdateHasChildren(False) 'updates the HasChildren Flag
End If
```

– `Sub AppendChild(ByVal child As LazyNode)`

Assumindo que foi executado previamente um CreateNode() ou um Clone() de modo a obtermos uma referência para um novo nó:

```
UPDATE Components SET ParentID={Id do nó actual}
WHERE ID={Id do nó a adicionar}
```

– `Function HasChildren() As Boolean`

```
SELECT HasChildren
FROM Components
WHERE ID = {Id do nó actual}
```

– `Function CreateNode() As LazyNode`

```
INSERT INTO Components ({campos não nulos})
VALUES ({valores por omissão})
```

– `Function Clone() As LazyNode CreateNode()`

```
NewNode.ImportFields(Me)
NewNode.Upload()

Dim Child As LazyNode For Each Child In Children()
    Dim NewChild As LazyNode = Child.Clone
    NewNode.AppendChild(NewChild)
Next
```

– `Function Parent() As LazyNode`

```
SELECT ParentID
FROM Components
WHERE Id={Id do nó actual}
```

5.1 Vantagens e desvantagens do modelo adoptado

O modelo apresentado é simples e apresenta algumas características interessantes. A primeira de todas é a capacidade podermos realizar catamorfismos sobre o modelo descrito sem termos que nos preocupar com as particularidades do nível de armazenamento. Esta característica é realmente importante se considerarmos que a maior parte das operações desempenhadas pelo software de gestão documental são realizadas por fundo (árvore) e que normalmente exigem travessias completas sobre o mesmo. Alguns exemplos deste tipo de operações são:

- Revisão automática de fundos
- Inferência de valores (de níveis inferiores para níveis de topo)
- Contabilização de registos por nível de descrição
- Normalização da codificação (para resolver incongruências herdadas da retroconversão)

A definição do catamorfismo fica a cargo da seguinte função:

```
Public Function Catamorphism(ByVal Gene As Gene) As Object
    Dim Child As LazyNode
    Dim ChildrenResults As New Collection
    For Each Child In Children()
        ChildrenResults.Add(Child.Catamorphism(Gene))
    Next
    Return Gene.Apply(Me, ChildrenResults)
End Function

Public Interface Gene
    Function Apply(ByVal obj As LazyNode, _
        ByVal ChildrenResults As Collection) As Object
End Interface
```

Exemplo de um Gene.

```
Public Class GeneValidator
    Implements Gene

    Public Function Apply(ByVal obj As LazyNode, _
            ByVal ChildrenResults As Collection) As Object

        Dim Result As ValidationErrorCollection
        Result = Validator.GetLazyNodeErrors(obj)
        Dim Child As ValidationErrorCollection
        For Each Child In ChildrenResults
            Result.Add(Child)
        Next
```

```
        Return Result
    End Function
End Class
```

O modelo adoptado é totalmente transparente no diz respeito à localização física da informação. Desde que os nove métodos descritos sejam correctamente implementados sobre a respectiva camada de dados, o mesmo conjunto de operações podem ser aplicadas sem mais alterações. A conversão entre modelos de dados é, também, simples e transparente.

Se pretendermos exportar uma árvore (fundo documental) armazenada numa base de dados relacional para um ficheiro EAD/XML basta efectuar uma travessia na árvore original e ir criando, progressivamente, os nós correspondentes na implementação XML do LazyNode.

Exemplo:

```
Private Sub MergeTrees(ByVal DstNode As LazyNode,
                       ByVal SrcNode As LazyNode)
    Dim Index As Integer
    Dim ChildSrcNode As LazyNode
    Dim Children As LazyNodeCollection

