XATA2005

XML: Aplicações e Tecnologias Associadas

3ª Conferência Nacional

Editores:

José Carlos Ramalho
Alberto Simões
João Correia Lopes

10 e 11 de Fevereiro de 2005
Braga

Ficha Técnica:

Design Gráfico: Benedita Contente Henriques
Redacção: J. C. Ramalho e J. C. Lopes
Composição: J. C. Ramalho e J. C. Lopes
Tiragem: 150 exemplares
Edição: Fevereiro de 2005
ISBN: 972-99166-1-6

Prefácio:

Esta é a terceira conferência sobre a temática do XML, que tenta reunir anualmente a comunidade XML portuguesa.

Olhando para o passado, e estudando o que aconteceu, podemos observar uma evolução interessante.

Na primeira conferência (2003), houve uma vintena de trabalhos submetidos, maioritariamente da autoria ou da supervisão dos membros que integravam a comissão organizadora.

Na segunda conferência (2004), houve uma participação mais forte da comunidade portuguesa mas ainda com números pouco expressivos. Nesta altura, apostou-se também numa forte participação da indústria, o que se traduziu num conjunto apreciável de apresentações de casos reais.

Nesta terceira edição, a participação excedeu as nossas expectativas. Houve uma forte adesão nacional e internacional (Espanha e Brasil, o que para um evento onde se pretende privilegiar a língua portuguesa é ainda mais significativo). A distribuição geográfica em Portugal também aumentou, havendo mais instituições participantes.

Se analisarmos as temáticas, abordadas nas três conferências, percebemos que também aqui há uma evolução no sentido de uma maior maturidade. Enquanto que no primeiro encontro, os trabalhos abordavam problemas emergentes na utilização da tecnologia, no segundo encontro a grande incidência foi nos *Web Services*, uma nova tecnologia baseada em XML e, no terceiro, a maior incidência é na construção de repositórios, motores de pesquisa e linguagens de interrogação. Isto poderá revelar que, de certa forma, os investigadores e os utilizadores já dominam bem a tecnologia, já têm as sua linhas de produção documental em XML e que agora é chegado o momento de armazenar aquela informação em repositórios e de a utilizar a partir deles. Se reflectirmos um pouco sobre o ciclo de vida documental, podemos concluir que o armazenamento aparece como uma das últimas fases. Por isso, podemos dizer que a comunidade portuguesa de XML "está a fechar" o ciclo de desenvolvimento documental XML.

Nesta edição, houve um elevado número de submissões o que levou a uma sobrecarga de trabalho por parte dos revisores. Estes conseguiram desempenhar bem o seu papel e cumprir as datas definidas pelo "chair". Mas o que aconteceu fez-nos pensar que há necessidade de alargar ainda mais a Comissão Científica. Como consequência do número elevado de submissões houve algumas rejeições. Aos respectivos autores, fica aqui expresso um voto para que melhorem os respectivos trabalhos e de que esperamos ter contribuído para isso com os relatórios de revisão que foram enviados.

XATA 2005

A XATA 2005 surge na sequência das edições de 2003 e 2004. Quem participou nas edições anteriores observou grandes alterações. Todo o processo de submissão e revisão de artigos foi gerido electronicamente. Aos autores exigiu-se mais que em edições anteriores tendo havido três fases de submissão:

resumo, artigo completo e artigo completo com revisão. Por tudo isto, a qualidade final do conjunto é boa e coerente. Foi ainda introduzida uma sessão de posters para permitir a apresentação de trabalhos em curso, nomeadamente da autoria de alunos.

As áreas que são abordadas nesta edição não são muito diferentes das edições anteriores havendo talvez um maior equilíbrio e melhor distribuição pelas diferentes temáticas.

Estrutura da Conferência

À semelhança das edições anteriores, a conferência encontra-se dividida em oito sessões temáticas repartidas pelos dois dias:

Sessão I: XML e Bases de Dados – repositórios Esta sessão está dedicada à implementação de repositórios documentais.

Sessão II: Web Semântica e Ontologias A Web Semântica é um tema recorrente na XATA. Mais uma vez, temos um conjunto de trabalhos que nos demonstram as várias vertentes de investigação na área.

Sessão III: XML e Bases de Dados – linguagens de interrogação Esta sessão complementa a linha de trabalhos agrupados na primeira sessão.

Sessão IV: Workflow e Gestão Documental – Nesta sessão, serão apresentados e discutidos trabalhos que abordam problemas relacionados com o workflow e a gestão documental no seio empresarial.

Sessão V: Web Services, Arquitecturas e Casos Práticos – Esta sessão apresenta alguns trabalhos onde é utilizada a tecnologia de Web Services para solucionar o problema ou para criar uma solução mais interessante.

Sessão VI: Edição e Processamento de Documentos XML – Esta sessão engloba trabalhos numa área onde a concorrência é enorme. Quem se arriscaria a criar mais um editor de XML? Ou um motor de processamento? É o que estes autores nos vêm mostrar e principalmente discutir a sua motivação.

Sessão VII: E-Learning – O E-Learning é uma área em clara expansão em Portugal e que atrai gente de todos os domínios do saber. Aqui está a resposta da comunidade XML apresentando propostas para a criação de conteúdos.

Sessão VIII: Aplicações XML – Esta é, talvez, a sessão mais heterogénea. Apresenta um conjunto de trabalhos relacionados com problemas reais e específicos de áreas bem diferentes.

Sessão IX: Posters – Esta será a nossa sessão "fantasma" pois não tem nenhuma fatia de tempo atribuída. No entanto, os posters serão colocados na sala contígua ao auditório, ou num local mais revolucionário se "as autoridades locais" permitirem, e haverá moderadores que se encarregarão de lançar a discussão sobre cada um dos trabalhos.

Esta edição do XATA, tem mais que as anteriores mas poderá ter ainda muito mais. Participe activamente no evento, partilhe as suas ideias e não receie levantar discussões. É para isso que todos lá estaremos.

Uns dias bem "xatos" para todos

José Carlos Ramalho

Comissão Organizadora:

José Carlos Ramalho	DI/UM — Chair
João Correia Lopes	FEUP/INESC Porto — Webchair
Pedro Henriques	DI/UM — Posters
Alberto Simões	DI/UM
Gabriel David	FEUP/INESC Porto
Luís Ferreira	IPCA
Maria Cristina Ribeiro	FEUP/INESC Porto

Comissão Científica:

José Carlos Ramalho	Universidade do Minho — Chair
Ademar Aguiar	Universidade do Porto, INESC Porto
Adérito Marcos	Universidade do Minho, CCG
Alberto Rodrigues da Silva	Instituto Superior Técnico
Ana Alice Baptista	Universidade do Minho
Gabriel David	Universidade do Porto, INESC Porto
João Correia Lopes	Universidade do Porto, INESC Porto
João Moura Pires	Universidade Nova de Lisboa
José João Almeida	Universidade do Minho
José Paulo Leal	Universidade do Porto
Luis Carriço	Universidade de Lisboa
Luís Ferreira	Instituto Politécnico do Cávado e do Ave
Luis Silva	Universidade de Coimbra
Maria Cristina Ribeiro	Universidade do Porto, INESC Porto
Marta Henriques Jacinto	ITIJ
Pedro Antunes	Universidade de Lisboa
Pedro Henriques	Universidade do Minho
Salvador Abreu	Universidade de Évora

Revisores Adicionais:

Alberto Simões	Universidade do Minho
Helena Sofia Pinto	Instituto Superior Técnico
Marco Sá	Universidade de Lisboa
Miguel Ferreira	Universidade de Aveiro
Miguel Oliveira	Universidade do Minho
Miguel Rodrigues	Universidade de Lisboa
Rui Lopes	Universidade de Lisboa
Vitor Beires Nogueira	Universidade de Évora

Agradecimentos:

Os editores deste livro de actas deixam aqui expresso o seu agradecimento a todas as pessoas que participaram nesta empreitada, sem as quais não teria sido possível concretizá-la.