    Children = DstNode.Children
    For Each ChildSrcNode In SrcNode.Children
        Index = Children.IndexOf(ChildSrcNode)
        If Index < 0 OrElse ChildSrcNode.isLeaf Then
            ' node doesn't exist. Create a new one
            Dim NewDstNode As LazyNode = DstNode.CreateNode()
           ' Copy all the stuff into the node
            NewDstNode.ImportFields(ChildSrcNode)
            NewDstNode.Upload()
            DstNode.AppendChild(NewDstNode)
            MergeTrees(NewDstNode, ChildSrcNode)
        Else ' Node exists... use that one
            Dim NewDstNode As LazyNode = Children.Item(Index)
            MergeTrees(NewDstNode, ChildSrcNode)
        End If
    Next
End Sub
```

Uma particularidade interessante deste modelo tem que ver com a sua independência ou portabilidade. Todas as operações descritas utilizam ferramentas *standard*, como o DOM ou o SQL, permitindo, assim, que o modelo seja facilmente transportado para outras plataformas.

O modelo de dados apresentado, embora simples de implementar e manipular, apresenta algumas desvantagens face a outras soluções, sendo a principal

destas o facto de não tirar o máximo partido das capacidades oferecidas pelas bases de dados relacionais. A granularidade do modelo obriga a que toda a informação seja carregada registo a registo, ou seja, primeiro é carregada a raiz, e progressivamente, cada um dos nós filho (quando estes são necessários). A simples procura de um nó na árvore implica realizar uma travessia completa até o nó ser encontrado. Num modelo relacional puro, o registo seria encontrado pelo motor da base de dados, bastando-nos apenas executar uma query SQL. Este é o custo a pagar pela simplicidade e transparência do modelo, no entanto, os testes de utilizador efectuados ao sistema revelam que a informação é encontrada em tempo útil, ou seja, o utilizador não considera excessivo o de espera pela resposta do sistema, encarando o facto como normal.

6 Conclusões e trabalho futuro

Neste artigo é proposto um modelo de dados, baseado em *middleware*, que oferece simplicidade e transparência na manipulação de informação hierárquica residente em diferentes níveis lógicos de armazenamento. Foram criadas duas implementações concretas, uma sobre ficheiros XML baseados na norma EAD e outra sobre uma base de dados relacional. Em ambos os casos, a informação manipulada baseia-se num modelo hierárquico - uma árvore.

A passagem de informação de um modelo para o outro é assegurada por uma simples travessia. A aplicação de catamorfismos é conseguida indiferentemente do modelo utilizado.

O modelo apresentado, por ser abstracto e generalista, não tira partido das funcionalidades oferecidas pelas bases de dados relacionais. Assim, algum trabalho futuro poderia ser desenvolvido na optimização do modelo sobre bases de dados relacionais. A utilização de stored procedures seria um bom ponto de partida para concretização deste objectivo, tendo como principal desvantagem, a perda de independência e portabilidade, características do modelo proposto.

Outra possibilidade de optimização seria utilizar mecanismos de cache e/ou *pre-loading* de forma a minimizar o tráfego entre a aplicação e a base de dados. A utilização destes mecanismos acarreta informação de controlo adicional para garantir que a informação em cache se encontra coerente/actualizada.

Referências

[Bou] Ronald Bourret. Mapping dtds to databases. Internet.

[Day] Igor Dayen. Storing xml in relational databases. Internet.

[dNdD99] Comité de Normas de Descrição. *ISAD(G): Norma geral internacional de descrição arquivística*, second edition, Setembro 1999. Setembro.

[dRdBE03] Gabinete da Rede de Bibliotecas Escolares. A pesquisa de informação - o professor e a biblioteca, parceiros do aluno. Internet, Junho 2003.

[Fox00] Michael J. Fox. Ead cookbook, Julho 2000.

[Nun03] Rui Alexandre Nunes. Armazenamento de dados complexos utilizando xml. Master's thesis, Instituto Superior Técnico - Portugal, 2003.

[oA96] International Council on Archives. *International Archival Authority Record for Corporate Bodies, Persons and Families*. 1996.

[oAA02] Society of American Archivists. Ead standards home page. Internet, 2002.

[SL] Mustafa Atay Farshad Fotouhi Shiyong Lu, Yezhou Sun. A new inlining algorithm for mapping xml dtds to relational schemas. Technical report, Wayne State University - Department of Computer Science, Detroit, MI 48202.

Utilizacion de RDF y bases de datos de metadatos nativas dentro del proyecto Omnipaper

Cesar Ariza and Ana Alice Baptista

Universidade do Minho
\{cariza,analice\}@dsi.uminho.pt

Resumo Este articulo describe el trabajo realizado para la creación de un prototipo para la búsqueda de información en archivos digitales distribuidos utilizando la tecnología RDF y una base de datos nativa. El articulo reseña los prerrequisitos para la descripción y normalización de la información de los archivos distribuidos, luego los criterios para la selección de la base de datos nativa, muestra las funcionalidades del prototipo creado y al final tiene una síntesis de las lecciones aprendidas y el trabajo futuro.
Palavras Chave: Digital Libraries, Distributed Information Retrieval, Wordnet, Metadata

1 Intrduccion

En las últimas décadas, la cantidad de información digital, así como el número de ordenadores y conexiones a Internet, ha crecido a un ritmo exponencial. Cada vez es más y más frecuente acceder a la información en formato electrónico, posibilidad que se ve altamente potenciada gracias a la red. Asimismo, cada vez resulta más fácil comparar información procedente de lugares geográficamente distintos, de ahí la creciente importancia de que esta información se encuentre relacionada entre sí a nivel semántico.

El proyecto OmniPaper (Smart Access to European Newspapers) patrocinado por la Comisión Europea IST investiga técnicas para acceder fuentes de información distribuidas con base en tecnologías de IA y XML (SOAP, RDF, XTM). La arquitectura de la solución propuesta por OmniPaper tiene en su base un conjunto de archivos digitales distribuidos, todos dentro de ambientes, bases de datos y mecanismos de indexación diferentes, los cuales pueden ser accedidos de una manera uniforme mediante una interfaz SOAP. Los archivos contienen recursos (artículos de periódicos) catalogados mediante RDF y XTM ("local layer") lo que permite una búsqueda inteligente gracias a una capa superior ("knowledge layer") que contiene conceptos relacionados entre si con ocurrencias en diferentes idiomas.

Fueron creados prototipos en RDF y XTM para comparar las dos tecnologías. Este artículo describe la fase inicial del prototipo RDF y la incorporación de WordNet para facilitar la navegación dentro del mismo.

2 RDF

RDF (del ingles "Resource Description Framework"), como su nombre lo indica es un marco de trabajo para describir e intercambiar metadatos. RDF se divide en dos partes, una de ellas el "Modelo RDF y especificación de sintaxis" que tiene un modelo para representar metadatos y la sintaxis para codificarlos y la segunda parte Especificación de Esquemas RDF para la ceración de vocabularios de metadatos, los vocabularios permiten describir objetos de un negocio o área de conocimiento. RDF esta construido en base a las siguientes reglas:

- Un recurso es cualquier cosa que puede tener un URI, esto incluye todas las paginas web, todos los elementos individuales de cada documento XML y mucho mas;
- Una propiedad es un recurso que tienen un nombre y que puede usarse como una propiedad, por ejemplo autor o titulo. En muchos casos todo lo que nos importa en realidad es el nombre, pero una propiedad necesita ser un recurso de forma tal que pueda tener sus propias propiedades;
- Una sentencia consiste en la combinación de un recurso, una propiedad y un valor.

Estas tres partes son conocidas como el sujeto, predicado y el objeto de la sentencia, un conjunto de las tres partes es llamado un triple. RDF se representa en XML, de otra manera RDF es una aplicación XML, los triplos RDF pueden ser almacenados en bases de datos normales o en bases de datos especializadas.

3 Prototipo RDF

Los objetivos del trabajo con RDF y el desarrollo del prototipo se derivan de los objetivos del proyecto Omnipaper. El principal objetivo es explorar la tecnología RDF para la descripción, búsqueda y recuperación de información en el contexto de un archivo digital de noticias distribuido.

3.1 Trabajo previo

El trabajo comienza con formalización de la descripción de los articulos de noticias; para describir los artículos se definió un vocabulario de metadatos común para tres archivos de noticias (los archivos pertenecen a tres empresas proveedoras de noticias en línea); se estudiaron varios vocabularios estándar principalmente dentro del sector de noticias. Finalmente fueron seleccionados los elementos que mejor se ajustaron a las necesidades y como resultado el vocabulario de metadatos de OmniPaper contiene elementos Dublin Core, NIFT y algunos elementos propios de Omnipaper, además, fue creado un perfil de aplicación ("Application profile") para el vocabulario. El vocabulario cuenta con 25 elementos pertenecientes a 6 categorías; la categoría "Clasificación del Artículo" tiene el elemento "keylist", que es una lista pesada de palabras llave contenidas en cada artículo, las cuales son extraídas mediante métodos de inteligencia artificial.

Con base en el vocabulario creado se describió un conjunto de noticias; dichas descripciones son documentos RDF/XML, los documentos RDF/XML fueron creados en dos pasos, el primero fue la transformación de los artículos, que originalmente son ficheros XML, por medio de plantillas de transformación XSL, se convirtieron en documentos RDF/XML, el proceso de transformación, mas que transformar mapeo los metadatos existentes dentro de los ficheros XML con los del vocabulario creado, se corrigieron algunos formatos de datos como el de la fecha; el contenido de los articulos no fue mapeado; el segundo paso es la inclusión de la lista pesada de palabras llave del artículo que previamente han sido extraídas por un proceso desarrollado para tal fin.

3.2 Base de datos nativa: RDF Gateway

La seleccion de la base de datos se hizo bajo la premisa de ser orientada a XML (nativa); inicialmente se exploro el producto TAMINO de SoftwareAG, después fue explorado RDF Gateway de Intellidimension; debido a que RDF-Gateway fue creado para manipular informacion en RDF/XML y ofrecía funcionalidades de manejo fue seleccionado, a pesar de ser un producto nuevo.

RDF Gateway además de manipular documentos RFD/XML, tiene un servidor web con un ambiente de programación que soporta paginas activas (RSP), facilidad de conexión con otros programas, y características que lo diferencian de las demás bases de datos normales, como reglas de inferencia, soporte para esquemas y la creación automática de índices con base en las raíces de las palabras ("steeming") contenidas en los campos indexados.

3.3 Descripción y funcionamiento del prototipo

El primer prototipo fue desarrollado en el ambiente de páginas activas (RSP) que soporta RDF Gateway; debido a que el conjunto de sentencias que ofrece RDF Gateway es limitado, el segundo prototipo (que es el descrito en este articulo) fue desarrollado en un ambiente de desarrollo VisualStudio de Microsoft y esta compuesto por varias paginas ASP. El acceso al motor de base de datos de RDF Gateway se hace vía ODBC.

El prototipo RDF es una aplicación que envía consultas inteligentes a la metabase. Una consulta inteligente tiene como base un consulta normal con dos diferencias, la primera, el enriquecimiento de la consulta con sinónimos, la segunda, gracias al motor de base de datos se realiza la búsqueda con la raíces de las palabras. La consulta inteligente tiene ventajas sobre una consulta normal de texto completo ("full text search"), principalmente en los tiempos de respuesta y en la calidad de la información encontrada. La consulta enviada al motor de base de datos se hace en RDFQL, que es un el leguaje de consultas parecido a SQL.

La interfaz con el usuario del prototipo RDF tiene cuatro componentes, la primera es una búsqueda inteligente, que permite la entrada de varias palabras llave para realizar una consulta, la segunda, permite la búsqueda en cada uno de los elementos de vocabulario de metadatos, la tercera es una búsqueda orientada

a la navegación, desde una palabra llave inicial, permite recorrer las palabras relacionadas y a su vez muestra los artículos relacionados a esas palabras, los resultados aparecen en el cuarto componente de la interfaz. El componente del interfaz orientado a navegación utiliza WordNet para ayudar al usuario a encontrar mejores resultados.

4 Lecciones aprendidas

Se comprobaron algunas premisas planteadas en otras áreas, las cuales se pueden resumir en:

- Los productos, como RDF Gateway, que ofrecen muchas funcionalidades, no tienen la suficiente calidad en todas las funcionalidades, de ahí que se cambio el ambiente de desarrollo a VisualStudio y a la tecnología ASP.
- La indexación que tienen los datos almacenados en RDF Gateway, produce buenos tiempos de repuesta, pero ralentiza la actualización e inserción de información.
- El diseño de las consultas y la base de datos es difícil debido a los conocimientos previos de SQL, normalmente se intenta hacer lo mismo que se realiza en una base de datos típica, olvidando que es una filosofia totalmente diferente.

5 Conclusiones y trabajo futuro

El principal beneficio de utilizar RDF y RDF Gateway es la y facilidad uso debido a que los conceptos que se necesitarían para realizar el mismo trabajo en otras tecnologías, ya están encapsulados en RDF. Esta facilidad se ve disminuida porque las tecnologías envueltas son muy recientes, lo cual no permite alcanzar resultados esperados.

Como trabajo futro se pretende la integración del prototipo RDF con el prototipo XTM en una sola aplicación, lo cual permitirá potenciar las dos tecnologías.

Um Extrator de Topic Maps a partir de Recursos Heterogêneos de Informação

Giovani Rubert Librelotto[1]*,
José Carlos Ramalho[1], and Pedro Rangel Henriques[1]

University of Minho, Computer Science Department
4710-057, Braga, Portugal
{grl, jcr, prh}@di.uminho.pt

Abstract. O processo de desenvolvimento de ontologias baseadas em Topic Maps é complexo, consumidor de tempo e requer grande quantidade de recursos humanos e financeiros, devido ao fato de qualquer topic map (por mais simples que seja) possuir um conjunto significativo de tópicos e associações; além disso, para que a ontologia extraída seja realmente significativa, pode envolver um grande número de recursos de informação que poderão ser de tipos diferentes. Para resolver este problema, este artigo propõe um extrator de ontologias, chamado *Oveia*, que constrói topic maps a partir de recursos heterogêneos de informação, onde a ontologia a ser extraída é definida em uma linguagem de especificação denominada XS4TM (XML Specification for Topic Maps). O topic map extraído pode ser armazenado em uma base de dados relacional (permitindo que as ontologias possam crescer, sem restrições), ou em um documento no formato XML Topic Maps (XTM). Essa dupla capacidade de manipular várias fontes de informação e de poder armazenar o resultado em um suporte diferente é vantagem na comparação com a primeira versão do extrator, chamado *TM-Builder*.

1 Introdução

No funcionamento normal de uma organização, tipicamente são produzidos grandes volumes de dados. Normalmente, para satisfazer os seus requisitos de armazenamento, estas organizações utilizam bases de dados relacionais que são bastante eficientes para lidar com esta situação. Entre outras razões, os seus sistemas de indexação estão otimizados para suportar grandes volumes de dados.

Quando há necessidade de uma estruturação de mais alto nível dessa informação, o paradigma Topic Maps – ISO/IEC 13250 – mostra ser uma excelente opção, por ser suportado em um número reduzido de elementos simples (tópicos e associações). Com um mapa conceitual da informação encontrada nas bases de dados, habilita-se uma navegação através dos conceitos e das relações. Para isso, é necessária a definição de uma ontologia adequada ao universo em que esse sistema se insere, na qual estejam especificados os conceitos e as relações entre as entidades do próprio sistema. Um mapa de tópicos pode ser visto como uma coleção de índices interconectados. Tópicos e associações permitem definir, de forma estruturada, a semântica de um conjunto de recursos de informaçção. Esta rede hierárquica de tópicos é chamada ontologia.

* Bolsista CNPq - Brasil

Um extrator de ontologias baseado em XTM (*XML Topic Maps*) foi apresentado em [**?**]. Este ambiente processa um conjunto de documentos XML – pertencentes à mesma família, ou seja, documentos que respeitem o mesmo DTD ou XML Schema – com um extrator (*TM-Builder*), criado a partir de uma especificação da ontologia em XSTM (*XML Specification for Topic Maps*). O TM-Builder produz um documento XTM, o qual contém a ontologia extraída de acordo com a especificação XSTM. Essa plataforma é, portanto, totalmente baseada em XML.

Porém, quando os recursos de informação não são documentos XML (como acontece em vários projetos concretos), é necessário proceder, previamente, a uma conversão das fontes em causa para XML. Esta tarefa não é a melhor escolha, pois a sincronização dos dados é complexa; quando o recurso original é modificado é necessário gerar novamente o documento XML, atualizando o seu conteúdo.

Como solução para essa dificuldade prática, apresenta-se aqui um novo construtor de Topic Maps, chamado *Oveia*. O *Oveia* evita a transformação de recursos de informação não-XML para documentos XML, pois ele extrai as informações diretamente dos recursos.

A ontologia a ser extraída é especificada em XS4TM (XML Specification for Topic Maps). Esta linguagem tem o mesmo objetivo que XSTM, porém está um nível acima: XS4TM cobre todos os elementos da especificação XTM, além de ser projetada para a extração de ontologias em recursos de informação heterogêneos.

O *Oveia* cria uma base de dados com a estrutura definida de acordo com o paradigma Topic Maps [**?**] contendo todos os tópicos e as associações entre eles, chamada *BD Ontologia*. Na verdade, o *Oveia* pode armazenar os topic maps extraídos tanto na BD Ontologia, como no formato XTM[1].

O artigo inicia apresentando o paradigma Topic Maps na seção 2; Topic Maps é um formalismo para representar conhecimento sobre um recurso de informação, organizando por tópicos. A descrição do sistema que propõe-se, o extrator de Topic Maps a partir de recursos heterogêneos de informação – *Oveia* – é feita na seção 3. A definição da linguagem XS4TM será encontrada na seção 3.5. Por fim, uma síntese do artigo e os trabalhos futuros são apresentados na conclusão.

2 Topic Maps

Topic Maps [**?**] é um formalismo para representar conhecimento acerca da estrutura de um conjunto de recursos de informação e para o organizar em *tópicos*. Esses tópicos têm ocorrências e associações que representam e definem relacionamentos entre os tópicos. A informação sobre cada tópico pode ser inferida ao examinar as associações e ocorrências ligadas ao tópico. Uma coleção desses tópicos e associações é chamada *topic map*. Também pode ser visto como um paradigma que permite organizar, manter e navegar pela informação, permitindo transformá-la em conhecimento. Falar sobre Topic Maps, é falar sobre estrutura de conhecimento.

[1] O que basicamente é um documento XML onde diferentes elementos são usados para representar: tópicos, ocorrências de tópicos e relacionamentos (ou associações) entre os tópicos [**?**].

Um mapa de tópicos expressa a opinião de alguém sobre o que os tópicos são, e quais as partes do conjunto de informação que são relevantes para cada tópico.

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

Tópicos são o ponto principal de Topic Maps [**?**]. Em um sentido mais genérico, um tópico pode ser qualquer coisa: uma pessoa, uma entidade, um conceito. Eles constituem a base para a criação de Topic Maps (TM). Cada tópico tem um tipo de tópico (*topic type*), ou talvez múltiplos tipos. Cada tipo de tópico pode ser visto como uma típica relação classe-instância.

Ao analisar Topic Maps, identificam-se duas camadas distintas: os tópicos e as ocorrências. Os tópicos podem ser divididos em duas partes: os que representam conceitos abstratos e os que representam conceitos concretos. A ontologia é definida pelos conceitos abstratos, ou seja, os que serão instanciados por outros tópicos; por exemplo: tipo de tópico, tipo de associação e pelo tipo de papel de atuação em ocorrências.

Os tópicos restantes formam a base de conhecimento associada à ontologia, os quais compõem um conjunto de objetos de informação que permite organizar e indicar os reais recursos de informação (um objeto pode ter múltiplas ocorrências nos recursos de informação). A figura 1 dá uma representação esquematizada desta visão.

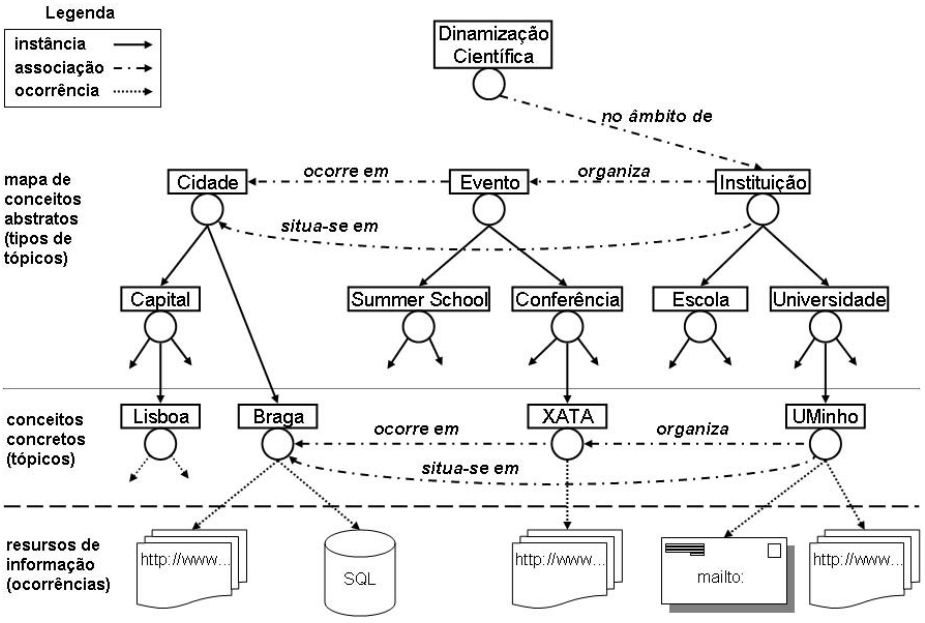


Fig. 1. O Mapa do Conceito *Dinamização Científica*.

O conceito de associação (*association*) permite descrever relações entre tópicos. Uma associação é (formalmente) um elemento de vínculo que define uma relação entre dois ou mais tópicos. Um ilimitado número de tópicos podem ser relacionados por uma associação.

3 Oveia – Um Extrator de Topic Maps a partir de Recursos de Informação

O *Oveia* é um extrator de ontologias em sistemas heterogêneos de informação baseado em Topic Maps. O *Oveia* foi desenvolvido com o objetivo de suprir as deficiências encontradas pelas atuais ferramentas de extração de ontologias. O *Oveia* é uma seqüencia do projeto que resultou no *TM-Builder* [**?**], o qual fornece um modelo de extração que consiste de uma especificação da ontologia a ser extraída.

Um fato que representa a evolução do *Oveia* em relação à sua versão inicial é a capacidade de extrair ontologias a partir de fontes de dados diversas. No *TM-Builder*, quando a fonte de informação é diferente de um documento XML, há uma necessidade uma conversão desta fonte para um arquivo XML. Portanto, considerando que a maior parte dos recursos de informação em empresas e instituições estão armazenadas em base de dados, para realizar uma extração de uma ontologia a partir delas, inicialmente seria necessário a geração de documentos XML com o conteúdo da base de dados.

Assim como o *TM-Builder*, o *Oveia* faz uso de uma linguagem de especificação de extração (XS4TM), a qual permite extrair Topic Maps de forma genérica e adaptativa. A especificação de extração de ontologias em XS4TM tornou-se mais flexível e completa, contemplando todos os elementos do padrão Topic Maps [**?**]. Isso garante maior flexibilidade de especificação para propósitos diversos de extração.

Na fase de especificação dos recursos de informação, é possível fazer transformações e filtros nessas fontes de dados, pois é utilizada a linguagem de consulta de cada recurso para sua especificação.

A linguagem de especificação de extração foi inspirada no modelo XTM. Isso significa que a especificação da ontologia a ser extraída (em XS4TM) é feita em um esquema XML similar ao esquema de XTM. Essa característica permite maior facilidade de compreensão da especificação proposta, pois o modelo XTM é um padrão que vem sendo adotado amplamente pela comunidade acadêmica. Assim, o projetista da ontologia apenas deve conhecer a sintaxe de XTM e a estrutura das fontes de dados, para estar habilitado a especificar extrações de ontologias em XS4TM.

A arquitetura do *Oveia* pode ser expressa conforme demonstra a figura 2: inicialmente, é feita em uma especificação XSDS (*XML Specification for DataSources/DataSets*), a qual define quais dados que devem ser recuperados pelo *Extrator de Datasets*; a informação extraída é armazenada em um formato intermediário, chamado *Datasets*. O próximo passo é a especificação da ontologia em XS4TM; essa fase determina o que é relevante para a extração dos tópicos e associações, assim como clarifica os limites que devem ser impostos ao topic map. O processador XS4TM recebe os *datasets* gerados e a especificação da ontologia na linguagem XS4TM (*XML Specification for Topic Maps*) e gera o topic map final. Por fim, o *Oveia* armazena o topic map gerado na BD Ontologia ou no formato XTM.

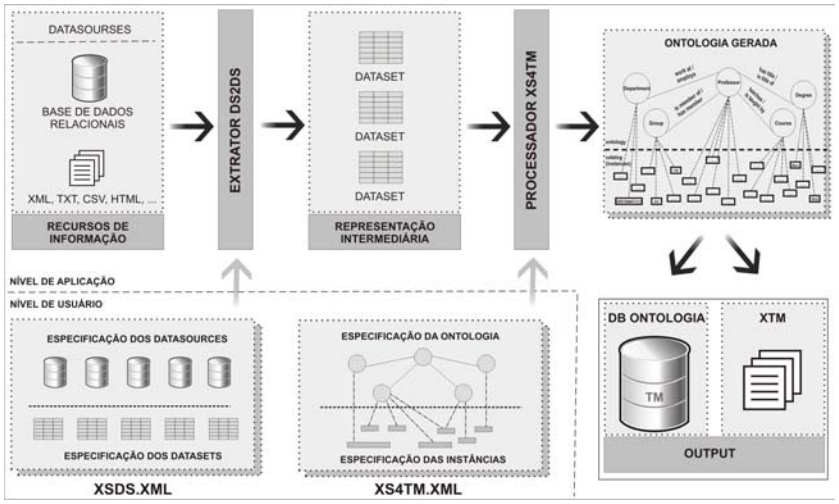


Fig. 2. Arquitetura do Oveia

As próximas sub-seções apresentam cada um dos componentes do *Oveia*.

3.1 Recursos de Informação: Os *Datasources*

Este componente é composto pelos recursos de dados: bases de dados, documentos XML, páginas HTML, etc. Ao final do processo de extração de ontologias, os recursos manterão-se inalterados, ou seja, o *Oveia* não modifica as fontes; somente copia as partes relevantes de informação para a construção do topic map. Esses recursos de dados são mapeados para uma representação intermediária, chamada *datasets*. Esse mapeamento é descrito pela linguagem XSDS.

3.2 Representação Intermediária da Informação: Os *Datasets*

Os *datasets* são a representação intermediária que contém os dados extraídos das fontes de informação. Cada *dataset* tem uma relação com uma entidade dos *datasources*, e seu conteúdo é representado na forma de uma tabela, onde cada linha é um registro segundo a estrutura definida em XSDS. Os *datasets* garantem que o *Oveia* tenha uma visão uniforme sobre a estrutura de dados que representam as fontes de dados participantes.

Cada *dataset* tem uma identidade única, a qual será usada pelo Oveia para o referenciar. A idéia fundamental é que todos os objetos tem rótulos que descrevem o seu conhecimento. Por exemplo, o seguinte objeto representa um registro da categoria de tipo de professor: <1,PhD>, onde ”1” é o identificador da categoria, enquanto que ”PhD” é um rótulo legível por humanos. Os *datasets* são simples, enquanto provém um poder de expressividade e flexibilidade necessário para integrar sistemas de informação de diferentes fontes.

3.3 XSDS: Especificação das Fontes de Dados da Extração

XSDS (*XML Specification for DataSources/DataSets*) é a linguagem definida com o intuito de especificar quais fontes de informação fornecerão dados para a criação de topic maps, de acordo com uma ontologia posteriormente especificada. Essa especificação fornece todos os elementos necessários para especificar as fontes de dados passíveis de extração de informação.

De um modo formal, a gramática de XSDS é dividida em duas partes: a definição dos *datasources* e a definição dos *datasets*. A primeira parte refere-se aos recursos físicos, ou seja, define-se quais fontes reais de informação serão usadas para a obtenção de dados; a segunda parte refere-se a quais campos de dados das fontes de informação devem ser extraídos, usando a linguagem de query de cada fonte em questão. Assim, pode-se dizer que a partir de um mesmo *datasource*, podem ser construídos vários *datasets*.

A definição da arquitetura do extrator foi idealizada para suportar extensão a diversos recursos de informação como fontes de dados. Para isso, essa arquitetura baseia-se no conceito de *drivers* de extração.

Especificação dos Datasources e Datasets em XSDS: Os *datasources* definem a localização física do recurso de informação. A declaração de cada uma das fontes de informação é feita no elemento <*datasource*>. Este elemento possui um atributo, chamado *extractorDriver* que indica qual *driver* de extração será utilizado: de acordo com o tipo de fonte de informação. Por exemplo: no caso de uma base de dados, além da localização da mesma, são passados parâmetros como o usuário e a senha a ser utilizada nesta base de dados, juntamente com o *driver* de extração que fará este processo; no caso da fonte de informação ser um documento XML, é necessário apenas o nome do arquivo com o seu caminho na árvore de diretórios do sistema operacional.

Para cada conjunto de dados dos recursos de informação (*datasets*) que se queira mapear a partir dos recursos de informação, é necessária a declaração do elemento <*dataset*>. Neste elemento, é necessário indicar qual fonte de dados provém os dados para a construção do *dataset* em questão.

O conteúdo do elemento <*dataset*> é uma expressão na linguagem de consulta referente ao tipo da fonte de informação. Caso esta fonte seja uma base de dados, o conteúdo deste elemento será uma expressão SQL para recuperar os dados referentes ao *dataset* em questão. Se a fonte de informação é um documento XML, o conteúdo deste elemento será uma expressão XPath, indicando o caminho para a informação deste *dataset*.

3.4 O Extrator DS2DS

O *Extrator DS2DS* (*DataSource to DataSet*) é um processador que extrai dados de recursos de informação e faz a criação dos *datasets*, de acordo com a especificação XSDS. Este componente processa uma especificação XSDS, a qual especifica a fonte dos dados a serem extraídos (*datasources*) e o destino das informações extraídas, as quais definem a representação intermediária (*datasets*).

Esta representação intermediária é composta por um conjunto de tabelas que contém a informação extraída dos datasources. Estas tabelas contém somente os dados selecionados nos elementos *datasets* da especificação XSDS em questão.

O *Extrator DS2DS* possui diversos *drivers* de extração que, como dito anteriormente, são os módulos responsáveis pela extração de informação das fonte de dados; portanto, há um *driver* desenvolvido para cada tipo de recurso de informação.

Atualmente, o protótipo do Oveia possui dois drivers implementados: para conectar com base de dados relacionais (br.uneb.dcet.tmbuilder.drivers.DataBase) e para recuperar informação de documentos XML (br.uneb.dcet.tmbuilder.drivers.XMLFile). A implementação de novos *drivers* para outros recursos de informação é um processo relativamente fácil e pode ser realizado conforme a necessidade.

3.5 XS4TM: Uma linguagem XML para especificar a extração de Topic Maps

A linguagem XSTM, proposta em [**?**], foi inicialmente definida como sendo um dialeto XML para especificar o topic map que se pretende construir ao analisar documentos anotados pertencentes a um mesmo esquema XML. Por essa definição, o XSTM está diretamente ligado a extrações a partir de documentos XML. Por outro lado, a necessidade de abranger novas fontes de dados fez com que se propusesse uma nova arquitetura para extração de ontologias. Dessa forma tornou-se necessário repensar e redesenhar o XSTM; o qual passa a ser denominado por XS4TM.

Cada especificação XS4TM é uma instância XML. Portanto, na prática a linguagem XSTM é definida por um DTD (e/ou um XML-Schema), de modo a permitir o uso de todos os ambientes de processamento XML.

A linguagem XS4TM tem por objetivo tornar a especificação da extração de Topic Maps mais completa e flexível. XS4TM é caracterizado por transformar o atual padrão XTM em um subconjunto da sua especificação.

O XS4TM possui dois elementos principais: *ontologies* e *instances*. Cada um destes elementos têm a estrutura de acordo com a especificação XTM. A única diferença está nos subelementos de *instances*, pois os elementos *topic* e *association* possuem um novo atributo. Esse atributo é denominado *dataset* e é utilizado para identificar que o determinado elemento (tópico ou associação) será construído a partir de um *dataset* em específico. Este atributo faz uma referência ao nome do *dataset*, declarado no documento de especificação XSDS; a figura 3 demonstra essa referência entre o *dataset DS_aluno* declarado em XSDS e o seu uso na especificação XS4TM.

Para o preenchimento das informações referentes a cada tópico, é necessário buscar tal informação no *dataset* que a contém. Assim, identifica-se essas propriedades com a expressão:

$$@ + \text{"dataset"} + \text{"."} + \text{"atributo"}$$

Mais detalhadamente, isto significa:

- O @ apenas indica que esta declaração é referente a uma propriedade de um *dataset*;
- Após o @, encontra-se o identificador do *dataset* (especificado em XSDS) ao qual deseja-se recuperar a informação. No exemplo da figura 3, o *dataset* selecionado é o *DS_aluno*.

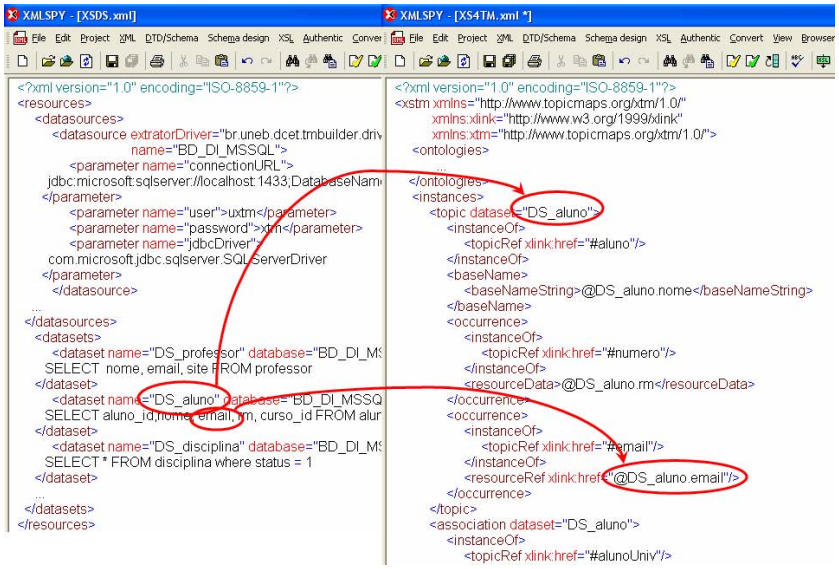


Fig. 3. Relações entre XSDS e XS4TM

– O *atributo* é uma referência ao campo do *dataset* que contém a informação desejada. Na figura 3, o atributo recuperado para a construção da ocorrência do tipo *email* é o campo *email* extraído pelo *dataset DS_aluno*.

Desta forma, cria-se uma forma de habilitar o uso das informações contidas nos *datasets*.

3.6 Processador de XS4TM

Este componente utiliza a especificação XS4TM para selecionar quais campos dos *datasets*, extraídos dos recursos de informação, são necessários para a formação do topic map. Este processador é um interpretador que tira vantagem da organização das informações em um formato uniforme.

O seu processo de execução pode ser resumido em três passos: (1) ler a especificação XS4TM e extrair os dados especificados que encontram-se nos *datasets*; (2) criar o topic map baseado na própria especificação XS4TM; (3) armazenar o topic map gerado na BD Ontologia ou em um documento no formato XTM.

3.7 Base de Dados de Ontologias

Um dos diferenciais desta ferramenta é o armazenamento dos Topic Maps extraídos em uma base de dados relacional. De acordo com [**?**], referente aos métodos de mapeamento de documentos XML para o modelo relacional, adotou-se na *BD Ontologia* o modelo de mapeamento por estrutura.

Conforme o mapeamento por estrutura, foi criada uma tabela para cada elemento de XTM 1.0 DTD. Esse processo consiste em identificar as características e os tipos de associações entre os elementos do DTD e representa-los no modelo relacional.

Pode ser entendido facilmente tomando um trecho desse mapeamento, como apresenta o segmento do XTM 1.0 DTD abaixo e a figura 4.

```
<!ELEMENT topic (instanceOf*, subjectIdentity?, (baseName | occurrence)*)>
<!ATTLIST topic
    id ID #REQUIRED
>
<!ELEMENT baseName (scope?, baseNameString, variant*)>
<!ATTLIST baseName
    id ID #IMPLIED
>
```

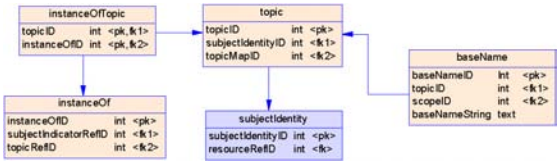


Fig. 4. Trecho do Modelo ER do BD Ontologia

A facilidade de compreensão desse modelo é garantida principalmente pelo fato de que este modelo segue o padrão XTM, o qual é bastante conhecido pela comunidade acadêmica. Essa foi umas das vantagens trazidas por essa opção de modelagem, preservando o padrão Topic Maps. Assim, a partir dessa base de dados, é possível navegar no Topic Maps utilizando consultas SQL.

4 Trabalhos Relacionados

O KAON[2] é um projeto *open-source* que fornece uma infra-estrutura para gestão de ontologias, voltado para aplicações de negócios. A ferramenta KAON REVERSE é um *plug-in* do *framework* KAON. O KAON REVERSE permite o mapeamento de bases de dados relacionais para uma ontologia, com o objetivo de extrair instâncias e relacionamentos entre instâncias, a partir da base de dados. Dentre as ferramentas conhecidas, esta é a que mais se aproxima do *Oveia*.

A tabela 1 mostra as características e funcionalidades do *Oveia* e do KAON REVERSE[3].

Analisando a tabela 1, é difícil dizer qual a melhor ferramenta, visto que é clara a complexidade entre ambas; apetece até dizer que o melhor dos mundos resultaria de sua fusão.

[2] Mais informações em: http://kaon.semanticweb.org/

[3] Os dados presentes nesta tabela foram utilizados a partir da análise feita em [**?**].

	KAON REVERSE	OVEIA
Linguagem	Java	Java
Uso de APIs	Sim	Sim
Uso de Engenharia Reversa	Sim	Não
Especificação	Árvore(GUI)	Documento XML
Fontes de Extração de Ontologias	BDs relacionais via JDBC	BDs relacionais via JDBC, XML, extensível a outras fontes
Padrão de Representação de Ontologias	RDF	Topic Maps
GUI (Interface Gráfica)	Sim	Não
Resultado Gerado	Documento RDF	Base de Dados de Ontologias

Table 1. Comparativo entre KAON REVERSE e Oveia.

Partindo deste ponto de vista, destaca-se as vantagens de cada uma. Por um lado, a KAON REVERSE apresenta vantagens em relação ao uso de uma interface gráfica para a especificação de ontologias e o uso de engenharia reversa das fontes de dados para auxiliar o mapeamento. Por outro lado, o *Oveia* destaca-se por ser mais flexível em relação às fontes de dados passíveis de extração (mesmo que todos os drivers de extração não estejam implementados) e em relação ao processo de especificação. Além disso, o *Oveia* diferencia-se por gerar uma base de dados de ontologias capaz de manter os dados extraídos.

Ao fazer uso de uma linguagem de especificação (XS4TM) semelhante à linguagem padrão que é usada para escrever os XML Topic Maps, o *Oveia* facilita o processo de extração ao projetista da ontologia, o que é uma clara vantagem, visto ser este o foco do sistema. Assim, torna-se muito simples e conciso criar uma nova visão conceitual diferente sobre as mesmas fontes.

5 Conclusão

O objetivo deste artigo foi a apresentação de uma arquitetura para a construção automática de Topic Maps, a partir da extração de informação de fontes de dados diversas. Esse sistema, designado por *Oveia*, resultou de uma proposta inicial denominada *TM-Builder*, o qual aborda um extrator de ontologias totalmente baseado em XML.

A extração de informação de fontes heterogêneas de informação é especificada pela linguagem XS4TM, a qual define quais os conceitos e relações encontradas nestas fontes de informação que serão mapeadas para tópicos e associações, respectivamente, no Topic Maps gerado pelo *Oveia*.

XS4TM é a linguagem para especificar a extração de Topic Maps a partir de recursos de informação. XS4TM classifica os tópicos, dando-lhes uma semântica mais concreta, através da associação de um tipo de tópico, um tipo de associação ou um tipo de papel de atuação em ocorrências. Então, do ponto de vista de descrição da ontologia, obtém-se ganho por se dispor de uma semântica mais precisa.

O *Oveia* é uma evolução do *TM-Builder*, o qual estava limitado ao processamento de documentos XML. A fim de evitar esse problema, foi construída uma nova arquitetura

que provê uma abstração dos tipos de fontes de dados baseada em *drivers* de extração. O resultado obtido foi uma camada de independência da fonte de dados, que torna possível a extração em fontes de dados diversas, de forma transparente e em uma única especificação.

A principal vantagem desta proposta é a possibilidade de adaptação do processo de especificação e extração de ontologias a um maior número de casos reais, sem acarretar em mudanças de formato do recurso de informação.

Outra vantagem é o que eventuais modificações nas fontes de informação (obviamente mudanças ao nível de seu conteúdo, e não estruturais – incluindo-se novos dados ou excluindo-os), não torna necessária uma modificação da especificação XS4TM; basta voltar a executar o extrator com a fonte de informação com os novos dados inseridos (ou retirados).

Um dos projetos a ser desenvolvido em breve será um módulo que permita a conversão de um documento XTM para uma BD Ontologia, assim como possibilite a extração de um Topic Map da BD Ontologia para um documento XTM. Com este módulo, o utilizador pode rapidamente ter um conjunto de topic maps armazenados no formato desejado, seja em documentos XML (no padrão XTM) ou em base de dados relacionais (em uma BD Ontologia).

A fusão entre Topic Maps, conhecida como *merge*, não foi considerada nesta versão de XS4TM. Isso implicaria um tratamento mais complexo no processo de extração, o que não seria viável no presente momento.

O *Oveia* não possui um ambiente amigável para o utilizador criar suas especificações. Aparentemente o processo de criação de um documento XS4TM ou XSDS mostra-se trabalhoso. Isso exige que o utilizador conheça o padrão dos documentos de especificação, que é o padrão XTM. Por outro lado, é sabido que esta tarefa pode ser auxiliada por ferramentas de edição de documentos XML, como exemplo o XMLSpy[4], mas essa função de criar uma interface de especificação agradável e fácil de usar será um outro tópico de investigação futura.

[4] Ferramenta para construção de documentos XML, desenvolvido pela ALTOVA. Mais informações em: http://www.altova.com

Parte VI

Dialectos XML

JavaML 2.0: Enriching the Markup Language for Java Source Code

Ademar Aguiar[1], Gabriel David[1], and Greg Badros[2]

[1] Faculdade de Engenharia da Universidade do Porto and INESC Porto
{aaguiar,gtd}@fe.up.pt
[2] Google Inc., 2400 Bayshore Parkway, Mountain View, CA 94043
badros@cs.washington.edu

Resumo Although the representation of source code in plain text format is convenient for manipulation by programmers, it is not an effective format for processing by software engineering tools at an abstraction level suitable for source code analysis, reverse-engineering, or refactoring. Textual source code files require language-specific parsing to uncover program structure, a task undertaken by all compilers but by only a few software engineering tools. JavaML is an alternative and complementary XML representation of Java source code that adds structural and semantic information into source code, and is easy to manipulate, query, and transform using general purpose XML tools and techniques. This paper presents an evolved version of JavaML, dubbed JavaML 2.0, and the enhancements made to the schema and respective converters: DTD and XML Schema support, cross-linking of all program symbols, and full preservation of original formatting and comments. The application of JavaML 2.0 is illustrated with concrete examples taken from the software documentation tool that motivated the enhancements.

1 Introduction

Since the first computer programming languages, programmers have used a plain-text format for encoding software structure and computation. The immediate users of this format include the compiler, which converts sequences of characters into a data structure that closely reflects the program structure — an *abstract syntax tree* (AST).

Despite the advances in compilers and document management tools, software engineers regularly manipulate source code using the plain text format, often employing superficial tools based on regular expressions.

Although the plain-text representation of source code is convenient for programmers and has nice properties, pure text is not the most effective format for manipulating source code at an abstraction level suitable for software engineering tools. Plain-text files are good for lexical analysis, but they are not suitable for structural and semantic analysis before being parsed and translated to higher level formats. There is thus a need for a standard format capable of directly representing program structure and semantics, a readable and widely supported format that tools can easily analyze and manipulate.

JavaML is a markup language for Java source code proposed by Greg Badros [1,2] that provides an alternative representation for Java source code. JavaML enriches source code text files with structural and semantic information typically present in a compiler annotated ASTs. Because the representation is XML-based, JavaML is easy to manipulate, query, and transform using general purpose and widely available tools and techniques.

This paper presents an evolved version of JavaML, dubbed 2.0, that incorporates enhancements and new features beyond the original JavaML [1], including: XML Schema support, updated converters, cross-referencing of all program symbols, and full preservation of original lexical information (formatting and comments). These enhancements were motivated by the implementation of XSDoc, an extensible infrastructure for documenting object-oriented frameworks [3].

The next section briefly discusses the benefits of using an XML format for representing source code. The sections that follow describe JavaML 2.0 and present concrete examples of application taken from the XSDoc infrastructure. The paper then reviews related work and concludes by suggesting ideas for future work.

2 Why represent source code in XML?

The plain-text representation of source code is convenient to express programmers ideas. It is concise, easy to read by programmers, and very simple to exchange and manipulate using a wide variety of tools, such as text editors, version control systems, and file system utilities. But the plain-text representation of source code also has many limitations. Perhaps the most significant is that the program structure is not directly represented in the format and thus requires language-specific processing to uncover it.

2.1 XML

XML is a universal format widely used to represent structured information in text-based files that was designed to be lightweight and simple. An XML document consists of text marked up with tags enclosed in angle braces.

XML documents have an inherent hierarchical structure and are therefore convenient for representing source code constructs. XML-based representations are easy to understand, easy to manipulate by tools, very flexible, extensible, and widely supported. Although XML documents are primarily intended for automatic processing by tools, the format is human readable.

XML documents are therefore an empowering complementary representation for source code, enabling the usage of tools at a higher level of abstraction.

2.2 Limitations of plain text for representing source code

Despite its nice properties, plain text source code files require parsing to uncover the most important information they contain about a program: the structural

and semantic information. This severe limitation forces the inclusion of language-specific parsers in each tool that needs to manipulate programs at an abstraction level higher than the lexical. While heavyweight software engineering tools can afford to embed such parsers, simple utility tools do not and are thus limited to lexical tasks.

Common manipulations of source code, such as generation, refactoring, reformatting, or reverse-engineering, usually require the manipulation of more abstract source code representations equivalent to ASTs.

Although modern integrated development environments (IDE) provide application programming interfaces (API) to manipulate in-memory AST representations of source code, these APIs are still not appropriate for simple tools, since they require integration in a complex environment — the IDE — that creates a strong and undesirable dependency.

2.3 Benefits of representing source code in XML

The representation of source code in XML has several benefits [1,4,5] and enables the use of more powerful software engineering and document management methods and tools.

Explicit code structure. XML documents are structured by nature, and can be used to build tree-like code representations and sophisticated manipulation of source code using general purpose XML tools. Some good examples are automatic synchronization of generated code and documentation, or quick code generation and transformation using predefined templates. With source code in XML it is possible to annotate generated code with its template definition so that the constructed code can be regenerated every time its template is updated. Complex templates for custom instantiation of design patterns would benefit from such mechanisms.

Powerful querying capabilities. In addition to textual searches using regular expressions, modern IDEs usually include tools specific for source code that allow searching for common programming language constructs, such as classes, methods, and fields. These features are useful but are only a small subset of what is possible to query with XML standards and tools, such as XPath [6] and XQuery [7].

Extensible representation. Plain-text source code is not easy to extend with new code constructs or annotations because they would disrupt the code structure and require parser modifications. Because of this, such extensions are often embedded as comments. In an XML document, the addition of new elements is much easier, because the structure is explicitly marked up, and we can separate different kinds of elements (e.g. language specific elements and user-defined elements) using XML namespaces. Distinct tools can define and insert their own

elements in the code structure and then process only those that are relevant for them. Examples of common extensions are code annotations, comments, meta-information, authoring and version information, revisions, documentation and conditional code.

Flexible formatting. Most programming languages, including Java, ignore the semantic value of whitespace and enable programmers to adopt diversified formatting conventions, which, despite its usefulness, are not easy to enforce and maintain in consistency. Mistakes in following formatting conventions sometimes result in an increased difficulty to locate structural or semantic errors, due to the suggestive nature of formatting. In an XML representation of code, the structure can be abstracted from the coding style. XML standards and tools, such as XSLT [8] facilitate the (re)formatting of source code using different styles which can enrich its readability through an appropriate usage of layouts, colors, fonts, and links.

Cross-referencing. Source code fragments in plain text are usually referenced by their file position, i.e. line and column numbers. In an XML-based structure of a program, it is possible to associate code fragments directly with code constructs, thus enabling the relocation of code fragments without disrupting references.

Wide support. A program representation must be widely supported in a wide variety of platforms, otherwise it can't succeed. XML tools are available on all major platforms and thus satisfy this requirement.

3 JavaML 2.0

The original JavaML markup language provides a complete self-describing representation in XML for Java source code. Unlike the classical plain-text representation of programs, JavaML reflects the structure of Java programs directly in the hierarchical structure of XML documents.

Because JavaML uses the XML format, a text-based representation, many of the advantages of the classical source representation remain. In addition, the JavaML representation is easy to parse and manipulate with general purpose XML tools, not requiring language-specific tools that are difficult to implement and maintain. Therefore, JavaML leverages the development of XML tools for the manipulation of Java source code in JavaML.

The immediate users of JavaML format are tools, and not is intended to be written directly by hand. Nevertheless the format is easily readable and understandable, enabling its direct inspection by developers.

JavaML 2.0 enriches the original JavaML [1] with more information at several levels of abstraction ranging from the lexical level to the semantic level.

The enhanced representation of JavaML 2.0 now includes full lexical information about tokens, comments and formatting, only small enhancements in terms of structural information, and much richer semantic information related with symbol definitions, references and type information.

3.1 Background on JavaML

In order to represent Java source code in XML there are many possible approaches [1]. Consider the following source file for the class `FirstTest`.

FirstTest.java