Em primeiro lugar, agradecemos a todos os autores a produção dos trabalhos que aqui ficam publicados. Sem eles não existiria uma XATA2005. Obrigado por terem aceite o desafio.

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

Fica aqui também, um agradecimento especial à Cristina Ferreira, economista do Departamento de Informática da Universidade do Minho, que tratou de várias diligências administrativas e da contabilidade do evento.

Por fim, a todos os participantes, que decidirem comparecer num evento onde se pretende privilegiar a troca e discussão de ideias, a sua presença que vem sempre engrandecer o debate.

Vamos ser xatos por dois dias no XATA2005

José Carlos Ramalho
Alberto Simões
João Correia Lopes

Patrocínios:

A Comissão Organizadora fica muito grata às seguintes organizações pelo seu suporte:

Microsoft Microsoft Corporation Portugal
DI/UM Departamento de Informática da Universidade do Minho
DEEC Departamento de Engenharia Electrotécnica e de Computadores da Universidade do Porto

Conteúdo

V Web Services, Arquitecturas e Casos Práticos

VI Edição e Processamento de Documentos XML

VII E-learning

X Índices Remissivos

Acceso a Datos Relacionales Bajo un Entorno XML. Un Caso Práctico

Ana Fermoso García, Roberto Berjón Gallinas

Escuela Universitaria de Informática, Universidad Pontificia de Salamanca, Compañía 5, 37002 Salamanca, España
{afermosoga, rberjonga}@upsa.es

Abstract. En nuestros días y especialmente con la explosión de Internet, las organizaciones tienen que manejar multitud de datos y además datos de fuentes heterogéneas, como por ejemplo datos procedentes de las tradicionales bases de datos relacionales y otros procedentes de nuevas tipos de datos aparecidos con la Web, como los documentos HTML y XML. Las organizaciones tienen que trabajar con todos estos tipos de datos al mismo tiempo, por lo que se hacen necesarios sistemas que actúen como "mediadores" para facilitar la integración de esta información. En nuestra investigación nos centraremos en los datos XML, estándar de facto para el intercambio de información en la Web, y las bases de datos relacionales, que continúan siendo uno de los principales sistemas de almacenamiento. El propósito es que los datos necesarios en cada momento acaben en el mismo formato, XML en este caso al ser este el lenguaje de la Web, antes de manejarlos de forma conjunta, y para ello proponemos un nuevo sistema al que hemos denominado XBD. XBD es un sistema de acceso eficiente a datos relacionales bajo un entorno XML. XBD incluye un modelo de adaptación para que cualquier base de datos relacional pueda ser consultada en nuestro sistema, dos nuevos lenguajes para realizar las consultas y la herramienta software que implementa la funcionalidad de este sistema de consulta, y todo ello, tanto el interfaz de consulta, como los lenguajes, así como los resultados de las consultas, tendrán una apariencia XML.

1 Introducción

Internet esta provocando grandes cambios en las organizaciones y su gestión. Toda organización tiene que acabar migrando a la Web. Por otro lado, la sociedad actual es también la sociedad de la información y el conocimiento, la información es poder y los datos tienen que acabar convirtiéndose en "información", que es un concepto más amplio. Cuando un dato se convierte en información quiere decir que puede servirnos para la toma de decisiones del negocio, por ejemplo. Además la cantidad de información que se maneja ha sufrido un gran incremento en cuanto a su volumen y heterogeneidad, los datos pueden venir de fuentes muy diferentes.

Los nuevos entornos de trabajo han hecho aparecer nuevos tipos y formatos de datos, como los utilizados para la publicación de información en la Web, HTML o XML [8]. Sin embargo, en paralelo se tiene que seguir trabajando con los datos almacenados en las tradicionales bases de datos, como las bases de datos relacionales.

En conclusión, en la organización se tiene que trabajar al mismo tiempo con datos procedentes de fuentes heterogéneas más o menos estructuradas, y por tanto será necesario en muchos casos, algún método intermedio que sirva como mediador y que permita integrar estos datos con diferentes formatos para trabajar de forma conjunta con ellos.

En cuanto a los nuevos tipos de datos creados como respuesta a las necesidades de la Web, destaca XML como estándar de facto para la representación e intercambio de información en este entorno. Una de las consecuencias de trabajar en este medio suele ser la necesidad de intercambiar información con otros sistemas y para ello el formato XML se ha convertido en el formato más usado. Incluso están apareciendo sistemas de almacenamiento específico para este tipo de datos, que van más allá de los documentos XML tratados de forma independiente, nos referimos a las bases de datos nativas XML.

El mundo de las bases de datos también se ha visto afectado por los nuevos cambios. Por un lado, las tradicionales bases de datos relacionales siguen almacenando la mayor parte de información de la organización, pero en las últimas versiones de los sistemas de gestión de base de datos, se han visto obligados también a adaptarse a las nuevas necesidades, por ejemplo permitiendo también tratar y almacenar información en formato XML, aunque el tema del almacenamiento de información XML en bases de datos relacionales quede fuera del ámbito de este trabajo.

También en relación con las bases de datos han aparecido nuevos tipos como los data warehouse, que se han convertido en sistemas de almacenamiento de datos de fuentes heterogéneas, desde los que luego se analizan esos datos para convertirlos en información y conocimiento y así ayudar a la toma de decisiones de la organización. En estos nuevos sistemas se harán también necesarios métodos para permitir integrar estos datos como paso previo a su análisis.

En nuestro trabajo, demostrada la importancia de XML en el medio Web por un lado, y la gran cantidad de información que continua siendo almacenada en las bases de datos relacionales, estudiaremos como podemos tratar de forma conjunta datos procedentes de ambos entornos. Si los datos procedentes de uno u otro entorno pudiesen estar en el mismo formato común, XML al ser éste el lenguaje de la Web, sería más fácil la integración de los mismos, tanto si se necesitan para el intercambio de información a través de la red, como si se utilizan para su posterior almacenamiento y tratamiento dentro de un data warehouse, como acabamos de comentar.