```
1  package junit.samples;
2  import junit.framework.*;
3
4  /**
5   * My first unit test.
6   */
7  public class FirstTest extends TestCase {
8     double value = 2.0;
9
10    public void testAdd() {
11       double result = value + 3;
12       // forced failure result == 5
13       assertTrue(result == 6);
14    }
15 }
```

The most obvious approach is to dump the derived AST to a XML format, but this would result very verbose and uninteresting due to the irrelevant grammar details it would reveal.

Another possibility is to markup the Java source program without changing the original text. The result would be a richer representation, easier to convert back to the original source file, but the retrieval of specific information from the representation would require undesired lexical analysis of element contents.

The representation chosen for JavaML aimed to model Java programming language constructs without binding to the specificities of the language syntax [1]. As a result of this design principle, JavaML can be used as a base for the design of a generalized markup language supporting other object-oriented programming languages, such as C#, C++ or Smalltalk.

To illustrate the approach followed by JavaML, consider the source code file presented above (`FirstTest.java`) and its corresponding basic JavaML representation (`FirstTest.java.xml`). The major design decisions are enumerated below.

FirstTest.java.xml

```
<?xml version="1.0" encoding="UTF-8"?> <!-- FirstTest.java.xml -->
<java-source-program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ... >
 <java-class-file name="f:/junit3.8.1/src/junit/samples/FirstTest.java">
  <package-decl name="junit.samples"/>
  <import module="junit.framework.*"/>
  <class name="FirstTest" id="Ljunit/samples/FirstTest;">
   <doc-comment>/**&#xA; * My first unit test.&#xA; */</doc-comment>
   <modifiers>
    <modifier name="public"/>
   </modifiers>
   <superclass name="TestCase" idref="Ljunit/framework/TestCase;"/>
   <field name="value" id="Ljunit/samples/FirstTest;value">
    <type name="double" primitive="true"/>
    <var-initializer>
     <literal-number kind="double" value="2.0"/>
    </var-initializer>
   </field>
   <method name="testAdd" id="Ljunit/samples/FirstTest;testAdd()V">
    <modifiers>
     <modifier name="public"/>
    </modifiers>
    <type name="void" primitive="true"/>
    <formal-arguments/>
    <block  >
     <local-variable-decl>
      <type name="double" primitive="true"/>
      <local-variable name="result" id="Ljunit/samples/FirstTest;var1946">
       <var-initializer>
        <binary-expr op="+">
         <field-ref name="value" idref="Ljunit/samples/FirstTest;value"/>
         <literal-number kind="integer" value="3"/>
        </binary-expr>
       </var-initializer>
      </local-variable>
     </local-variable-decl>
     <send message="assertTrue" idref="Ljunit/framework/Assert;assertTrue(Z)V">
      <arguments>
       <binary-expr op="==">
        <var-ref name="result" idref="Ljunit/samples/FirstTest;var1946"/>
        <literal-number kind="integer" value="6"/>
       </binary-expr>
      </arguments>
     </send>
    </block>
   </method>
  </class>
 </java-class-file>
```

Representation of code constructs. JavaML represents the important concepts of the Java language, namely classes, superclasses, fields, methods, variables, message sends, and literals, directly in document elements and attributes.

Code structure is reflected in the nesting of elements. Program structure is reflected in the nesting of elements. Figure 1 presents a visual presentation of the document tree, which shows the nesting of the literal number `3` in the initializer part of the variable declaration it appears.

Generic elements. In order to reduce the size and complexity of the schema, JavaML generalizes related concepts and represents them using generic elements

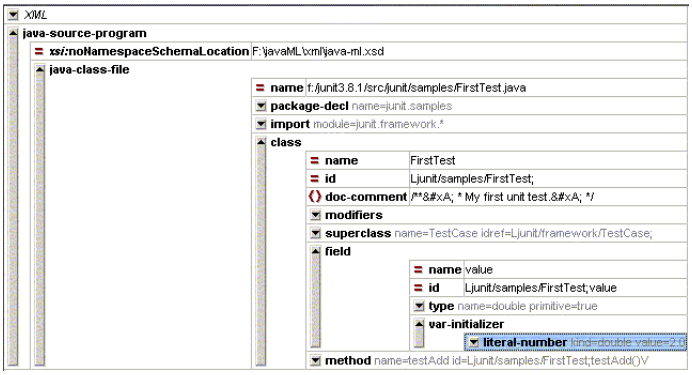


Figura 1. Tree view of the JavaML representation[9].

with different attribute values. Loops and literal numbers are just two examples of such generalizations. In `FirstTest.java.xml`, the lines 15 and 31 contain `literal-number` elements with different `kind` attribute values to disambiguate the representation of a `double` and an `integer`. In addition, unstructured information is stored in attribute elements as illustrated in the `modifier` element, in line 20.

3.2 The new JavaML 2.0 schema

The major enhancements of JavaML 2.0 consisted on improving the schema to accommodate richer lexical and semantic information. Our primary goal is to enable full preservation of original source files and complete cross-linking of program symbols. The corresponding converters were re-implemented to cope with these new schema requirements.

These enhancements were mainly motivated by the implementation of XSDoc, an infrastructure for framework documentation that uses JavaML to dynamically integrate source code in the documentation.

All JavaML and JavaML 2.0 artifacts referred in this paper are available online [2,10].

DTD and XML Schema support. Both XML Schema and DTD provide a basic grammar for defining XML documents in terms of the metadata that comprise the shape of the document. XML Schemas are themselves XML documents. XML Schemas provide a more powerful means to define a XML document structure and validations than DTDs, because provide an object-oriented approach, with all the benefits this entails, namely the ability to reuse and extend type definitions. Because both DTD and XML Schema have their own advantages, JavaML 2.0 is primarily designed for XML Schema but still supports DTDs. The JavaML 2.0 XML Schema has around 90 elements. The original DTD was four times shorter in terms of number of lines.

Cross-referencing of program symbols. JavaML representation includes information to link symbol references to their definitions. This linking is achieved through the standard XML mechanism using `id` attributes in definitions and `idref` attributes in references. Although not strictly necessary, JavaML considers variables, fields, and arguments as distinct kinds of symbols. In line 39, we can see a reference to the local variable named `result` that points back to its definition in line 27.

FirstTest.java.xml

```
27      <local–variable name="result" id="Ljunit/samples/FirstTest;var1946">
39        <var–ref name="result" idref="Ljunit/samples/FirstTest;var1946"/>
```

Because these references are defined in the XML Schema, they are automatically checked when the document is validated. The listing below shows the lines that define the keys and references for `local-variable` elements in the JavaML 2.0 schema (`javaml.xsd`). The key is defined as a value appearing in the `id` attribute (line 213) of `local-variable` elements or `formal-argument` elements that are immediate children of `catch` elements (line 212). This key is referenced by values of the `idref` attribute (line 217) of `var-ref` elements (line 216). The values used in these keys and references are typical to symbol tables and are automatically generated by the Java compiler.

javaml.xsd

```
211       <xs:key name="KeyLocalVariable">
212        <xs:selector xpath=".//local–variable|.//catch/formal–argument"/>
213        <xs:field xpath="@id"/>
214       </xs:key>
215       <xs:keyref name="RefLocalVariable" refer="KeyLocalVariable">
216        <xs:selector xpath=".//var–ref"/>
217        <xs:field xpath="@idref"/>
218       </xs:keyref>
```

Type dependencies. Source code files usually have dependencies to other source files and libraries. During type checking and name resolution, all these files must be analyzed and a type dependency graph can be built. JavaML 2.0 representation includes these dependencies to complement the structural information presented before.

FirstTest.java.xml

```
48  <type–dependences>
49   <type–dependence filename="f:/junit3.8.1/src/junit/samples/FirstTest.java"
50         signature="Ljunit/samples/FirstTest;">
51    <type–ref signature="Ljunit/framework/TestCase;"/>
```

This information enables following references to the source code and documentation of external types. In line 6 of `FirstTest.java.xml`, we can see the value used to uniquely identify the class `FirstTest` and in line 11 there is a reference to the external type `TestCase`.

FirstTest.java.xml

```
6    <class name="FirstTest" id="Ljunit/samples/FirstTest;">
11    <superclass name="TestCase" idref="Ljunit/framework/TestCase;"/>
```

Preservation of formatting and comments. The preservation of comments and whitespace were two issues not completely addressed in the original JavaML implementation [1].

Although the comments are easy to store, they are challenging to attach to the correct elements. JavaML 2.0 preserves all comments present in source files and attaches the formal ones, i.e. Javadoc comments [11], to their respective code elements (`class`, `method`, `field`, etc.) using the rules defined by Javadoc. Informal comments (non-Javadoc) are deliberately not attached to code elements because the semantic inference of the respective code element based on their relative locations is not precise, as it is not possible to ensure that a specific comment near a code element contains information about that element.

To enable the exact regeneration of original source files, the JavaML 2.0 has new `codeline`, `token`, and `comment` elements to store all the lexical information of a source code file. In addition, each major programming element has three optional attributes to store the identifiers of the starting token (`startToken`), the ending token (`endToken`), and the respective comment (`endToken`). Below, we present the JavaML 2.0 lexical representation of line 1 of `FirstTest.java`.

FirstTest.java.xml

```
<java-source-code>
 <codeline no="1">
  <token idx="1" line="1" column="1" type="preprocessor" lexeme="package"
                afterEol="true"/>
  <sp/>
  <token idx="2" line="1" column="9"  type="normal" lexeme="junit"/>
  <token idx="3" line="1" column="14" type="normal" lexeme="."/>
  <token idx="4" line="1" column="15" type="normal" lexeme="samples"/>
  <token idx="5" line="1" column="22" type="normal" lexeme=";"/>
 </codeline>
```

Due to the verbosity of this information, it is optionally generated. In the following listing, we show how Javadoc comments are represented.

FirstTest.java.xml

```
 <codeline no="4">
  <comment idx="1" line="4" column="1" type="formal">/**</comment>
 </codeline>
 <codeline no="5">
  <comment idx="1" continued="true"> * My first unit test.</comment>
 </codeline>
 <codeline no="6">
  <comment idx="1" continued="true"> */</comment>
 </codeline>
```

3.3 Java to JavaML converter

Because JavaML was designed to be primarily manipulated by tools, it is mandatory to have a converter from Java source files to JavaML. The original approach consisted of adding one `XMLUnparse` method for each AST node of the IBM Jikes Java compiler framework (version 1.12)[12], resulting in a fast and robust JavaML converter [1].

To implement the schema enhancements of JavaML 2.0, the converter was initially migrated to Jikes 1.18 and then evolved to support the new requirements. The generation of lexical information is implemented by embedding Jikes scanner results in JavaML elements. JavaML elements are generated by visiting the AST and its associated symbol and type annotations. The generation of semantic information to assign to the `id` and `idref` attributes of each program symbol is the most challenging feature to implement because it requires navigation in the AST and lookups in the symbol table and type definitions. The overall code that adds JavaML support to Jikes is about 2000 lines of C++.

3.4 JavaML converters

Once generated, the JavaML representation can be easily processed using general purpose XML tools. Using an XSLT stylesheet it is possible to convert the JavaML representation to several other formats. Below we describe two converters: one for producing an HTML view of the source code and another for regenerating the original Java source file.

JavaML to HTML. The JavaML tools include an XSLT stylesheet that converts JavaML to HTML, named (`javaml-to-html.xsl`). Because JavaML 2.0 contains more lexical information than the original JavaML, the new stylesheet is simpler than the original [1]. It produces HTML that cross-links all symbol references (types, methods, variables, etc.) to their definitions in source code or documentation, depending on what is available. When compared to the original source code, the generated HTML view is more convenient for program understanding, because it enables good navigation from references both to internal and external definitions.

The conversion consists basically on two tasks: first, we apply a predefined style to each token, based on the kind it was assigned during scanning (literal, keyword, etc.); and, second, we define anchor elements (`<a name=>`) and reference elements ((`<a href=>`) for each symbol definition and symbol referenced, respectively. The most important part of the linking is the conversion of `id` and `idref` values to anchor names and references. The conversion is done with the help of the Java function `getLinkFromId()`, which receives symbol information (containing file and type, identifier, kind of symbol) and path information, and computes a unique link for that symbol. The new stylesheet (`javaml-to-html.xsl`) has 21 template rules and around 300 lines. Below is listed the line of the template rule `create-link` that creates and formats the anchor name element.

javaml-to-html.xsl

```
217   <xsl:value-of select="linker:getLinkFromId($javadoc-url,$filename,
218       $type-signature,$kind,key('KeyTypeSignature',$signature)/@filename,$id)"/>
```

JavaML to Java. The regeneration of Java source code is straightforward to implement because the JavaML representation contains low-level lexical information about the tokens, comments and spaces of the original source file. The corresponding XSLT stylesheet has only 30 lines and only 6 simple template rules. The most complex template rule is for processing `token` elements.

javaml-to-html.xsl

```
14 <xsl:template match="token">
15     <xsl:if test="not(@type='TK_EOF')">
16         <xsl:value-of select="@lexeme" disable-output-escaping="yes"/>
17     </xsl:if>
18 </xsl:template>
```

4 Applying JavaML

JavaML 2.0 is the result of evolving original JavaML [1] to fit the requirements of XSDoc [3], an extensible infrastructure based on a WikiWikiWeb engine that supports the creation, integration, publishing and presentation of documentation for object-oriented frameworks. XSDoc helps to create and annotate framework documents and to integrate different kinds of contents (text, models and source code). It provides a simple and economic cooperative web-based documentation environment that can be used standalone in a web-browser, or inside an integrated development environment.

To illustrate the application of JavaML 2.0, we present a concrete example taken from a simple usage of XSDoc for integrating a web document source code from the file `FirstTest.java` and some text. XSDoc provides two dynamic mechanisms for the integration and synchronization of the possible kinds of document contents (source code, UML diagrams and XML files): inlining of contents and automatic linking.

The inlining of Java source code is defined with a reference to the specific contents, annotated with the `<javaSource>` tags.

extract of a XSDoc wiki topic

```
See below the method for testing the addition:
[<javaSource>]junit.samples.FirstTest#testAdd(); comments=no;
lines=first, last; [</javaSource>]
```

For example, the text above extracts the code fragment corresponding to the method `testAdd()` of class `junit.samples.FirstTest`, then removes all its comments, and returns its first and last line. This produces the web page in Figure 2.

4.1 Generating JavaML

After parsing the Java source code reference (in Javadoc format) contained in the `javaSource` tag, XSDoc finds that the code to inline must be in the source code file named `FirstTest.java`. If the JavaML representation is outdated, it is updated by invoking the Jikes compiler with the appropriate arguments.

```
jikes +B +L +c +T=3 +ulx FirstTest.java
```

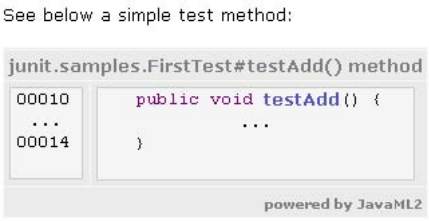


Figura 2. Example of a XSDoc page inlining code from `FirstTest.java`.

4.2 Filtering JavaML

The JavaML document is then filtered to get the requested method. Based on the source code reference, XSDoc builds a XPath query and applies it using Saxon [13].

extract of a query dynamically created by XSDoc