El objeto de nuestra investigación será precisamente, proporcionar un entorno de acceso común para datos en formato XML y relacionales al que hemos denominado XBD (acceso a Bases de Datos con XML) . Para el usuario resultará un sistema transparente en cuanto a que accederá a datos XML y relacionales de una forma muy similar, con lo que para él es como si toda la información estuviera en formato XML y sólo estuviese consultando datos de este tipo. Se trata por tanto de acceder a datos relacionales bajo un sistema con apariencia de consulta a datos XML. El resultado obtenido tras las consultas a la base de datos relacional, también sería XML.

Según lo expuesto, una vez vistas en esta introducción las necesidades que las nuevas tecnologías plantean a las organizaciones, queda con ello justificado la utilidad del nuevo entorno de consulta a bases de datos relacionales que proponemos.

En el siguiente apartado se tratará de hacer un estudio de la situación actual de otros sistemas que también tienen relación con el tema de las bases de datos y XML, permitiendo obtener datos XML de fuentes relacionales. A continuación, y una vez analizados las limitaciones de los sistemas estudiados con relación a nuestros objetivos, ya se pasaría por fin a describir en detalle las características de nuestro nuevo sistema de consulta XBD que hemos diseñado como respuesta a las nuevas necesidades de gestión de datos heterogéneos. La descripción del sistema incluirá su arquitectura, el modelo de adaptación que se propone para que cualquier base de datos relacional pueda ser consultada en el entorno XBD, los lenguajes de consulta que en él se utilizan y el procesamiento en general de las consultas en este sistema. Finalmente se demostrarán sus ventajas y las pruebas que con él se han realizado y por último se expondrían las nuevas líneas de investigación en las que estamos ya trabajando y que siguen relacionadas con el área de la integración entre datos relacionales y XML, pero tratada desde una nueva perspectiva.

2 Obtención de datos XML de Fuentes Relacionales

El propósito de nuestra investigación es ayudar a integrar bases de datos relacionales con datos XML, obteniendo datos XML a partir de información almacenada en base de datos relacionales. Una vez logrado que tanto la información procedente de fuentes XML como de fuentes relacionales esté en el mismo formato, en nuestro caso XML, resultará más fácil trabajar de forma conjunta con ella, tal como se hace necesario en muchos sistemas actuales.

Las últimas versiones de los Sistemas de Gestión de Base de Datos (SGBD) más utilizados como Oracle [6][7], Microsoft SQL Server [5] o IBM DB2 [4], han tenido que adaptarse a los nuevos tipos de información permitiendo trabajar de una forma más o menos directa con datos XML.

En consecuencia, estos SGBD ya permiten almacenar datos XML junto a sus tradicionales datos relacionales, definiendo nuevos tipos de datos como XMLType en Oracle y SQL Server, o XML Column y XML Collection en DB2. Una vez almacenada la información XML en la base de datos habrá que disponer también de herramientas para acceder a ella. Si la información en XML almacenada en la base de datos tiene una estructura más o menos fija (documento XML centrado en los datos) y reconocida por la base de datos a través de una especie de esquema, conocido por ejemplo como DAD (Definición de Acceso a Datos) en IBM DB2, se podrá acceder en detalle a la información XML almacenada. Esto se consigue en la mayor parte de los SGBD ampliando el lenguaje de consulta estándar SQL con construcciones de XPath embebidas en dichas consultas SQL y dirigidas a los datos XML. Si no hay una estructura reconocida de la información XML almacenada, es decir, se trata de documentos XML centrados en el documento sin una estructura prefijada, no en los datos, los SGBD permitirán acceder a todo el documento XML en su conjunto, al documento XML completo, no a una parte en particular del mismo.

Sin embargo, todos los aspectos de las nuevas versiones de los SGBD que acabamos de comentar en relación a XML se refieren al almacenamiento de este tipo de información en las bases de datos relacionales y nosotros estamos más interesados

no en el almacenamiento, sino en la obtención de datos en formato XML generados a partir de información que no está en este formato. En este sentido también los SGBD están proporcionando nuevas herramientas que permiten realizar consultas a las bases de datos en el tradicional lenguaje de consultas de datos relacionales SQL, y obtiene los resultados de la consulta en XML.

Los mismos resultados que con estas últimas herramientas se pueden conseguir con las páginas activas como las JSP (Java Server Pages), ASP (Active Server Pages) o PHP. En todas ellas es posible incluir consultas a base de datos en SQL y obtener los resultados de dichas consultas en XML. Estos resultados se añadirán al resto del contenido de la página Web a la hora de mostrar su información final.

En los dos casos anteriores, tanto los de la herramientas de los SGBD que retornan los resultados en XML como en las páginas activas, la consulta se realiza utilizando un lenguaje propio del entorno de base de datos como SQL. Sin embargo, nuestro objetivo es que para facilitar la integración de datos relacionales con datos XML y que el usuario no tenga que preocuparse ni del formato, ni del tipo de fuente al que accede, y ni siquiera del lenguaje de consulta, sino que para él es como si todo lo que consulta fuese información en XML, por esta razón el lenguaje de consulta también debería estar basado en XML, no ser un lenguaje del entorno de base de datos. Esto por tanto representa una desventaja de las herramientas anteriores si tenemos en cuenta nuestras metas.

En este sentido, existen ya algunos sistemas que permiten la consulta a bases de datos relacionales obteniendo los resultados en XML, pero además usando lenguajes de consulta propios del entorno XMLm y sin que los datos originales de la base de datos tengan que sufrir ningún tipo de transformación ni traducción a ningún formato. Dentro de estos sistemas destacan SilkRoute [1] y XTABLES [2].

En ambos trabajos se propone la creación de una vista virtual en XML de los datos de la base de datos y sobre esta vista se realizan las consultas. La vista se considera virtual porque en realidad los datos de la base de datos no se transforman a XML, se le presentan al usuario con apariencia XML, pero en realidad ni se traducen, ni se almacenan en XML. En los dos sistemas se realiza una consulta inicial a partir de la cual se genera esa vista virtual XML, que contendrá a los datos de la base de datos afectados por la consulta y el resto de consultas tendrán que hacerse sobre los datos de esa vista virtual y en un lenguaje de consulta basado en XML, es decir, se condiciona al resto de consultas a los resultados obtenidos en esa consulta inicial.

En XML la vista XML virtual recibe el nombre de “default view” y como único lenguaje de consulta, tanto para la consulta inicial como para el resto, se utiliza XQuery. En SilkRoute la vista virtual obtenida tras la primera consulta recibe el nombre de “query view” y para obtenerla se utiliza un lenguaje propio de este sistema denominado RXL. El resto de consultas se realizarán sobre los datos de la vista virtual pero utilizando otro lenguaje de consulta denominado XML-QL.

El fin último de estos dos sistemas coinciden con el nuestro, se consulta una base de datos relacional obteniendo los resultados de la consulta en formato XML y además utilizando también lenguajes basados en XML para hacer las consultas y sin necesidad de realizar ninguna transformación de los datos a consultar. El inconveniente sin embargo de estos dos sistemas, es que en ambos existe una consulta inicial que condiciona al resto de consultas, tal que sólo podrán ser consultados los datos afectados por esta primera consulta. En el sistema que nosotros proponemos se

consiguen los mismos resultados, consultar datos relacionales con lenguajes de consulta basados en XML y obtener también los resultados en XML, pero de forma que en cualquier momento se pueda consultar cualquier parte o dato de la base de datos, no existirá una consulta inicial que condicione el resto de consultas, y esta es otra de sus ventajas. Como luego veremos en este nuevo sistema al que denominamos XBD ya no existe ninguna vista virtual porque el proceso de adaptación y consulta de cualquier base de datos relacional bajo este entorno es distinto a la idea de la "vista virtual XML", en su lugar se crea el "Esquema XML De Base de Datos" que es un documento XML con un significado distinto al de la vista virtual.

Analizadas las características de estos sistemas que permiten obtener datos XML de fuentes relacionales y expuestas sus limitaciones e inconvenientes de acuerdo a nuestros objetivos, a continuación exponemos las características del nuevo sistema que nosotros proponemos denominado XBD, y después demostraremos sus ventajas y bondades con relación a los sistemas analizados y a nuestras metas.

3. El nuevo Sistema de Consulta XBD

La idea fundamental del nuevo sistema XBD (XML para Bases de Datos) que nosotros proponemos, y del que podemos encontrar un estudio más detallado en [1], trata de solventar los principales problemas de los sistemas analizados en relación con el tratamiento de datos relacionales y XML en el entorno Web.

En nuestro sistema será posible consultar datos relacionales bajo un entorno con apariencia XML, de hecho, para el usuario la apariencia es prácticamente igual a la consulta a datos XML incluido el lenguaje de consulta, como luego veremos, y los resultados obtenidos, que también serán XML.

Para poder hacer efectivas estas consultas además, la gran ventaja es que no se necesita hacer ninguna conversión de información. No es necesario traducir los datos relacionales a XML, y ni siquiera será necesario mostrar estos datos como una vista XML.

Para conseguir estos objetivos hay que afrontar dos aspectos fundamentales, por una parte, cómo se van a consultar los datos relacionales, con qué lenguaje, y en segundo lugar, cuál debe ser la arquitectura interna del sistema que permita que hechas las consultas a la base de datos en el lenguaje del sistema, los resultados obtenidos en formato ya XML, sean los correctos.

Para permitir consultar una base de datos desde el entorno XBD serán necesarios dos procesos básicos. El primero permitirá adaptar la base de datos de modo que esta sea consultada en nuestro sistema. Para ello, ahora no se obtendrá una vista virtual en XML como en los casos anteriores, sino que se propondrá otra solución distinta de modo que como resultado del proceso se obtendrá el documento en formato XML que denominamos "Esquema XML de BD" cuyo significado explicaremos luego. Una vez adaptada la base de datos, el siguiente proceso ya nos permitirá que sea consultada. Para ello será necesario especificar por un lado, las características del lenguaje de consulta y por otro, el proceso interno que permita que estas consultas se ejecuten sobre la base de datos y se obtengan sus resultados en XML. De esta forma,

internamente XBD estará organizado en dos subsistemas principales que luego detallaremos al describir su arquitectura, el subsistema de adaptación y el de consulta.

En primer lugar analizaremos el tema del lenguaje a utilizar para realizar las consultas a la base de datos y luego ya el de la arquitectura del sistema con los dos subsistemas que lo forman.

4. Lenguajes de Consulta en el Sistema XBD

Si se desea que todo el sistema de consulta, incluido el lenguaje que se utiliza para dichas consultas, tenga una apariencia XML, tenemos dos alternativas: o crear un nuevo lenguaje de consulta basado en XML o reutilizar alguno de los ya existentes adaptándolo a nuestras necesidades.

Como deseamos que para el usuario resulte un sistema transparente y que de hecho para él es como si consultase información XML, la primera solución quedaría descartada porque se trataría de que el usuario no tuviese que emplear apenas tiempo de aprendizaje utilizando algún lenguaje que ya conozca del entorno XML.

En cuanto a lenguajes de consulta del entorno XML, el lenguaje de consulta por excelencia es XQuery [12], sin embargo XSL [11], aunque realmente es un lenguaje más de presentación que de consulta, indirectamente también nos va a permitir consultar información. En definitiva, en nuestro sistema se van a utilizar estos dos lenguajes, previa adaptación de los mismos a la consulta de base de datos, lo que implicará añadir alguna modificación, pero muy pequeña, a la sintaxis de las versiones estándar de estos lenguajes.

A los lenguajes obtenidos de esta adaptación, que son los que se van a usar para hacer las consultas en XBD les denominaremos *XQuery Adaptado* y *XSL Adaptado*. Su sintaxis es muy semejante a la de los lenguajes XQuery y XSL estándar, la diferencia fundamental radica en el significado que ahora toman las construcciones de estos lenguajes al utilizarlas para la consulta a base de datos. Cambia la semántica, pero la sintaxis es prácticamente la misma.

A continuación explicamos las características fundamentales de ambos lenguajes.

4.1 XSL Adaptado

Para adaptar el lenguaje XSL a nuestro sistema XBD sólo necesitamos algunas de la construcciones del lenguaje XSL estándar: `stylesheet, apply-templates, template, value-of, if, for-each, when y short`. A todas estas construcciones se les añade el prefijo "`xsl_adapt`" para señalar que forman parte de este nuevo lenguaje y que tienen un significado especial respecto a la versión estándar.

En relación con la consulta a bases de datos y teniendo en cuenta que al final, como resultado de la consulta en XSL adaptado, se obtiene una sentencia SELECT de SQL de consulta a base de datos, la equivalencia o significado de las construcciones en XSL para obtener la SELECT de SQL equivalente, sería la siguiente:

- *xsl_adapt:stylesheet:*

Contiene información acerca del fichero que contiene el *esquema XML de base de datos*, que es un documento XML obtenido a partir del proceso de adaptación de la base de datos al sistema XBD. Este fichero contiene información acerca de la estructura de la base de datos: sus entidades, atributos y relaciones. El nombre de este fichero resultado del proceso de adaptación, aparece como valor del atributo "`Schema_db`" de esta construcción.
Además esta construcción también contiene información sobre como conectarse a la base de datos a la que se refiere la consulta.

- *xsl:adapt:apply-template / xsl_adapt:template*
  Cada `template` equivale a una sentencia SELECT de SQL. Ambas construcciones actúan de forma combinada. En el `apply-template` determinamos la tabla principal de la consulta y dentro del `template` asociado aparecerá el resto del contenido de la sentencia SELECT.
- *xsl_adapt:value-of*
  Muestra los campos consultados de la base de datos. Estos campos aparecerán en la sentencia SELECT cuando la consulta se traduce a SQL.
- *xsl_adapt:if*
  Añade una condición a la consulta. Esta construcción se traducirá a una cláusula WHERE en la sentencia SELECT de SQL asociada.
- *xsl_adapt:for-each*
  Permite agrupar los resultados de la consulta. Se traduce a la cláusula GROUP-BY de la SELECT de SQL.
- *xsl_adapt:when*
  Añade una condición a la cláusula GROUP-BY de agrupamiento. Equivale a la cláusula HAVING de SQL.
- *xsl_adapt:sort*
  Permite clasificar la consulta. Equivale a la cláusula ORDER BY de SQL.