```
*[(name()='class' or name()='interface')
   and @name='FirstTest']/method[@name='testAdd'
   and ./formal_arguments/descendant_or_self::*[last()=1]]
```

After this structural filtering, the document is lexically filtered to remove the comments. Next, the first and last line are extracted.

4.3 Converting to HTML

Finally, the resulting document is converted to HTML using the XSLT stylesheet described before in section 3.4.

5 Related Work

There are various research activities involving XML grammars for modelling source code and describing analytical aspects of code. These activities would benefit from having source code already in XML. The work most closely related with JavaML are srcML [14] and cppML [4]. Other similar work abounds [1,5,14,15] .

Source Markup Language, srcML, is an XML format for source code markup that adds a layer on top of the original source code, leaving the source code untouched. The disadvantage of srcML compared to JavaML 2.0 is that in srcML the code is only semi-parsed because it doesn't include type specifications, for example, and in JavaML 2.0 the code is completely parsed and annotated with symbol and type information.

cppML is an XML grammar for C++ code that takes a similar approach to JavaML, but it doesn't provide a standalone cppML generator, requiring the usage of a compiler integrated in the VisualAge for C++ from IBM.

6 Conclusions and Future Work

XML-based representations for source code have several benefits over classical plain-text files that facilitate their manipulation by software engineering tools. They have an explicit structure, they contain information equivalent to an annotated abstract syntax tree, and they don't require language-specific parsing, but only general purpose processing using widely available XML tools. JavaML 2.0 is a rich alternate XML representation for Java source code evolved from the original JavaML that adds more source code information to the representation.

The JavaML 2.0 representation includes source code information at various levels of abstraction, starting from the lexical (tokens, comments, and formatting) and structural levels (abstract-syntax tree) to the semantic level (symbols, types and references). As a result, with JavaML 2.0 it is possible to convert from Java source code to XML files and convert back the representation to the original format without loosing information. Software engineering tools that intend to manipulate Java source code at above the lexical abstraction can now do it directly in JavaML representation without the effort of embedding Java language-specific parsers. JavaML 2.0 is thus an empowering representation that supports the development of more sophisticated software engineering tools for manipulating Java source code.

Future work should continue to refine the schema in order to make it more concise and even easier to produce and manipulate. Although the Jikes compiler proved to be a fast and robust JavaML converter, it would be valuable to augment open IDEs, such as Eclipse [16], with JavaML parsing and generation capability. This would promote the usage of JavaML in a wider range of tools and will enable new applications of JavaML.

Referências

1. Greg J. Badros. JavaML: a markup language for Java source code. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):159–177, 2000.
2. Greg J. Badros. JavaML Home Page. http://javaml.sourceforge.net/.
3. Ademar Aguiar, Gabriel David, and Manuel Padilha. XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation. In Ernesto Pimentel, Nieves R. Brisaboa, and Jaime Gómez, editors, *JISBD*, pages 11–24, 2003.
4. Evan Mamas and Kostas Kontogiannis. Towards Portable Source Code Representations Using XML. In *Proceedings of WCRE'00, Brisbane Australia*, pages 172–182, November 2000.
5. Hrvoje Simic and Marko Topolnik. Prospects of encoding Java source code in XML. In *Proceedings of the ConTel 2003: 7th International Conference on Telecommunications, Zagreb, Croatia*, June 2003.
6. World Wide Web Consortium. XML Path Language (XPath) Version 1.0, November 1999. Available from http://www.w3.org/TR/1999/REC-xpath-19991116.
7. World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Data Model, November 2002. Available from http://www.w3.org/TR/2002/WD-query-datamodel-20021115.

8. World Wide Web Consortium. XSL Transformations (XSLT) Version 1.0, November 1999. Available from http://www.w3.org/TR/xslt.
9. Altova. XMLSPY integrated development environment. http://www.altova.com/.
10. Ademar Aguiar. JavaML 2.0 Home Page. http://www.fe.up.pt/~aaguiar/javaml/.
11. Sun Microsystems. Javadoc Tool Home Page. http://java.sun.com/j2se/javadoc/.
12. IBM. Jikes java compiler. http://www.ibm.com/developerWorks/oss/jikes/.
13. Michael H. Kay. SAXON Home Page. http://users.iclway.co.uk/mhkay/saxon/.
14. Michael L. Collard, Jonathan I. Maletic, and Andrian Marcus. Supporting Document and Data Views of Source Code. In *Proceedings of DocEng'02, McLean, Virginia USA*, November 2002.
15. Rudolf Ferenc, Susan Elliott Sim, Richard C. Holt, Rainer Koschke, and Tibor Gyimothy. Towards a standard schema for c/c++. In *Working Conference on Reverse Engineering*, pages 49–58, 2001.
16. Eclipse. Eclipse, an open and extensible integrated development environment, 2003. Available from http://www.eclipse.org.

Protótipo de um sistema para elaboração e manutenção de um manual da qualidade usando tecnologia XML e Docbook

Marco Rodrigues and Jenny Ferreira

\{mei03016|mei03003\}@fe.up.pt
Faculdade de Engenharia
Universidade do Porto

Resumo Os sistemas de gestão da qualidade estão, hoje em dia, cada vez mais presentes nas empresas. A necessidade de uma melhoria contínua de processos internos e serviços prestados, de forma a garantir aos clientes níveis adequados de qualidade, levou à criação de departamentos especializados neste domínio. O manual da qualidade é o documento base que orienta a implementação e manutenção do Sistema de Qualidade.
Neste artigo estão compilados os resultados e alguns detalhes funcionais da construção de um protótipo de um *sistema para elaboração e manutenção de um manual da qualidade*,usando, como base, as tecnologias XML e Docbook.
Palavras-chave: XML, Docbook, qualidade, manual, Web.

1 Introdução

O objectivo deste trabalho é o desenvolvimento de um protótipo de aplicação capaz de resolver a problemática da manutenção de manuais da qualidade, tendo em conta os seguintes requisitos:

- A aplicação deverá ser simples de usar e intuitiva, com mecanismos de edição directos e rápidos.
- Deverá ser usado o XML assegurando a compatibilidade com o vocabulário Docbook.
- A WEB deverá ser o ambiente escolhido.
- As regras de construção e edição de um manual da qualidade deverão ser respeitadas e implementadas.

2 Análise do Domínio

Foi levado a cabo um processo de pesquisa às normas da qualidade existentes assim como à análise de alguns manuais da qualidade de organizações. A amostra utilizada teve por base os manuais da qualidade de duas indústrias do ramo têxtil, uma do ramo automóvel e uma do ramo de serviços de redes e telecomunicações.

O objectivo foi conhecer a estrutura e conteúdos de um manual de qualidade. Como é óbvio, a actividade das empresas condicionou certos componentes do manual.

Concluiu-se que não existe uma estrutura pré-definida e normalizada sobre o manual da qualidade e outros documentos deste processo. No entanto, foi possível constatar que, na maioria dos aspectos, os manuais da qualidade seguem uma estrutura similar (por exemplo: Capítulos, Secções, Revisões, etc.), independentemente do sector de actividade.

Foi criada uma estrutura genérica e proprietária, na concepção do XML, com os elementos base necessários para gerar um qualquer manual da qualidade. Os manuais da qualidade analisados evidenciaram uma série de regras e restrições que, após validação com os seus autores, foram implementadas sob a forma de uma documento DTD (*Document Type Definition*. Essa estrutura está exemplificada nos diagramas presentes nas figuras 1, 2 e 3.

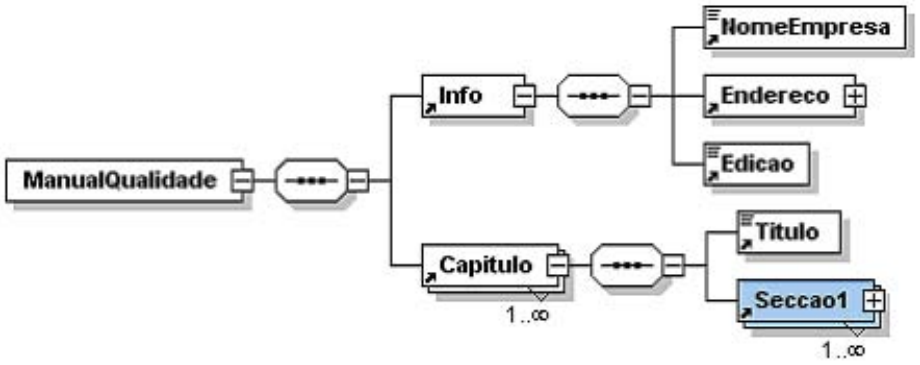


Figura 1. Diagrama DTD para o elemento ManualQualidade

3 Compatibilidade Docbook

Conforme descrito no ponto anterior, foi criado um documento XML, com sintaxe própria. A razão pela qual não se optou pelo uso directo da linguagem de anotação Docbook, é que esta não disponibiliza a totalidade dos elementos semanticamente associados aos manuais da qualidade. A sua adaptação poderia colocar em risco a inteligibilidade do documento XML. Saliente-se que, é essencial a compreensão do significado de cada elemento do ficheiro XML, já que a cada organização compete o desenvolvimento de uma máscara (ficheiro XSLT) de forma a dar o aspecto final que esta pretende.

No entanto, e para que a integração da informação do manual seja possível com organizações, por exemplo, no estrangeiro, desenvolveu-se um processo de

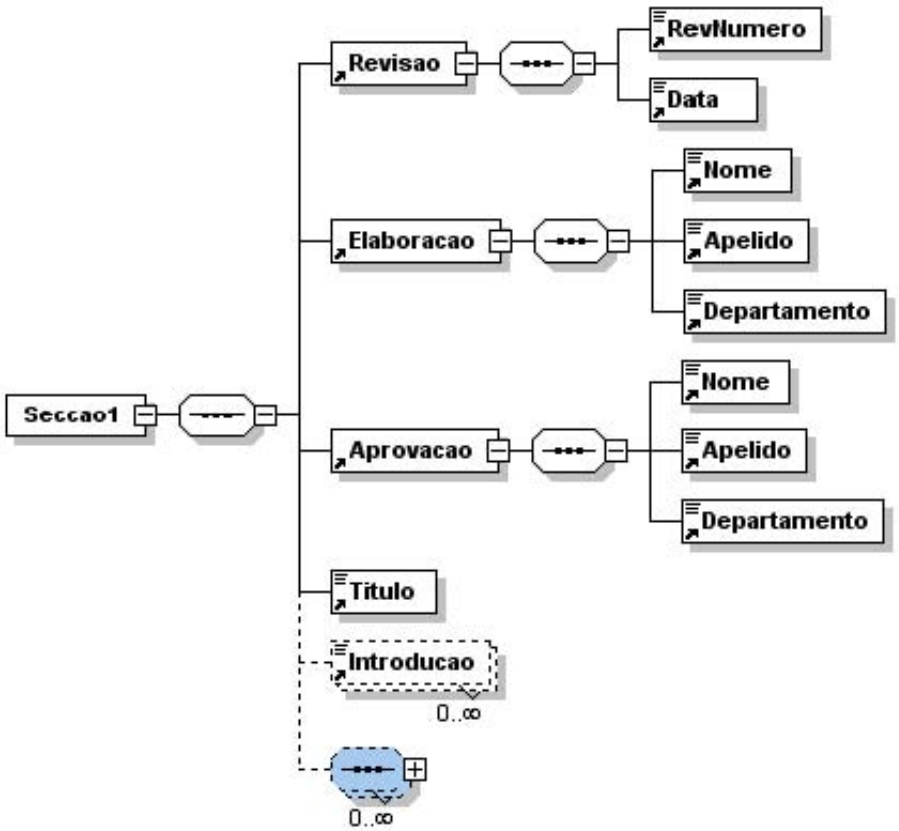


Figura 2. Diagrama DTD para o elemento Seccao1 (cont.)

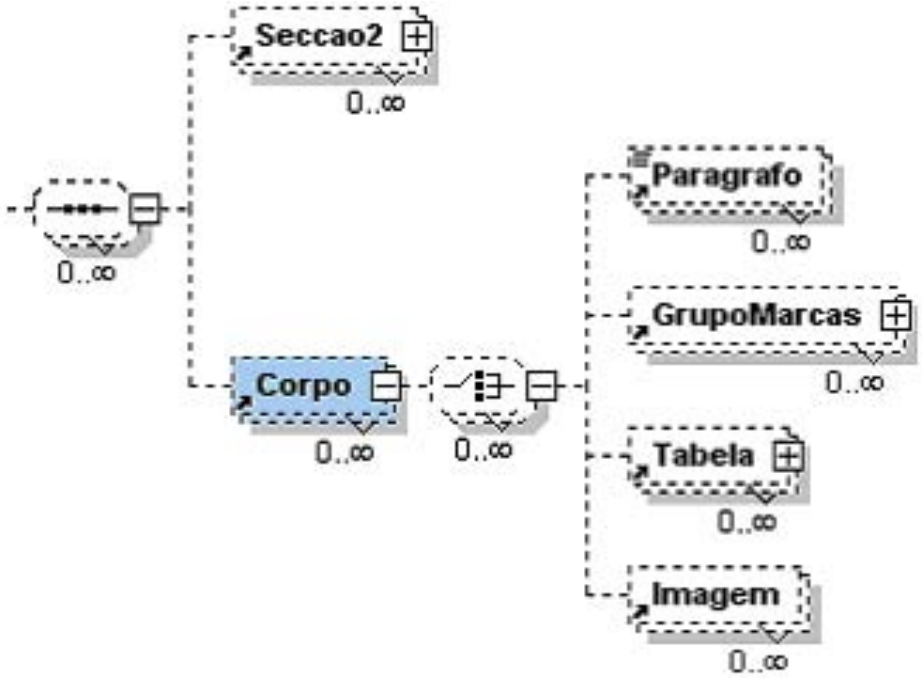


Figura 3. Diagrama DTD para o elemento Corpo (cont.)

conversão do ficheiro XML original em XML compatível Docbook. O processo de conversão foi conseguido através da criação de uma folha de estilo sob a forma de um ficheiro XSLT, conseguindo-se a transformação automática em XML com vocabulário Docbook.

O ficheiro XML gerado poderá ser utilizado para intercâmbio de dados, no caso de ambas as partes falarem Docbook.

A título de exemplo podemos verificar e analisar os seguintes excertos dos dois documentos xml.

```
<ManualQualidade id="root" LastId="15200">
   <Info id="x10">
           <NomeEmpresa id="x121">TÊXTEIS JF</NomeEmpresa>
           <Endereco id="x123">
                   <Rua1>Rua António Roberto Freitas, 12</Rua1>
                   <Rua2>Apartado 133</Rua2>
                   <Localidade>Porto</Localidade>
                   <CodigoPostal>4520-122</CodigoPostal>
                   <Pais>Portugal</Pais>
           </Endereco>
           <Edicao id="x122">2</Edicao>
   </Info>
   <Capitulo id="x1" Etiqueta="1">
           <Titulo id="x12">PROMULGAÇÃO</Titulo>
           <Seccao1 Estado="Activo" id="x13" Etiqueta="1">
                   <Revisao id="x14">
                           <RevNumero>3</RevNumero>
                           <Data>2000-12-12</Data>
                   </Revisao>
```

Tabela 1. Excerto do ficheiro original.xml

4 O processo de edição (authoring)

No que diz respeito à forma de edição de documentos XML, nomeadamente daquele criado no âmbito deste projecto, foram encontrados alguns programas capazes de o fazer. No entanto, o método de edição e sua usabilidade estão longe de tornar o processo simples e prático para o utilizador final.

Assim, optou-se pela criação de uma solução capaz de tornar possível a implementação de um projecto deste género.

Como requisitos principais, o protótipo a desenvolver deveria permitir a edição de toda informação contida no documento XML (com as restrições próprias de um manual da qualidade), a criação de novas revisões das secções principais do documento, mantendo as obsoletas inactivas, mas disponíveis para consulta, a