Todas estas construcciones tienen diferentes atributos: `select`, `match`, `test`, ..., en los que pueden aparecer expresiones de camino que utilizan la sintaxis de XPath [9], a través de las cuales referenciaremos a tablas y campos de la base de datos.

Además será posible anidar sentencias SELECT, definiendo un "`template`" dentro de otro, es decir, insertando una sentencia "`template`" dentro de un "`apply-template`", pues cada "`template`" equivale a una sentencia SELECT.

A la hora de traducir las consultas anidadas a la correspondiente sentencia de consulta en SQL se pueden dar dos posibilidades. Si el "`apply-template`" aparece aislado, se traducirá a una sentencia SELECT anidada en la cláusula FROM de la sentencia SELECT inicial, es decir, como una subconsulta de SQL. Si la construcción "`apply-template`" tuviera a su vez anidada una construcción "`value-of`", entonces la anidación se traduciría a un "CURSOR" dentro de la sentencia SELECT equivalente. A los resultados asociados a este cursor habría que darles un nombre en el resultado de la consulta en XML para identificarlos. Este nombre estará formado por la unión del nombre de la tabla principal de la consulta anidada y la tabla principal de la sentencia SELECT inicial.

Para entender mejor el modelo de traducción descrito de cualquier consulta a base de datos en el lenguaje XSL Adaptado, a su equivalente sentencia SELECT de

consulta en SQL, a continuación detallamos el Data Type Definiyion (DTD) que contendrá implícitas las reglas de este proceso de traducción.

```
<!ELEMENT xsl_adapt:if ( (xsl_adapt:value-of)+,
(xsl_adapt:apply-templates)?)>
          <!ATTLIST xsl_adapt:if
             test CDATA #REQUIRED>
<!ELEMENT xsl_adapt:apply-templates (xsl_adapt:value-
of)*>
          <!ATTLIST xsl_adapt:apply-templates
             select CDATA #REQUIRED>
<!ELEMENT xsl_adapt:value-of EMPTY>
          <!ATTLIST xsl_adapt:value-of
             select CDATA #REQUIRED>
<!ELEMENT xsl_adapt:for-each ((xsl_adapt:when)?)>
          <!ATTLIST xsl_adapt:for-each
             select CDATA #REQUIRED>
<!ELEMENT xsl_adapt:when EMPTY>
          <!ATTLIST xsl_adapt:when
             test CDATA #REQUIRED>
<!ELEMENT xsl_adapt:sort EMPTY>
          <!ATTLIST xsl_adapt:sort
             select CDATA #REQUIRED>
```

Según este DTD y deduciendo a partir de él las reglas de traducción del lenguaje XSL Adaptado a SQL, por ejemplo la sentencia SQL: "`SELECT nom_prod FROM producto`", en XSL adaptado equivaldría a la siguiente:

```
<?xml versión="1.">
<!DOCTYPE xsl_adapt:stylesheet system "xsl_adapt.dtd">
<xsl_adapt:stylesheet
version = "1.0"
schema_db= "schema_db_produc.xml">
  <xsl_adapt:template match=/">
     <xsl_adapt:apply_template
          select="Producto">
  </xsl_adapt:template>
  <xsl_adapt:template
          match=//Producto">
     <xsl_adapt:value_of
          select="nom_prod">
  </xsl_adapt:template>
</xsl:stylesheet>
```

4.2 XQuery Adaptado

Para adaptar el lenguaje XQuery a la consulta a base de datos, sólo se necesitan las sentencias FOR-LET-WHERE-RETURN (FLWR) de este lenguaje. La equivalencia con las cláusulas de la sentencia SELECT de SQL sería la siguiente:

- FROM: determina las tablas de la base de datos a consultar.
- RETURN: especifica los atributos del resultado de la consulta
- WHERE: añade una condición, cláusula WHERE, a la consulta.
- LET: permite agrupar los resultados de la consulta. Equivale al GROUP-BY de SQL.

Las expresiones de camino con XPath que aparecen en las sentencias FLWR, van a determinar las tablas y campos de la consulta.

Según esto, la sentencia del ejemplo anterior "`SELECT nom_prod FROM producto`", en XQuery Adaptado, tendría la siguiente sintaxis.

```
FOR $p IN document("schema_db_produc.xml")
    RETURN Producto/nom_prod
```

5. Arquitectura Sistema XBD

El sistema XBD implementa dos tareas principales. La primera permitirá adaptar la base de datos a consultar, para que de hecho pueda ser consultada en el sistema XBD. La segunda, una vez adaptada la base de datos, permitirá consultarla utilizando cualquiera de los lenguajes del apartado anterior.

Según esto la arquitectura del sistema tendría dos componentes principales, el de adaptación y el de consulta. Esta arquitectura se representa en la Figura 1. En ella podemos observar que el documento "Esquema XML de BD" se obtiene como resultado del proceso de adaptación y se utiliza como entrada para el subsistema de consulta.

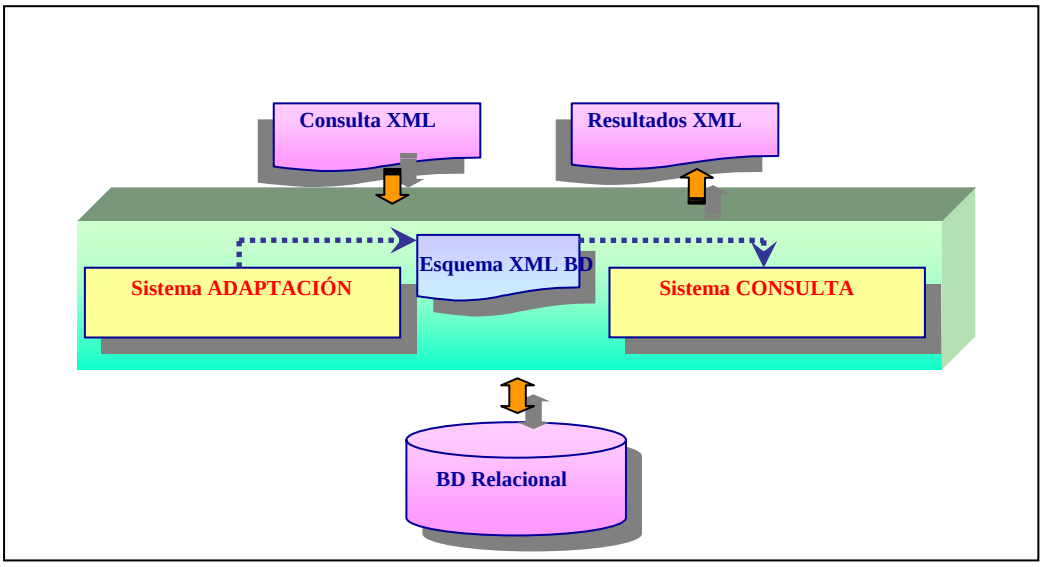


Figura 1. Arquitectura Sistema de Consulta XBD

5.1 Adaptación de una base de datos relacional a XBD

Esta tarea permite que cualquier base de datos relacional sea consultada en el sistema XBD. Para conseguirlo es necesario obtener un documento XML especial denominado *Esquema XML de Base de Datos* (Esquema XML de BD) que contendrá información acerca de la estructura de la base de datos: tablas, relaciones y campos. Este documento se utilizará como entrada del siguiente componente de XBD, el que se encarga de hacer las consultas.