```
<book>
   <bookinfo>
           <orgname>TÊXTEIS JF</orgname>
           <address>
                   <street>Rua António Roberto Freitas, 12</street>
                   <street>Apartado 133</street>
                   <city>Porto</city>
                   <postcode>4520-122</postcode>
                   <country>Portugal</country>
           </address>
           <edition>2</edition>
   </bookinfo>
   <chapter label="1">
           <title>PROMULGAÇÃO</title>
           <sect1 label="1" condition="Activo">
                   <sect1info>
                           <revhistory>
                                   <revision>
                                           <revnumber>3</revnumber>
                                           <date>2000-12-12</date>
                                   </revision>
                           </revhistory>
```

Tabela 2. Excerto do ficheiro Docbook gerado

visualização dos elementos principais do documento e, finalmente, a visualização e impressão da versão final deste.

Não foi difícil chegar à conclusão que o ambiente web deveria ser o escolhido, de forma a aplicação ser o mais flexível possível, tendo por base uma plataforma global.

5 Tecnologias utilizadas

O sítio web foi desenvolvido utilizando o ambiente de desenvolvimento Microsoft Visual Studio .NET para a construção das páginas web e o Altova XMLSpy 2004 Enterprise Edition para o desenvolvimento dos documentos DTD e XSLT.

As linguagens utilizadas foram o HTML, VBScript e JavaScript e, em alguns casos, a framework do .NET, utilizando sempre o DOM para o acesso ao XML.

6 Estrutura e funcionalidades principais

A página inicial está dividida em duas partes, sendo que, a página da esquerda mantém a árvore dos elementos que compõem o manual, e a da direita permite a edição e visualização da informação contida no elemento seleccionado.

Conforme se pode verificar pela figura 4, a disponibilização da estrutura do documento em forma de árvore, permite um acesso muito rápido à informação

desejada, mantendo, a todo o momento, o utilizador informado do local em que se encontra relativamente ao documento.

Foi mantida, por motivos de consistência e de usabilidade, a mesma estrutura em grande parte da solução, já que a maior parte das funcionalidades está acessível sem que o utilizador tenha que mudar de ecrã.

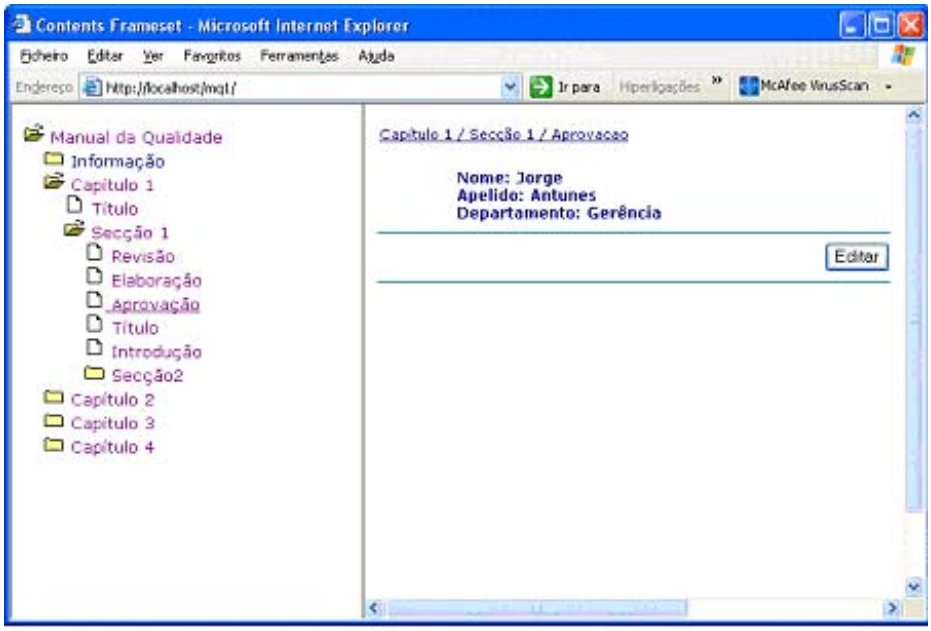


Figura 4. Imagem da página inicial do sítio

As opções de edição são mostradas de acordo com o tipo de elemento seleccionado e é permitido, entre outras:

- alterar a informação relativa ao elemento;
- efectuar uma pré-visualização;
- eliminar o elemento;
- acrescentar novos elementos ao nível daquele seleccionado ou de nível inferior.

Em casos especiais, como o da secção de nível 1, é permitida a criação de uma nova revisão, mantendo a original como obsoleta.

Sempre que o utilizador efectua uma alteração ao documento, este é validado de acordo com as regras do manual da qualidade, explícitas no ficheiro MQ.DTD. O uso desta validação constante, mantém a consistência do ficheiro que contém a informação e sustenta a compatibilidade com Docbook.

7 Conclusão

No final do desenvolvimento deste protótipo, verificou-se que as tecnologias XML e associadas, permitem, o manuseamento flexível da informação, num formato standard. O contínuo desenvolvimento de linguagens XML denuncia a crescente adesão a esta tecnologia por cada vez mais empresas, a nível global.

O protótipo construído pretende apenas demonstrar as capacidades desta linguagem e das implementações desta sob a forma de APIs disponíveis para muitos ambientes de desenvolvimento.

As folhas de estilo XSLT permitem o controlo absoluto sobre os elementos XML através das expressões XPath, utilizadas neste projecto.

Desta forma, concluímos referindo o enorme prazer no desenvolvimento e utilização deste conjunto de ferramentas, que serão certamente o futuro na integração global da informação.

Bibliografia

- BRADLEY, N., The XML Companion, Addison Wesley Publishing Company, 1998
- HAROLD, E.R., XML Bible (2nd Edition), John Wiley & Sons, 2001
- RAMALHO, J. C., HENRIQUES, P., XML & XSL, FCA - Editora de Informática, Lda., 2002

Referências

[1] Quality Manual, `http://www.quality-manual.com`;
[2] Guidance on the Documentation Requirements of ISO 9001:2000 `http://isotc176sc2.elysium-ltd.net/Documentation.doc`, `http://www.apcer.pt`
[3] W3C — World Wide Web Consortium, `http://www.w3.org`
[4] DocBook XML, `http://www.oasis-open.org/docbook/xml/`
[5] The DocBook Wiki, `http://www.docbook.org/wiki/moin.cgi/`
[6] XML DOM Reference TopXML, `http://www.topxml.com/xml\_dom/`

Música e XML

David Freitas and Jorge Amaral

Faculdade de Engenharia da Universidade do Porto

Resumo A música é indispensável na cultura humana. Cada vez mais é necessário criar um formato universal que a represente e reproduza com qualidade. Descrevem-se vários formatos musicais que tentam solucionar este problema. É dado ênfase ao MusicXML que começa a ser um standard nesta área. Apresentamos aplicações desenvolvidas pelos autores aproximando o MusicXML destes objectivos. Avaliamos este formato de representação e reprodução musical e apresentamos usos futuros e caminhos para a universalidade da aplicação do XML à música.

1 Introdução

A representação de notação musical em formato electrónico não é um fenómeno recente. O MIDI, muito provavelmente o formato musical com maior popularidade, já existe há mais de 30 anos. Estranhamente, para um formato com tanto tempo de existência, não existia até há muito pouco tempo, nenhuma ferramenta que permitisse a sua representação gráfica, reprodução sonora e utilização na Web. Algumas tentativas de conseguir estes objectivos foram feitas. Poucas tiveram sucesso. Áreas como a música em formato áudio ou livros em formato electrónico, estão a ser muito explorados enquanto que a publicação de notação musical através da Internet representa um potencial inexplorado. A maioria da música publicada em formato digital está representada em PDF (Portable Document Format), que não acrescenta qualquer informação semântica à música representada. Este artigo pretende analisar o que o XML trouxe de novo a esta área, se este é o bom caminho e, se sim, o que pode ser melhorado. Pretende-se ainda deixar algumas ideias de novas funcionalidades que, com o MusicXML, podem ser mais facilmente implementadas. Durante este estudo utilizamos como base o MusicXML, formato que começa a ser um Standard entre as principais aplicações musicais.

2 Música e sua representação electrónica

A representação e reprodução de música é uma operação complexa. Nesta secção vamos ver alguns formatos que se propuseram resolver estes problemas: O NIFF (Notation Interchange File Format) e o SMDL (Standard Music Description Language) e o MIDI (Musical Instrument Digital Interface). Este último o formato com maior sucesso.

2.1 NIFF (Notation Interchange File Format)

O NIFF (Notation Interchange File Format) representa música de uma forma gráfica. Não existe o conceito de nota. Todos os elementos: notas, acidentes, tempo, etc. são representados pela sua posição na pauta e/ou pelo seu grafismo. Esse formato é excelente quando a fonte de informação é o scanner e o único objectivo é a representação gráfica. A sua utilização fora das aplicações de digitalização é muito restrita e de uma forma geral mal sucedida. Operações de análise e de reprodução sonora são, para além de extremamente complexas, pouco eficientes e passam, geralmente, por uma representação complementar. Existe um projecto para associar o NIFF ao XML ([3]).

2.2 SMDL (Standard Music Description Language)

Em 1984, Charles Goldfarb, o inventor do SGML, propôs um projecto ANSI para criar um standard de representação musical em SGML: o SMDL (Standard Music Description Language). Esta foi uma tentativa para criar uma especificação formal para representação musical baseada no Standard Generalized Markup Language (SGML). Foi concebido sem guias de problemas de implementação. Tentou resolver todos os problemas da representação e reprodução do passado, presente e futuro. Obviamente, resultou numa representação extremamente complexa, de tal forma que se diz que esta não é percebida por ninguém. Parece não ter tido quaisquer resultados a nível comercial. Para saber mais sobre a história do SMDL pode consultar ([4]).

2.3 MIDI (Musical Instrument Digital Interface)

O MIDI - Musical Instrument Digital Interface, é um protocolo de comunicação entre instrumentos musicais electrónicos inventado em 1981 por Dave Smith (co-fundador da empresa Sequential Circuits), que pretendia servir de standard de comunicação, para que as diferentes marcas de instrumentos da época não utilizassem formatos proprietários, implicando que um músico não pudesse trabalhar em simultâneo com instrumentos de diferentes fabricantes. O MIDI é sem dúvidas o único formato que obteve sucesso para representação e reprodução musical em formato electrónico. Neste formato a informação é descrita através de eventos. Cada evento é descrito pelo início (evento on), fim (evento off), e o tempo em que ocorre (definido em unidades de tempo - *ticks*). Um exemplo pode ser visto na Figura 2.3. No entanto, o modo como a informação musical é representada não é apropriada para muitas aplicações, como por exemplo, aplicações que realizam análise musical. O formato MIDI normalmente não fornece informações sobre tonalidade, marcação (4/4 ou 2/4) e não distingue notas iguais com nomes diferentes (C# e Db). O MIDI também não é apropriado para geração de partituras pois, além das ambiguidades já citadas, a informação rítmica contida num arquivo MIDI é insuficiente para extrair a notação adequada. Pode consultar mais informações em ([2]).

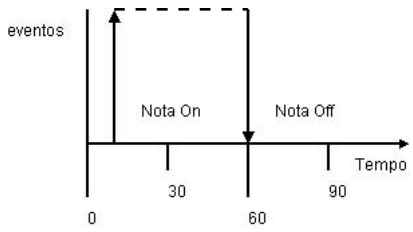


Figura 1. Funcionamento básico do MIDI.

2.4 MusicXML

Actualmente, o MusicXML está disponível através de uma licença livre baseada no modelo da W3C e é suportado, entre outros, pelos seguintes programas comercias: Finale, SharpEye Music Reader e o Dolet. Existem ainda alguns projectos open source como o XEMO e o KGuitar (pode obter mais informações em [1]). Na realidade, a adopção do MusicXML é a mais rápida desde o MIDI. Só se pode esperar ainda maiores e melhores desenvolvimentos no futuro. O MusicXML pretende funcionar como um tradutor universal para as notações musicais mais comuns, suportando ainda aplicações de análise, recolha de informação e execução. Existe uma separação dos elementos sonoros dos de notação, permitindo que um arquivo represente uma informação sonora fiel e que a sua representação e execução seja menos limitada. Na Figura 2.4 a duração e o tipo de representação, apesar de possuírem uma dependência semântica, são definidos separadamente e independentemente.

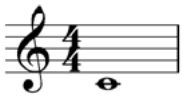


Figura 2. Exemplo mínimo de uma pauta.

Algumas Aplicações Já são muitas as aplicações que utilizam MusicXML. De seguida descrevemos algumas das funcionalidades já implementadas assim como algumas ideias de possíveis novas aplicações.

Migração de música entre software Devido ao grande número de formatos proprietários existentes, o simples facto de ser possível, através do MusicXML, permitir a tradução entre estes, é suficiente para causar grande impacto na comunidade de músicos internacional.

Tradução universal entre métodos de notação musical ocidental Obter notação musical a partir de um ficheiro MusicXML é também uma funcionalidade importante, uma vez que qualquer notação musical pode ser convertida para MusicXML. Podemos, assim, ter um tradutor de notação universal baseado em MusicXML, sem que se perca informação no processo como acontece com software especializado de notação.

Publicação de música em formato não proprietário A publicação na Internet, por exemplo, de ficheiros MusicXML, pode resolver os problemas de quem necessita de um formato que permita não só execução, como também visualização de uma música. Actualmente, o mais comum é recorrer a serviços de download de ficheiros MIDI e utilizar um conversor MIDI para pauta ou tabulatura com resultados muito imperfeitos. O MusicXML garante a correcta representação visual da música, aliada a uma também correcta execução.

Análise O MusicXML tira partido de ser um documento XML. Pode utilizar o XQuery, XML Document Object Model (DOM) ou Simple API for XML Parsing (SAX) ou ainda a XML Path Language (XPath) para mais facilmente analisar composições musicais. Deste modo é mais fácil recolher informações de estilo ou fraseamento.

Execução A incorporação do MIDI no MusicXML permite a comunicação directa com literalmente todos os sistemas electrónicos de execução musical dos últimos 20 anos. Paralelamente, a execução humana está também garantida através da visualização de pautas e tabulaturas.

3 Aplicações

A maioria das aplicações que suportam MusicXML fazem-no apenas para exportação de dados. Com o nosso trabalho pretendemos mostrar que é possível reproduzir e representar pautas directamente a partir do MusicXML.

Foram desenvolvidas três aplicações com funcionalidades distintas baseadas em MusicXML. Uma aplicação gráfica, construída em Java, que permite visualizar uma pauta, gerar e reproduzir MIDI. As outras duas aplicações permitem converter MusicXML em XHTML e SVG para poderem ser visualizadas directamente num browser. As aplicações tiveram como objectivo mostrar as potencialidades do MusicXML e não a representação completa da pauta. Estas aplicações são um ponto de partida para uma aplicação mais complexa.

3.1 Aplicação Gráfica

A aplicação gráfica utiliza a API Java - SAX. A sua utilização facilitou imenso o desenvolvimento da aplicação. Esta escolhabaseou-se no facto do SAX ser baseados em eventos que era exactamente o que queríamos como explicamos a seguir. A solução utilizada para enquadrar os vários elementos e seus atributos

foi criar um conjunto de variáveis globais (uma por cada elemento) às quais são atribuídos os valores verdadeiro ou falso sempre que se inicia ou finaliza o processamento de determinado elemento. Desta forma é extremamente simples saber em que contexto se encontra o elemento. Um exemplo pode ser encontrado na Figura 3.1. As notas são desenhadas dinamicamente permitindo assim operações como zoom, mudanças do seu aspecto em tempo-real como por exemplo mudar de cor quando a nota estiver a tocar (ainda não implementado) ou ainda a possível geração de SVG. A reprodução e sua gravação num ficheiro MIDI foi possível através da API: Java Sound API ([5]). Esta aplicação focou-se especialmente nas pautas para bateria que introduzem diferente notações. Um exemplo da aplicação pode ser visto na Figura 3.1.

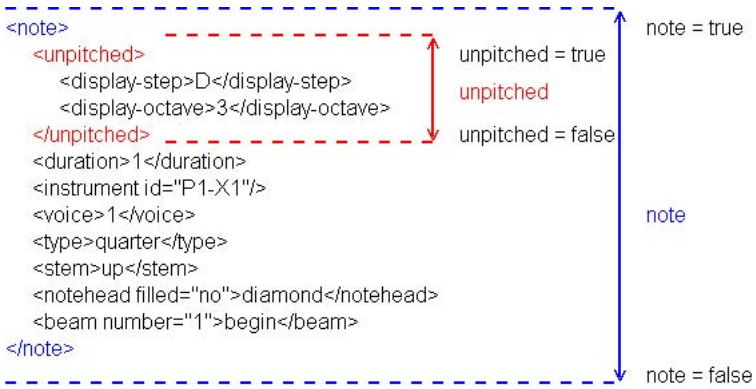


Figura 3. Criação do contexto de cada elemento.

3.2 MusicXML2SVG

A aplicação MusicXML2SVG pretende ser uma aplicação simples de linha de comando que permita, a partir de um ficheiro MusicXML obter um ficheiro SVG com a visualização da pauta correspondente a este. Assim, o ficheiro SVG de saída apresenta:

- As linhas de pauta necessárias para suportar as notas a representar;
- Claves;
- Tempos;
- Divisão de compassos;
- Notas desde a semibreve à semifusa;
- Pontos de aumento;
- Pausas correspondentes às notas possíveis;
- Apresentação do nome da peça e autores.

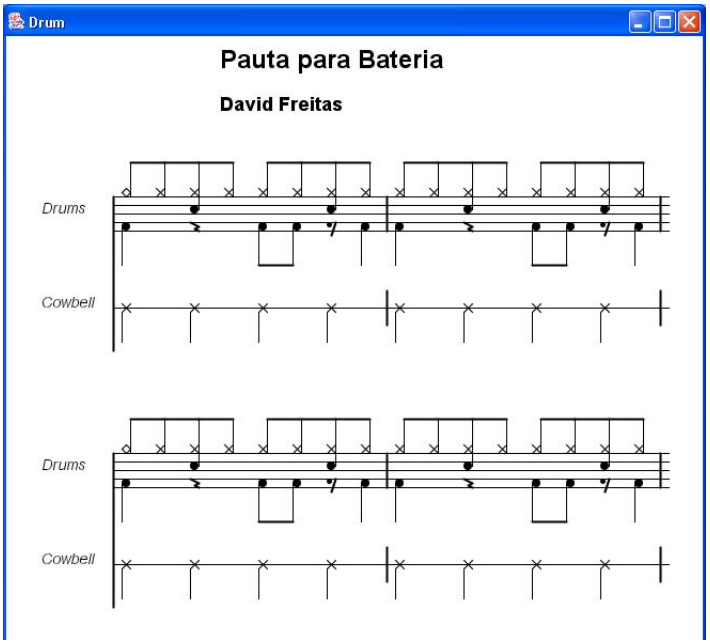


Figura 4. Aplicação gráfica para apresentação de pautas em MusicXML.

Esta aplicação foi desenvolvida em Java, usando SAX para ler o ficheiro XML e fazer a construção do ficheiro SVG a partir deste. Trata-se de uma aplicação de consola que recebe o nome do ficheiro XML e o nome do ficheiro de saída desejado.

3.3 MusicXML2HTML

A aplicação MusicXML2XHTML pretende ser uma aplicação simples de linha de comando que permita, a partir de um ficheiro MusicXML obter-se um ficheiro XHTML e um ficheiro CSS que permita a visualização da pauta correspondente a este. Esta aplicação foi desenvolvida em Java, usando SAX para ler o ficheiro XML, fazer a construção dos ficheiros XHTML e CSS a partir deste. Trata-se de uma aplicação de consola que recebe o nome do ficheiro XML e o nome base para os ficheiros de saída desejados. Um exemplo da aplicação pode ser visto na Figura 3.3. Os resultados desta aplicação são os mesmos que os da aplicação anterior.

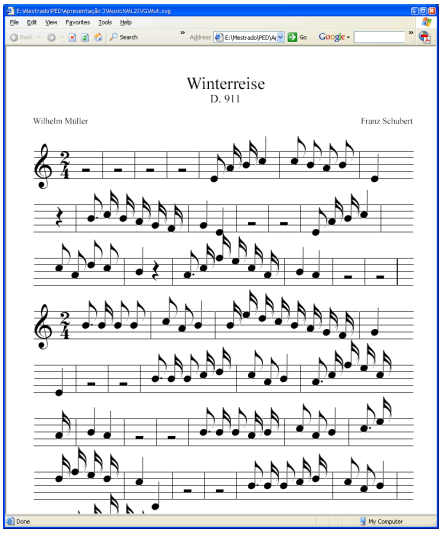


Figura 5. Resultado gerado pela aplicação MusicXML2SVG e MusicXML2HTML.

4 Próximos Desenvolvimentos

O MusicXML permite não só melhorar o que já existe, mas também permite, muito facilmente, oferecer novos serviços. Descrevemos duas novas aplicações que poderiam ser úteis, que pensamos ainda não existir.

4.1 Pesquisa Semântica Musical

Um músico (possivelmente amador) deseja comprar um CD que contenha uma música da qual apenas de lembra da melodia. O nome do artista, música ou álbum, que hoje em dia são essenciais para comprar o disco não são conhecidos. Ele pode transcrever a melodia utilizando uma aplicação que gere MusicXML através de um teclado, guitarra, voz, ... e que submeta o documento XML, possivelmente a uma empresa, que oferece os seus serviços de consulta à sua base de dados em XML devolvendo-lhe, através da análise da música, o nome do(s) artista(s), música(s) e disco(s), assim como parte(s) da(s) música(s) que interpretam ou da(s) que mais se parece(m) com a pretendida. Eventualmente poderá através de um serviço web (implementado com WebServices) permitir-lhe a compra on-line do(s) disco(s).

4.2 Sugestão Real de Música

Uma loja especializada em vendas on-line permite aos seus utilizadores criar um perfil dos seus gostos musicais. Até aqui a sugestão de discos que possam

interessar aos cliente é feita pelo estudo dos artistas que gosta, estilo do artista, grupo de utilizadores que têm gostos parecidos, etc. Nunca é utilizada a própria semântica musical: o ritmo da música, os instrumentos, acordes mais ou menos complexos, etc. Com o MusicXML é possível estudar melhor a música que o utilizador gosta.

5 Conclusão

Existem alguns pontos do MusicXML que devem ser melhorados e outros que devem ser estudados. Por exemplo, é possível representar pautas incorrectamente no MusicXML. Uma outra preocupação é a complexidade do MusicXML. Esta limita muito a possibilidade de escrita directa de MusicXML por um músico. Como consequência o MusicXML será utilizado apenas como formato de armazenamento e transferência de músicas. A possibilidade de criação musical directamente em XML é um possível futuro objectivo que permitiria não recorrer a software de notação profissional para a criação de MusicXML. A definição de um formato reduzido do MusicXML ou de uma nova linguagem para composição musical são duas possibilidades a estudar.

O MusicXML pode ser considerado um sucesso no principal objectivo dos seus criadores: a criação de uma linguagem que sirva de tradutor universal entre aplicações comerciais de notação musical ocidental. Este objectivo parece ser, no entanto, apenas uma pequena parte do que o XML pode fazer pela música. Se, por exemplo, recolhermos e armazenarmos composições musicais tradicionais japonesas, por exemplo, até que ponto o MusicXML pode ser útil? Infelizmente o MusicXML não permite armazenar música não ocidental. De uma forma mais geral, não permite armazenar música que não esteja conforme o sistema ocidental de 12 notas por oitava. As microtonalidades inerentes a muitos tipos de música não são possíveis de representar. A estreita ligação que existe entre o MusicXML, a pauta ocidental e o MIDI, impossibilita a utilização universal do MusicXML. O futuro deverá passar por uma linguagem que permita a descrição de toda a música, de uma ferramenta realmente universal para o armazenamento e estudo da música.

Apesar disto, estamos no caminho certo para a representação de pautas em formato electrónico. Da mesma forma que o MIDI resolveu o problema da compatibilidade e do transporte de música entre instrumentos musicais no início dos anos 80, altura que não conseguiam que instrumentos de companhias diferentes tocassem em uníssono e levou ao aumento das vendas de instrumentos musicais, o XML (seja com o MusicXML ou outro Standard) pode fazer proliferar a distribuição e negócio de pautas musicais em formato electrónico. Novos serviços, como os que referimos, podem ser implementados mais facilmente e os que existem podem ser melhorados.

Referências

1. Recodare LLC: http://www.musicxml.com/

2. MIDI Manufacturer's Association: http://www.midi.org
3. http://niffty.sourceforge.net/xml/
4. http://www.sgmlsource.com/history/hthist.htm
5. API javax.sound: http://babbage.clarku.edu/java/docs/api/javax/sound/midi/package-summary.html

Parte VII

XML e Tecnologias

TX — validação de XML baseada em tipos dinâmicos

José João Almeida and Alberto Manuel Simões

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

Resumo Desde o advento do SGML e posteriormente do XML, que a validação de documentos tem sido focada.
Esta validação surgiu para analisar a estrutura dos documentos SGML e XML usando DTDs. Além dessa, e devido às restrições do XML em relação ao SGML, a validação de XML bem formado também tem sido usada. Mais recentemente, os Schema e Schematron vieram permitir a validação a um nível superior: não só a estrutura do documento mas também alguma validação de conteúdo.
Neste artigo apresentamos a ferramenta TX que visa outro nível de validação, em que os tipos possam ser mais ricos e/ou calculados dinamicamente, e onde se possa definir funções de anotação e/ou correcção das porções do documento que não sigam as especificações.

1 Introdução

A definição de estrutura de um documento XML e a sua validação estrutural em relação a um DTD é, já desde os seus primórdios, uma tarefa que tem sido tratada com relativo sucesso.

A validação semântica tem vindo a ser uma preocupação crescente mas de difícil solução. A tipagem de elementos XML tem sido ignorada ou, recentemente com Schemas[3], Schematrons[5] e XCSL[7], demasiado rígidas em relação aos tipos usados.

Se considerarmos um XML que esteja a funcionar como base de dados (contendo informação fortemente estruturada), poderemos usar validadores de conteúdo baseadas em tipos como *strings* ou *inteiros* ou mesmo *inteiros menores do que 5 e maiores do que -5*. No entanto, a validação de conteúdo impõe frequentemente questões mais complexas.

Por exemplo, numa memória de tradução em formato TMX[8] (baseado em XML), uma tarefa da validação deverá, sem dúvida, verificar se o conteúdo das etiquetas de cada uma das memórias de tradução está na língua correcta, e sem erros ortográficos.

Esta foi a motivação para a construção da ferramenta a que chamamos TX (Typed XML) — uma *framework* de construção de validadores (usável como biblioteca Perl), incluindo um comando Unix para validação de documentos XML usando especificações TX. O use do TX é, por vezes, pouco genérica, ganhando-se em expressividade, nomeadamente com a possibilidade de validação com semântica operacional, tipos dinâmicos e anotação/correcção dos documentos.

1.1 Validação de Semântica Operacional

Consideremos os endereços de Internet: embora exista uma definição rígida para URIs, e embora seja possível definir uma expressão regular que valide essa sintaxe, isso não nos garante que se trate de um URI activo — todos sabemos que existem URIs totalmente válidos sintacticamente mas que não pertencem ao conjunto dos URIs vivos.

Frequentemente queremos que o nosso critério de validação verifique se os URLs estão activos, e não só se estão correctos. Um exemplo é a validação de um ficheiro de *bookmarks*.

Consideremos a correcção ortográfica de que já falamos: esta pode ser realizada por um programa externo como o ispell[4], aspell ou jspell, que suportam XML (e portanto, ignorará as etiquetas). No entanto, esta abordagem não é a mais correcta para a correcção de qualquer documento — já que analisa todo o documento independentemente do contexto.

Veja-se, por exemplo, o caso das memórias de tradução[6], em que o inicio do documento contém meta-informação e o corpo é constituído por frases em várias línguas.

Obrigaria ao utilizador duas passagens do ispell sobre o texto, uma em cada uma das línguas, tendo o cuidado de ignorar em cada passagem metade do texto.

Não sendo esta uma solução viável, torna-se importante existir a possibilidade de dar tipos a porções de documento com línguas diferentes, em que a correcção seleccione automaticamente a língua a usar.

1.2 Validação com Tipos Dinâmicos

A validação com tipos dinâmicos tem o objectivo de calcular o tipo do elemento e portanto o seu critério de correcção, com base no próprio elemento.

Por exemplo, é impossível tipar genericamente qualquer memória de tradução, já que as línguas usadas não são necessariamente as mesmas em cada uma[1]. Desta forma, é necessário que o validador consiga discernir em que língua está o elemento antes de o avaliar. Este tipo de informação pode ser obtida usando, por exemplo, em função do atributo `xml:lang` existente nas memórias de tradução.

1.3 Anotação e Correcção

Supondo que já temos um sistema para validar este tipo de informação, o passo seguinte será discutir a forma de reacção aos erros encontrados. A forma habitual é indicar que na linha x, o elemento y tem texto inválido. Embora útil, esta informação não é suficientemente precisa. Mesmo que a mensagem de erro inclua, por exemplo, a palavra com erro ortográfico, quando o utilizador tentar editar

[1] Uma memória de tradução tem frases em duas ou mais línguas, pelo que nada obsta a que se tenha uma memória de tradução que use apenas Português e Inglês, e uma outra que use apenas Francês e Dinamarquês

o XML para o corrigir irá, com certeza, ter-se esquecido da palavra em causa. Para colmatar este tipo de problema, propomos a marcação do texto analisado para facilitar a sua posterior correcção (eventualmente usando uma ferramenta específica para esse efeito).

Parece-nos igualmente importante que um validador seja capaz de não só validar mas também corrigir (ou propor correcções). Sem dúvida que grande parte dos erros que um validador encontra não podem ser corrigidos sem intervenção humana mas, isso não implica que o sistema não seja capaz de assistir, interactivamente, no processo de correcção dos erros encontrados (como é habitual, por exemplo, nas ferramentas de correcção ortográfica.

2 Abordagem Proposta

Na abordagem que aqui se propõe, pretendemos enriquecer o processo de validação de documentos XML através de:

- *associar* (estática ou dinamicamente) *tipos* a alguns dos elementos;
- *criar* (externamente) *validadores de tipos* que sejam capazes de marcar e alterar (automaticamente ou de modo interactivo) os elementos incorrectos desse tipo.
- mecanismos de validar tipos baseados em *semântica operacional*: ou seja que os tipos possam ficar associados a uma função (no caso presente imperativa, Perl) que tire partido da funcionalidade existente nas bibliotecas e no sistema operativo.
- criar mecanismos (uma API, um módulo Perl) de *definir novos tipos* e até validadores.

Na definição dos tipos pode haver necessidade de lhes associar:

- uma *função de marcação* de erros encontrados — esta função pode ser tão simples como: para a correcção ortográfica, preceder cada palavra desconhecida por um símbolo; pela inclusão de um comentário; pela adição de etiquetas à volta do elemento ou conteúdo errado;
- uma *função de correcção* automática ou interactiva das anomalias encontradas;
- uma *política de inclusão ou não dos filhos* (recursividade ou não pelas subárvores dos elementos filhos).

A ferramenta utiliza valores por omissão no caso destas propriedades não estarem definidas.

3 TX *by example*

Considere-se um dicionário definido em XML com cabeçalho e entradas, de acordo com a seguinte gramática (DTD):

```
<!ELEMENT dic       (cabecalho, entrada*)      >
<!ELEMENT cabecalho (...)                      >
<!ELEMENT entrada   (termo*, definicao, figura)>

<!ELEMENT termo     (#PCDATA)                  >
    <!ATTLIST termo xml:lang CDATA #REQUIRED   >

<!ELEMENT definicao (#PCDATA)                  >

<!ELEMENT figura    (#EMPTY)                   >
    <!ATTLIST figura url CDATA #REQUIRED       >
```

e, a titulo exemplificativo, considere-se o seguinte documento:

```
<dic>
  <cabecalho>...</cabecalho>
  <entrada>
    <termo xml:lang="pt">violino</termo>
    <termo xml:lang="en">violin</termo>
    <definicao>Instrumento de cordas...</definicao>
    <figura url="http://imagem.de/violino.png"/>
  </entrada>
  <entrada>
  ...
  </entrada>
</dic>
```

A validação da estrutura deste documento pode ser feita usando um validador comum sensível a DTDs (por exemplo, `xmllint`[9]). No entanto, o uso deste tipo de ferramenta nos garante que:

- cada *termo* está na língua definida, e sem erros ortográficos;
- a *definição* está em Português (para manter coerência em todo o dicionário);
- a *imagem* existe, e está acessível publicamente;

De acordo com a abordagem TX, o que pretendemos é associar a cada elemento um tipo. Para especificar o elemento, a forma mais expedita é o uso de XPath[2]. A cada um destes elementos queremos associar um tipo estático ou um tipo dinâmico:

$$TXdef \equiv XPath \rightharpoonup (Tipo + TipoDinamico(elemento))$$

em que `TipoDinamico` é uma função que, dado o elemento ou atributo XML a analisar nos devolve o tipo desse elemento.

$$TipoDinamico : elemento \longrightarrow Tipo$$

Um exemplo de associação de tipos ao DTD apresentado anteriormente, usando a sintaxe TX, seria:

```
1  //definicao      Text("PT")
2  //termo          Text(@xml:lang)
3  //figura@url     ActiveURL
```

Esta sintaxe é definida por duas colunas; um selector em XPath do elemento ou atributo que queremos validar, e a definição do tipo estático ou dinâmico a ser usado:

1. a primeira linha apresenta um tipo estático, que valida a língua do conteúdo do elemento definição, como sendo Português;
2. a linha seguinte, é semelhante à anterior, mas de ordem superior: a língua a ser validada é determinada a partir do atributo `xml:lang` desse elemento (usando uma directiva XPath);
3. o último exemplo é uma regra directa sobre um atributo. O tipo estático ActiveURL valida se determinado URL ainda está vivo.

Embora o tipo `ActiveURL` seja *built-in*, na secção 3.1 usá-mo-lo para demonstrar como se pode adicionar a esta lista de associações a definição de novos tipos.

O modo de utilização da ferramenta TX sobre a especificação completa seria:

```
1  tx file.txd file.xml
```

em que `file.txd` é a nossa definição de tipos e `file.xml` o ficheiro a validar. Por omissão o `tx` marca os erros encontrados. Se usada a opção `-correct` a ferramenta invoca a função de correcção associada a cada tipo (caso não exista usará a de marcação).

3.1 Definição de validadores para novos tipos

No exemplo que seguidamente se apresenta é criado um um novo tipo (ActiveURL), usado no exemplo anterior.

```
1  //url ActiveURL

2  %%
3  use LWP::Simple;

4  XML::TX::addType(
5     ActiveURL =>
6         { markit => sub{ $c = markAsErr($c) unless (LWP::Simple::head($c));
7                          toxml()}, } );
```

Esta especificação é composta por duas partes:

1. uma zona de associação Xpath → tipo (linha 1)
2. uma zona de criação de novos tipos: através de código Perl, invocando funcionalidade do módulo XML::TX para adicionar o tipo definido.

A função **addType** usada para definir o novo tipo urlActive, está a receber uma função (`markit`) para marcar as situações consideradas erradas. Essa função de marcação dos erros está a usar o módulo Perl `LWP::Simple` [1] para descarregar o cabeçalho (função `head`) ligado ao URL a validar.

Saliente-se que este tipo de validação é difícil de obter usando apenas especificações declarativas.

3.2 Validadores usando extrai-processa-reconstroi

Por uma questão de eficiência, não é aceitável arrancar um processo externo de validação para cada ocorrência de um determinado elemento[2].

Por exemplo um corrector ortográfico normalmente carrega para memória o dicionário associado, ou seja, carrega para memória alguns MBytes de informação. Se esse carregamento for feito para cada parágrafo encontrado, a validação vai ser extremamente lenta.

Este facto pode ser ultrapassado com a seguinte estratégia:

1. Criar um texto com os elementos de cada tipo (extrai)
2. executar interactivamente o validador sobre cada ficheiro de cada tipo criado (processa)
3. substituir no texto XML original os elementos que tiverem sido alteradas no processo de validação (reconstrói)

Esta estratégia de validação é genérica e o módulo XML::TX dispõe de função de ordem superior (`ext_proc_rec`) que pode ser usada na definição de novos tipos.

```
Correcção=ext_proc_rec(CorrectorInteractivo,....)
```

3.3 Validadores usando apenas o módulo XML::TX

O módulo XML::TX, pode ser usado para criar validadores completos totalmente escritos em Perl.

No exemplo que seguidamente se apresenta é criado um validador.

```
use XML::TX;
my $types={ sentencePt => text("pt"),
            sentenceEn => text("en"),
            definition => sub{text($v{'xml:lang'} || "pt")},
            url        => "urlActive",
          };

XML::TX::addType(
   urlActive =>
       { markit => sub{ $c = markAsErr($c) unless (LWP::Simple::head($c));
```

[2] O esforço de arranque de uma aplicação externa pode ser grande.

```
10                          toxml()}, } );