La apariencia del "Esquema XML de BD" es similar a la de un Schema XML clásico [10].

En XML hay dos tipos de esquemas principales, el formato XSD del W3C que es el más utilizado, y el formato XDR de Microsoft que fue el primero. En el sistema XBD se podrá optar por utilizar un formato similar a uno u otro, para crear el documento resultado del proceso de adaptación, el Esquema XML de BD.

Los elementos de estos documentos dependen del formato utilizado. En XDR los principales elementos son "`element`" y "`ElementType`". En XSD, "`SimpleType`", "`element`", "`ComplexType`" y "`sequence`". Estos elementos contendrán información acerca de la base de datos: tablas, relaciones con otras tablas y cardinalidad, campos de cada tabla con sus tipos de datos y si dichos campos son o no clave.

Si usamos el formato XSD, con `SimpleType` definimos cada campo o columna de la base de datos junto al tipo de datos, a través de los atributos "`name`" y "`type`", de estos campos. Cada tabla se describe con un elemento `ComplexType`. Dentro de cada `Complextype` existirá un elemento "`sequence`" a través del que definiremos la relación y cardinalidad de la tabla con el resto de tablas del sistema con las que esta

relacionado, y varios elementos "element" anidados dentro de "sequence" para definir cada uno de los campos que forman la tabla, junto a información sobre si ese campo es clave primaria de dicha tabla, clave secundaria cuando sirve para relacionar a la tabla con otra, o ninguna de las anteriores.

En el formato XDR, con element se define cada campo de cada tabla y con ElementType se define cada tabla. Anidado dentro de cada ElementType, se define el contenido de cada tabla, campos que lo forman y otras tablas con las que se relaciona junto a su cardinalidad.

Para entender mejor la estructura de cualquier Esquema XML de Base de Datos en formato XSD y reflejar lo que acabamos de comentar sobre cómo este esquema describe internamente la estructura de la base de datos, a continuación exponemos el DTD que deberían seguir los documentos de este tipo.

```
<!ELEMENT Schema_bd (conexion, (SimpleType)+,
(ComplexType)+)>
<!ELEMENT conexion EMPTY>
          <!ATTLIST conexion
type (basedatos|docxml) "basedatos"
            owner CDATA #REQUIRED
            cad_conex CDATA #REQUIRED>
<!ELEMENT SimpleType EMPTY>
          <!ATTLIST SimpleType
            name CDATA #REQUIRED
type CDATA #REQUIRED>
<!ELEMENT ComplexType ((sequence)+)>
          <!ATTLIST ComplexType
            name CDATA #REQUIRED>
<!ELEMENT sequence ((element)+)>
          <!ATTLIST sequence
            minOccurs CDATA #IMPLIED
maxOccurs (1|unbounded) "unbounded">
<!ELEMENT element EMPTY>
          <!ATTLIST element
            type CDATA #REQUIRED
            pk CDATA #IMPLIED
            fk CDATA #IMPLIED
            ref CDATA #IMPLIED>
```

De la misma forma exponemos también el DTD que se debería seguir si se utiliza el formato de Esquema XML de Base de Datos más parecido al XDR de Microsoft.

```
<!ELEMENT Schema_bd (conexion, (ElementType)+)>
<!ELEMENT conexion EMPTY>
<!ATTLIST conexion
type (basedatos|docxml) "basedatos"
owner CDATA #REQUIRED
cad_conex CDATA #REQUIRED>
<!ELEMENT ElementType (element)*>
          <!ATTLIST ElementType
            name CDATA #REQUIRED
content (columna|tabla) "columna"
            type CDATA #IMPLIED>
```

```
<!ELEMENT element EMPTY>
          <!ATTLIST element
            type CDATA #REQUIRED
            minOccurs CDATA #IMPLIED
            maxOccurs (1|*) "*"
            pk CDATA #IMPLIED
            fk CDATA #IMPLIED
            ref CDATA #IMPLIED >
```

Finalmente, y para visualizar la utilización de los Esquemas XML de Base de Datos, como ejemplo se da a continuación la definición de las tablas de PRODUCTOS y PEDIDOS. Además se sabe que la tabla de PRODUCTOS guarda una relación de 1 a N con PEDIDOS. Según esto el contenido del Esquema XML de BD en formato XSD para la tabla PRODUCTOS sería el que aparece después de la definición de las tablas.

```
PRODUCTOS (cod_prod, nom_prod, stock_prod)
PEDIDOS (cod_ped, fecha_ped, cod_prod, cant_ped)
```

```
<SympleType name="cod_prod" type="NUMBER">
<SympleType name="nom_prod" type="VARCHAR2">
<SympleType name="stock_prod" type="NUMBER">
<ComplexType name="PRODUCT0S">
 <sequence minOccurs="1"
   maxoccurs="1">
    <element type= "cod_prod"
     pk="true">
    <element type= "nom_prod>
    <element type= "stock_prod>
  </sequence>
  <sequence maxoccurs="unbounded">
    <element type="PEDIDOS">
  </sequence>
</ComplexType>
```

5.2 Consulta a una base de datos relacional en XBD

Esta tarea permite consultar la base de datos, previamente adaptada con la tarea anterior y gracias a la utilización del documento *Esquema XML de BD* obtenido en el proceso anterior.

Las consultas se podrán realizar en cualquiera de los lenguajes de consulta del sistema XBD: XSL o XQuery Adaptado.

Además, el componente que implementa esta tarea tiene una apariencia muy semejante a la consulta a un documento XML, en cuanto a que el interfaz de consulta presenta la información de la base de datos, su estructura, en un formato arborescente similar al árbol DOM de un documento XML.

En esta estructura arborescente cada tabla se representa como un nodo que parte del nodo raíz. Cada rama asociada a cada nodo a su vez contendrá a los campos que forman la tabla y otras tablas con las que se relaciona. Junto a esta estructura

arborescente, el interfaz también suministra ayuda para editar la consulta, con información sobre la sintaxis de los lenguajes de consulta a utilizar.

Por último, además de la opción de edición de las consultas, el componente de consulta también suministra otras dos opciones, las de validación y ejecución de la consulta.

La validación permitirá comprobar si la consulta es correcta tanto desde el punto de vista de los datos consultados en la base de datos: tablas, campos y relaciones, como de la sintaxis de la consulta en relación a los lenguajes de consulta del sistema XBD.