11  XML::TX::markit($filename,$types);
```

O validador criado usa os tipos predefinidos:

- os elementos *sentencePt* são do tipo predefinido *text*("*pt*")
- os elementos *sentencePt* são do tipo predefinido *text*("*en*"),
- os elementos *definition* são de um tipo dinâmico calculado em função do atributo *xml:lang*,
- os elementos *url* são de um tipo *urlActive* que é definido através da invocação da função *addType*.

4 Tipos predefinidos

Na ferramenta TX existem alguns tipos predefinidos que resultaram de necessidades encontradas em problemas reais.

Tipo ActiveURL

O tipo ActiveURL foi já apresentado em secções anteriores.

Este tipo de validade é por vezes mais importante do que a simples validação sintáctica já que permite a detecção de gralhas que tipicamente mantêm o URL sintacticamente correcto.

Tipo Email

Para os e-mails não podemos, como para os URLs, validá-los de forma sistemática, podendo apenas definir os sintacticamente válidos ou inválidos.

Tipo Data

A validação de datas também é simples, sendo no entanto complicado validar todas as variantes possíveis para a escrita de uma data. Noutros casos, existe a noção de formato válido, e todos os outros são considerados errados.

Neste ponto ganha-se bastante em ter a possibilidade de correcção. Sabendo-se o conjunto de formas diferentes usadas para escrever datas, será possível, sem grande esforço, fazer a conversão para um formato de data *standard* no documento.

Tipo Texto-$\mathcal{L}_\alpha$

No processamento de textos de uma língua $\mathcal{L}_\alpha$ há necessidade de fazer a respectiva análise/correcção ortográfica.

A marcação de erros está a ser feita do seguinte modo:

1. enviar cada palavra a um processo Ispell[4] ou Aspell;
2. prefixar as palavras desconhecidas com uma marca (`##`)

A função de correcção está a usar a estratégia extrai-processa-reconstrói, atrás referida:

1. Criar um texto com as frases de cada língua (extrai)
2. executar interactivamente um corrector ortográfico sobre cada ficheiro de língua criado (processa)
3. substituir no texto XML original as frases que tiverem sido alteradas no processo de correcção ortográfica (reconstrói)

5 Conclusões

A tipagem dos métodos tradicionais é insuficiente para manter documentos com semântica forte, pelo que a possibilidade de validação com base em tipos operacionais e dinâmicos é importante.

No entanto, esta ferramenta não pode ser vista como uma substituição aos métodos tradicionais, mas antes mas um passo na validação de um documento XML:

- correcção sintáctica;
- correcção estrutural;
- correcção de tipos básicos;
- correcção de tipos dinâmicos e operacionais;

A possibilidade de programação da ferramenta obriga a maiores conhecimentos, nomeadamente da linguagem Perl, mas facilita a definição de novos tipos e validadores associados.

Referências

1. Sean M. Burke. *Perl & LWP*. O'Reilly, 2002.
2. James Clark and Steve DeRose. XML Path Language (XPath), 1999. `http://www.w3.org/TR/xpath`.
3. David C. Fallside. XML Schema — W3C Recommendation, 2001. `http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/`.
4. R. Gorin, P. Willisson, W. Buehring, and G. Kuenning. Ispell, a free software package for spell checking files, 1971.
5. Rick Jelliffe. Resource Directory (RDDL) for Schematron 1.5, 2003.
6. OSCAR. Open Standards for Container/Content Allowing Re-use — TMX home page, 2003. `http://www.lisa.org/tmx/`.
7. José Carlos Ramalho. Constrain content: Specification and processing. 2001. XML Europe.
8. Yves Savourel. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association, 1997.
9. Libxml. `http://www.libxml.org`.

Comparação de Linguagens para descrição de Interfaces baseadas em XML

Ricardo Alexandre Martins, José Carlos Ramalho, and Pedro Rangel Henriques

{ram,jcr,prh}@di.uminho.pt
Departamento de Informática — Universidade do Minho

Resumo Há alguns anos, houve um esforço considerável por parte dos meios académico e industrial para normalizar a representação de diferentes tipos de dados, de forma a facilitar a interoperabilidade das aplicações. Desse esforço resultou a adopção do XML, o que tornou possível representar dados, vindos de uma mesma fonte, de uma maneira estruturada e aberta (partilhada por todos) para, assim, transformá-los sistematicamente em diferentes resultados.
Actualmente, o desafio é estabelecer um formato único, também normalizado, para descrever as interfaces através das quais os dados podem ser visualizados e manipulados em ambientes web; a ideia é poder desenvolver os sítios WWW e as modernas interfaces web, de forma idêntica à que vem sendo usada na programação convencional. No momento, existem algumas propostas de linguagens formais baseadas em XML que visam solucionar este tipo de problema, de modo a que se possa gerar dinamicamente uma aplicação completa—como é sabido, havendo uma especificação rigorosa é possível criar geradores de programas.
Este artigo apresenta uma comparação entre três linguagens diferentes para definição de interfaces, a saber: XUL, *XML User Interface language*, UIML, *User Interface Markup Language*, e MXML, *Macromedia fleX Markup Language*. Serão analisadas as vantagens e desvantagens de cada linguagem através da avaliação de diferentes parâmetros de comparação, de modo a determinar a linguagem que oferece mais recursos para o utilizador. Acaba-se concluindo que a UIML é ligeiramente inferior às outras e que a MXML oferece de facto uma integração entre as várias componentes da aplicação que corresponde ao nível desejado (como se referiu acima).

Gerador de Web Services para cadeias de tranformações de documentos XML

José Carlos Ramalho, Pedro Taveira, Ricardo Ferreira, and Vasco Rocha

DI/UM

jcr@di.uminho.pt pjstaveira@netcabo.pt
ricardomiguel@myrealbox.com vmrocha@netcabo.pt

Resumo Os Web Services são cada vez mais utilizados na implementação de aplicações distribuídas. Apesar de utilizarem normas standard de baixa complexidade, a quantidade de normas utilizadas e a interacção entre elas aumenta razoavelmente a complexidade da sua implementação. A ideia inicial deste projecto foi estudar a viabilidade da geração automática de Web Services partindo de uma especificação abstracta do processo. Para isso definiu-se uma linguagem XML muito simples com a qual se especifica um processo ou uma cadeia de processos e criaram-se as transformações necessárias para desta especificação se chegar ao Web Service final.
Posteriormente, decidiu-se particularizar o sistema para Web Services que implementam cadeias de transformação XML. Estas cadeias de transformação correspondem a aplicações XML de segunda geração ou, por outras palavras, com reflexão a nível da transformação.
Neste momento, o sistema possui uma interface em .Net com a qual é possível especificar a cadeia de operações a realizar pelo Web Service e que permite gerar a aplicação servidor do serviço e a aplicação cliente, ambas em tecnologia .Net.

1 Introdução

Um dos principais problemas no mundo informático é a interoperabilidade estre as diversas plataformas. O aparecimento dos Web Services veio atenuar este factor, utilizando normas standard como o XML e tornando possível a comunicação entre aplicações como se estas fossem caixas negras, ou seja, é possível comunicar com as aplicações de uma forma normalizada desconhecendo os detalhes da sua implementação.

Mesmo com estas vantagens, à primeira vista, criar um Web Service pode não ser muito simples. Há várias camadas de software envolvidas. Há um conjunto de normas que se inter-relacionam e que o programador deve conhecer. Por estas razões, surgiram no mercado várias plataformas para desenvolvimento de Web Services em várias linguagens de programação. Se se optar por uma destas plataformas a criação de servidores de serviços e de clientes pode ser sistematizada podendo mesmo ocultar-se todo o processo de implementação. Foi este pressuposto que deu o mote deste trabalho. Assim o primeiro objectivo foi criar uma

linguagem XML com a qual fosse possível especificar ao nível mais abstracto possível um Web Service, e desenvolver as ferramentas necessárias que a partir da especificação de um Web Service gerasse o código para o implementar, quer o servidor quer o cliente.

Cedo se verificou que só se conseguia uma verdadeira abstracção se se concretizásse um pouco o Web Service, i. e., se o sistema que se estava a desenvolver estivesse preparado para gerar Web Services de uma determinada espécie. Com esse objectivo, seleccionou-se um conjunto de aplicações para as quais já existia há algum tempo a intenção de as expôr na Web. Estas aplicações são, muitas vezes, designadas por transformações XML de ordem superior e caracterizam-se por serem transformações com reflexão numa determinada fase ou em várias fases da sua execução. A execução de uma aplicação destas não é simples para o utilizador comum e há muito que se pensava em expô-las na Web como um serviço ocultando os pormenores, às vezes complicados da sua execução.

A plataforma escolhida para o desenvolvimento deste projecto foi a plataforma .NET da Microsoft. A escolha desta plataforma para o desenvolvimento deste projecto deveu-se ao facto de permitir uma implementação relativamente simples e intuitiva de Web Services.

Neste artigo, descrevem-se os passos seguidos na implementação de um gerador de Web Services para o tipo de aplicações descrito acima. O sistema desenvolvido pode ser adaptado facilmente para outro tipo de aplicações.

Na próxima secção descreve-se com um pouco mais de detalhe as cadeias de transformações de documentos XML. Na secção seguinte, apresenta-se a linguagem XML que se definiu para especificar a cadeia de processos que o Web Service a gerar deve implementar. A seguir descreve-se um pouco o processo de geração propriamente dito. O artigo termina com uma secção onde se faz uma síntese do trabalho realizado e onde se apontam linhas para trabalho futuro.

2 Cadeias de Transformações de Documentos XML

As aplicações XML implementadas através de cadeias de transformações com reflexão são aquilo que às vezes se designa por aplicações de ordem superior. Neste tipo de aplicações, o utilizador trabalha numa camada de abstracção superior ao de uma aplicação normal. A figura 1 apresenta o esquema duma aplicação deste tipo.

Como se pode ver na figura 1 o utilizador produz um documento XML com a especificação abstracta de um determinado processo de transformação; este documento é processado por uma metastylesheet (que implementa o nível superior de abstracção) e que gera como resultado uma stylesheet XSL específica que implementa o processo especificado; esta nova stylesheet pode depois ser usada para aplicar o processo de transformação especificado no nível superior a vários documentos XML concretos.

A figura 2 representa o mesmo esquema genérico mas agora concretizado numa aplicação concreta que utiliza estes dois níveis e que foi designada pelo autor por *XPW: Xpath Wrapper, uma ferramenta para auxiliar no ensino de*

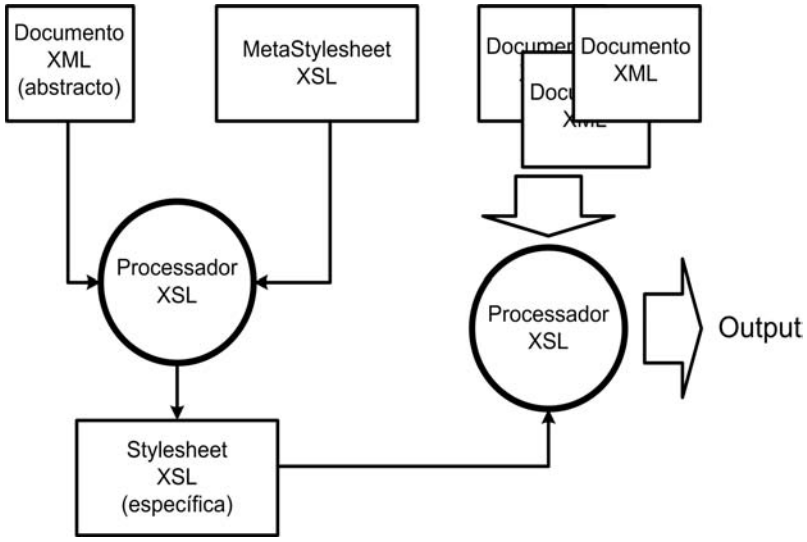


Figura 1. Esquema genérico de uma transformação com reflexão

XPath. Como o próprio nome indica esta ferramenta tem vindo a ser utilizada no ensino de XPath.

O XPW foi construído de modo a permitir ao utilizador realizar queries, especificadas em XPath, ao conteúdo de documentos XML. Assim, numa primeira etapa, o utilizador especifica num documento XML as queries que quer realizar; numa segunda etapa, esse documento é passado a um processador de XSL conjuntamente com a metastylesheet que implementa o nível abstracto do XPW; deste processo resulta uma stylesheet que implementa as queries especificadas; quando aplicada a um documento XML esta stylesheet irá extrair as partes seleccionáveis com as queries especificadas.

Em resumo, uma aplicação deste tipo tem dois níveis de transformação. O primeiro do abstracto para o concreto, e o segundo a transformação em concreto. Se, por exemplo, se acrescentasse ao XPW uma terceira transformação para tratar os resultados teríamos uma cadeia de três transformações.

Quando já se trabalha há algum tempo na área da transformação documental (documentos XML) é normal que o nível de abstracção, das aplicações que se vão criando, vá aumentando. Isto vai-se traduzindo em, cada vez maiores, cadeias de transformações.

É assim que surge a ideia de tornar estas cadeias transparentes para o utilizador. A estratégia para as ocultar, que foi seguida neste trabalho, foi colocá-las por detrás de um Web Service. Como este Web Service é, em termos funcionais, muito semelhante de umas aplicações para as outras, surgiu também a ideia de tornar a sua geração automática a partir da especificação abstracta de uma cadeia de transformações.

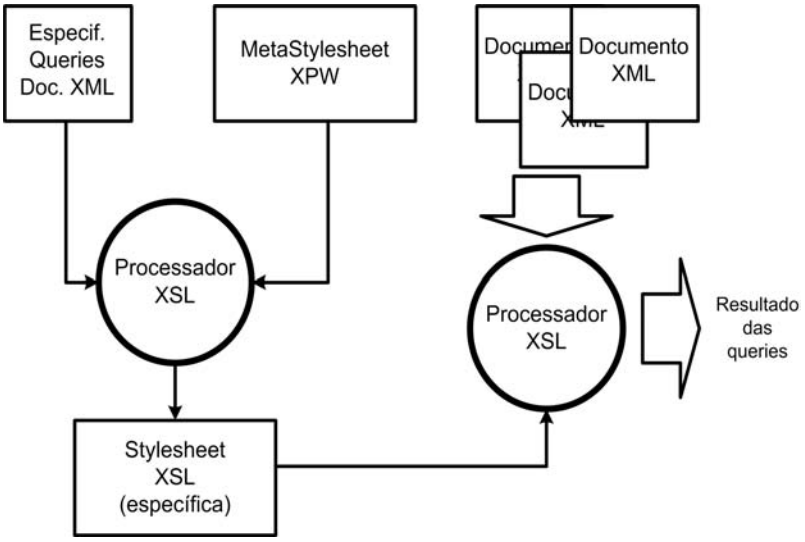


Figura 2. Esquema do XPath Wrapper

Na próxima secção, apresenta-se a linguagem XML definida para especificar cadeias de transformação.

3 Especificação de Cadeias de Transformação

O primeiro passo no processo de geração de Web Services para cadeias de transformações de documentos XML é a definição da cadeia de transformação. Para isso foi definida uma linguagem XML.

Para especificar uma cadeia de transformações começa-se por indicar todos os documentos XML e XSL que vão participar nas transformações da cadeia. Cada documento terá um identificador único e um atributo que identifica a sua localização (cliente – o documento é fornecido pelo cliente; servidor – o documento encontra-se no servidor).

Depois de definidos todos os documentos XML e XSL é preciso definir definir as transformações que precisam de ser executadas. Uma transformação define-se indicando qual o documento XML e qual a stylesheet XSL que irão participar nela. O documento e a stylesheet são referidos pelos identificadores únicos associados aos documentos aquando a sua definição. Assim, uma transformação é definida por um par (id_1, id_2): o atributo id_1 identifica o documento a ser transformado e o atributo id_2 identifica a folha de estilo a aplicar. Cada transformação também tem um identificador único que ficará depois associado ao seu resultado. Desta maneira, o resultado de uma transformação poderá ser utilizado como entrada de uma nova transformação.

O utilizador também deve associar um título e um nome à cadeia. O título irá aparecer na página do cliente. O nome possui um atributo que permite especifiar

se o resultado da cadeia de transformações é HTML ou XML. Esta funcionalidade foi implementada porque muitas vezes, da última transformação da cadeia resulta uma página HTML para melhor se poder visualizar os resultados obtidos e os Web Services precisam de indicar que tipo de mensagens estão a trocar.

Na figura 3 mostra-se uma representação gráfica da estrutura da linguagem.

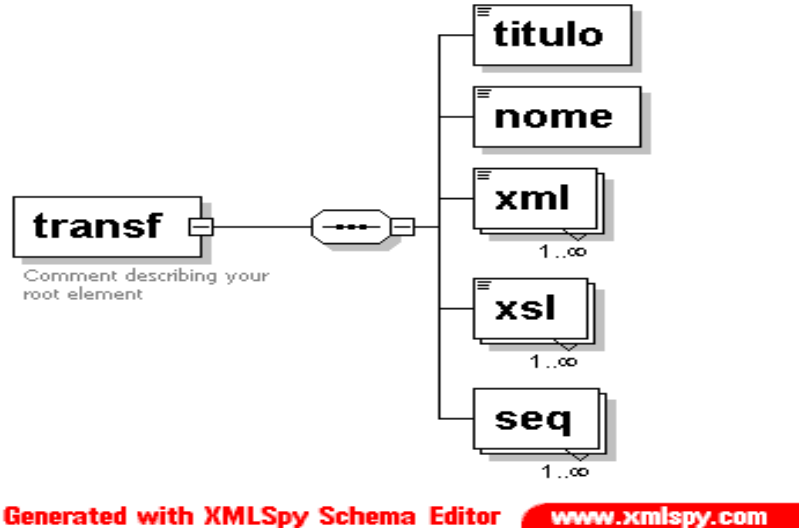


Figura 3. Schema da Linguagem de Anotação para Especificação de Cadeias de Tranformações

A título de exemplo, apresenta-se a seguir, a especificação da cadeia de transformações para uma utilização do XPW.

```
<?xml version="1.0" encoding="UTF-8"?>
<transf>
  <titulo>Web Service do XPath</titulo>
  <nome resp="xml">Queries XPath</nome>
  <xml id="poema" tipo="cliente">Ficheiro do Poema</xml>
  <xml id="querie" tipo="cliente">Ficheiro com as Queries</xml>
  <xsl id="xpath" tipo="servidor">Ficheiro com a metastylesheet</xsl>
  <seq id1="querie" id2="xpath" id="res"/>
  <seq id1="poema" id2="res" id="fim"/>
</transf>
```

A cadeia de transformações descrita acima define uma cadeia onde estão envolvidos dois documentos XML (`poema` e `querie`, localizados no cliente) e um documento XSL (`XPW` localizado no servidor). O documento `querie` é transformado pela folha de estilo `XPW` e o seu resultado irá ser uma folha de estilo que irá transformar o documento poema, obtendo-se assim o documento final.

Na próxima secção, discute-se a geração do serviço.

4 Geração do Web Service

A parte crítica deste projecto é a geração do Web Service a partir da linguagem descrita na secção anterior. No sentido de simplificar a comunicação entre o cliente e o servidor, decidiu-se utilizar `Strings` para representar os documentos que são transportados pela "rede". Por este motivo, a implementação de um cliente tornou-se quase obrigatória dado que, para podermos utilizar a página de teste que a plataforma .NET disponibiliza teríamos que introduzir todo o documento à mão nas respectivas caixas de texto. A geração do código do cliente e do servidor é efectuada por duas folhas de estilo distintas, que aplicadas a um documento XML escrito na linguagem descrita na última secção, irão gerar duas vistas distintas: o código do cliente e o código do servidor.

As stylesheets são demasiado extensas para serem discutidas aqui. No entanto, apresenta-se a seguir o servidor gerado para a especificação exemplo mostrada acima.

4.1 Servidor Gerado

```
<%@ WebService Language="c#" Class="WSGen.Queries XPath" %>
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;
using System.IO;
using System.Xml.Serialization;

namespace WSGen
{
    [WebService(Namespace="http://www.WSGen.com/")]
    public class Queries XPath : System.Web.Services.WebService
    {
        [WebMethod]
        public string transf(string poemap, string queriep)
        {
            XmlDocument poema = new XmlDocument();
            poema.LoadXml(poemap);
            XmlDocument querie = new XmlDocument();
            querie.LoadXml(queriep);
            XslTransform xpath = new XslTransform();
            XmlTextReader xpathr = new XmlTextReader(Server.MapPath("xpath.xslt"));
            xpathr.MoveToContent();
            xpath.Load(xpathr);
```

```
30              XmlReader resp = xpath.Transform(querie, null);
31              resp.MoveToContent();
32              XslTransform res = new XslTransform();
33              res.Load(resp);
34              XmlReader fimp = res.Transform(poema, null);
35              fimp.MoveToContent();
36              XmlDocument fimx = new XmlDocument();
37              fimx.Load(fimp);
38              return fimx.DocumentElement.OuterXml;
39          }
40      }
41  }
```

Note que o código gerado para o servidor é implementado na linguagem C# e o gerado para o cliente é em .ASP da plataforma .NET.

5 Conclusões

Para simplifiar ainda mais todo este processo criou-se um interface em C# que permite ao utilizador especificar a linguagem de uma forma intuitiva, bem como gerar o servidor e o respectivo cliente a partir das folhas de estilo de uma forma bastante simples e automática.

A utilização da aplicação desenvolvida no ensino começará já nos próximos meses. Utilizando esta plataforma pode-se criar Web Services que ocultam os pormenores mais complexos das aplicações fazendo com que o aluno se concentre apenas naquilo que o nível de abstracção mais elevado da aplicação lhe exige.

Índice de Autores

Abreu, Célio, 115
Abreu, Luís, 3
Aguiar, Ademar, 183
Almeida, José João, 59, 217
Amaral, Jorge, 205
Araújo, Mário André, 88
Ariza, Cesar, 166

Badros, Greg, 183
Baptista, Ana Alice, 166
Bento, Nuno, 4
Biscaia, Miguel, 23

Cardoso, Jorge, 71
Castro, José Alves de, 23
Clérigo, Filipe Costa, 21
Costa, Francisco, 23

David, Gabriel, 123, 183

Ferreira, Helder Filipe P.C., 101
Ferreira, Jenny, 197
Ferreira, Márcio, 115
Ferreira, Miguel, 147
Ferreira, Ricardo, 226
Ferreira, Rogério, 45
Freitas, David, 205

Gonçalves, Rui Alberto, 115
Guinovart, Xavier Gomez, 59

Henriques, Pedro Rangel, 170, 225

Leal, José Paulo, 45
Librelotto, Giovani Rubert, 170
Lopes, Carlos J. Feijó, 33
Lopes, João Correia, 6, 71

Martins, Ricardo Alexandre, 225

Nunes, Sérgio Sobral, 123

Pereira, Carlos Ângelo, 6
Pereira, Tomás, 136
Pias, João, 115

Ramalho, José Carlos, 33, 147, 170, 225, 226
Rocha, Artur, 71
Rocha, Jorge Gustavo, 88
Rocha, Vasco, 226
Rodrigues, Marco, 197
Ruivo, José, 136

Santos, Vitor, 23
Seabra, Pedro, 21
Silva, José António, 22
Simões, Alberto Manuel, 59, 217

Taveira, Pedro, 226

Vila Cova, André, 136