Finalmente, la opción de ejecución nos permitirá ejecutar la consulta sobre la base de datos. Esto supone traducir internamente la consulta a una única sentencia SELECT de SQL, siguiendo las reglas o semántica de las construcciones del lenguaje de consulta, con relación a la equivalencia con las cláusulas de la sentencia SELECT de SQL vistas en el apartado 4 de este trabajo. La consulta SQL obtenida como resultado del proceso de traducción, se ejecutará sobre la base de datos y el propio DBMS que contiene a la base de datos ya dispone de herramientas para convertir los resultados de la base de datos obtenidos de la consulta, a XML. De esta forma, tal como pretendíamos, tanto el lenguaje de consulta como el formato del resultado será XML.

En la figura 2 se muestra la apariencia del interfaz de consulta. En él, tal como hemos comentado, podemos observar ala izquierda el árbol que muestra la estructura de la base de datos, y a la derecha la pantalla en la que editamos la consulta. En la parte inferior podemos utilizar la ayuda que se nos ofrece para la edición de la sintaxis de las construcciones de la consulta, y también podemos observar la ruta seleccionada en el árbol. Esta ruta utiliza la sintaxis de XPath y podemos utilizarla para añadirla como valor de alguno de los atributos de las construcciones de la consulta que se esta editando.

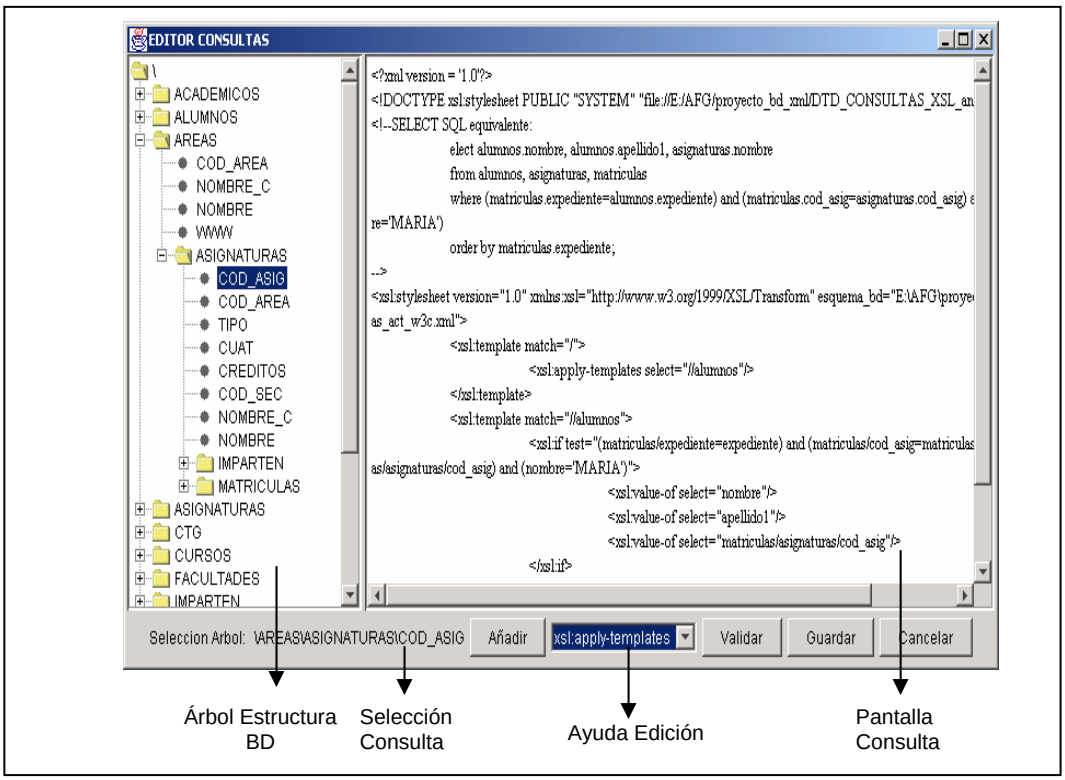


Figura 2. Interfaz de Consulta del Sistema XBD

6. Conclusiones y Líneas Futuras

Se demuestra, una vez descritas las características de nuestro sistema, que se consiguen los objetivos que nos habíamos propuesto. XBD permitirá consultas a base de datos con un formato y lenguajes semejantes a la consulta a documentos XML, para que al usuario le resulte un sistema transparente y considere que está consultando únicamente datos XML, tal como si estuviera en un entorno Web.

Para conseguir este efecto, no se necesita transformar la información de la base de datos a ningún formato, como ocurría en otros sistemas. Además se podrá consultar en cualquier momento cualquier campo de la base de datos, por que en otros sistemas esta consulta estaba restringida a la parte de la base de datos afectada por una consulta inicial.

Por último, el sistema resulta muy eficiente porque quien realiza la consulta final en SQL sobre la base de datos, obtenida tras el proceso detraducción de la consulta

realizada en XSL o XQuery Adaptado, es el propio DBMS que contiene a dich abase de datos y sólo se añade el paso especial de traducción de la consulta a SQL, cuyo tiempo de ejecución es despreciable en comparación con la ejecución de la consulta completa.

Esta eficiencia ha quedado demostrada también al utilizar nuestro sistema dentro de una aplicación de nuestra Facultad cuya tarea es permitir la consulta de las calificaciones de nuestros alumnos y mostrarlas a través de la Web junto a otras informaciones académicas referidas al alumnado que existen ya en formato XML.

Una vez demostradas las bondades de XBD, sin embrago, también se podría mejorar. Nuestras líneas de investigación actuales giran entorno a esta idea. Se trataría de conseguir que en la misma consulta y al mismo tiempo se pudiesen consultar distintas fuentes, diferentes bases de datos y/o documentos XML, y además aprovechando la máxima potencia de cada una. Es decir, en vez de poder obtener sólo sentencias SQL en formato estándar, que también se pudiesen obtener sentencias de consulta que aprovechasen las ampliaciones de este lenguaje SQL propias de cada DBMS, e igual para los sistemas de consulta a documentos XML.

Referencias

1. Fermoso, A, Berjón, R. Acceso a datos relacionales en entornos web. Un caso Práctico. Ed. Universidad Pontificia de Salamanca (2004)
2. Fernández, M., Kadiyska, Y., Morishima, A., Suciu, D., Tan., W.C., SilkRoute: a framework for publishing relational data in XML. ACM Transactions on Database Systems (TODS), Vol. 27, Nª 4, December (2002) pp. 438-493.
3. Funderburk, J. E., Kiernan, G., Shanmugasundaram, J. , Shekita, E., Wei, C., XTABLES: Bridging relational technology and XML. IBM Systems Journal, Vol. 41 , Nª 4 (2002)
4. IBM. IBM DB2 Universal Database. XML Extender Administration and Programming. Version 8. IBM Corporation. (2002)
5. Microsoft, SQL Server 2000. XML and Internet Support. Microsoft Corp. (2002)
6. Oracle, Oracle 9i Release 2. Database Concepts. Oracle Corp., March. (2002)
7. Oracle, Oracle 9i Release 2. XML Database Developers's Guide-Oracle XML DB. Oracle Corp., October 2002.
8. World Wide Web Consortium, Extensible Markup Languaje (XML). http://www.w3c.org/xml. (2004)
9. World Wide Web Consortium, XML Path Language (XPath). http://www.w3c.org/TR/xpath. (2004)
10. World Wide Web Consortium, XML Schema. http://www.w3c.org/2001/XMLSchema (2004)
11. World Wide Web Consortium, Extensible Style Language (XSL). http://www.w3c.org/Style/XSL (2004)
12. World Wide Web Consortium, XQuery: A Query Language for XML. http://www.w3c.org/TR/xquery (2004)

Obtención de datos XML a partir de información almacenada en bases de datos

Roberto Berjón Gallinas[1], Ana M. Fermoso García[1], and María J. Gil Larrea[2]

[1] Universidad Pontificia de Salamanca, Escuela Universitaria de Informática.
Salamanca, España.
{rberjon,afermoso}@upsa.es
[2] Universidad de Deusto, E.S.I.D.E.
Bilbao, España.
marijose@eside.deusto.es

Abstract. XML se ha convertido en el estándar para la presentación de información en la Web y también para su intercambio en los flujos inter o intra empresas. Por ello en éstas resulta cada vez más necesario generar XML a partir de la información que tienen almacenada en sus bases de datos. Los sistemas de gestión de bases de datos(DBMS) empleados siguen siendo en su mayoría relacionales, aunque actualmente también se utilizan otros que sí permiten el almacenamiento y obtención de información en formato XML: los RDBMS habilitados para XML y los DBMS nativos XML. Esta diversidad de fuentes, unido a la consiguiente distribución de la información entre ellas, dificulta la generación de XML.

En este trabajo se presenta detalladamente las herramientas existentes que generan XML a partir de información relacional, así como también las características de los DBMS que permiten el almacenamiento XML. Una vez analizados todos estos sistemas, se planteará una propuesta que trate de solventar las limitaciones que éstos poseen.

1 Introducción

XML se ha convertido en el estándar para la presentación de información en la Web y también para su intercambio en los flujos inter o intra empresas. Por ello en éstas resulta cada vez más necesario generar XML a partir de toda la información que poseen. Esta información generalmente se encuentra distribuida entre varias fuentes de información, como documentos XML, las tradicionales bases de datos relacionales, y también otro tipo de bases de datos que permiten el almacenamiento y obtención de información en formato XML: los RDBMS habilitados para XML y los DBMS nativos XML. Esta diversidad de fuentes, unido al hecho de que la información no se encuentre siempre en el mismo formato, dificulta la generación de XML.

En este trabajo se va a realizar un estudio de las principales fuentes de información de que disponen las empresas y de cómo se puede obtener XML de ellas. Lógicamente la situación ideal sería la de contar con una herramienta que

pudiese consultar simultáneamente diversas fuentes de datos, sean éstas de tipo XML o no, y poder generar XML a partir del resultado de dichas consultas. Para abordar esta problemática, se va a realizar una clasificación de las distintas fuentes de información, así como un estudio de las herramientas existentes que realizan dicha tarea. En virtud de las conclusiones obtenidas, se planteará una nueva propuesta que intente solventar las limitaciones encontradas.

El trabajo se encuentra organizado en tres capítulos, en el primero se analizarán las fuentes de datos no XML y las características de las herramientas existentes que permiten la transformación de sus datos a este formato. En el segundo, se estudiarán las fuentes de datos XML y, finalmente, en el último se presentará la nueva propuesta.

2 Fuentes de datos no XML

La principal fuente de datos no XML con que cuentan las empresas son los RDBMS. Para convertir la información que almacenan a un formato XML se precisa de herramientas externas. Éstas forman una capa intermedia entre las aplicaciones cliente y la base de datos (ver Fig. 1), de tal forma que los clientes envían sus peticiones hacia la herramienta, ésta consulta la base de datos y transforma el resultado obtenido a una representación XML que devuelve como respuesta.

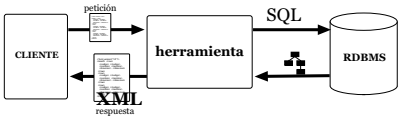


Fig. 1. Esquema gnral. de herramientas que convierten a XML datos de RDBMS

Estas herramientas se pueden clasificar, atendiendo a su patrón de diseño, en dos tipos: las que transforman el modelo relacional a una representación XML y aquéllas que lo que transforman es el resultado de consultas SQL a la base de datos.

2.1 Herramientas que transforman el modelo relacional a XML.

El esquema de funcionamiento de todas es similar al que se muestra en la Fig. 2. Crean de forma virtual una vista XML con información contenida en la base de datos. El hecho de que sea virtual quiere decir que la vista XML nunca llega a materializarse. Al ser esta vista lo único que realmente ve el usuario, éste debe emplear un lenguaje orientado a XML para consultar dicha información. Las herramientas traducen cada consulta a una o varias sentencias SQL que se emplearán para extraer la información necesaria de la base de datos. Posteriormente transforman a XML, siguiendo las indicaciones que el usuario haya definido en su petición, la información relacional obtenida. A continuación se describe las principales herramientas que siguen este modelo.

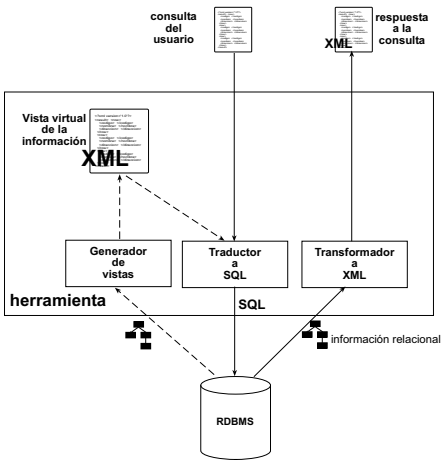


Fig. 2. Esquema de la transformación del modelo relacional a XML

XTABLES: [2](anteriormente conocida como XPERANTO [3]) crea automáticamente una vista canónica[3] denominada *default* que contiene la información de todas las tablas de la base de datos. Esta herramienta también permite crear nuevas vistas a través de consultas XQuery sobre la vista *default* o cualquier otra creada con posterioridad. El usuario final realiza consultas, también en XQuery, sobre el conjunto de vistas definidas.

Adicionalmente, XTABLES también permite la consulta simultánea tanto de datos relacionales como de otros documentos XML. Para ello, previamente se debe registrar dichos documentos XML en la herramienta, lo que implica el almacenamiento de estos documentos en la base de datos relacional. Este registro sigue los mismos conceptos enunciados por Shanmugasundaram en [4]. El proceso es el siguiente: a la herramienta se le debe facilitar el DTD de los documentos que se desee registrar. A partir de él crea un *DTD graph* que refleja la estructura jerárquica indicada por el DTD. Cada nodo de esta estructura representa un elemento, un atributo o un operador. Del *DTD graph* se crea un esquema relacional siguiendo las siguientes pautas: se crea una relación para el elemento raíz, todos los hijos de un elemento se representan como atributos de la

[3] recibe este nombre porque se aplica la misma regla para convertir cualquier tabla relacional a una representación XML. Esta regla determina que para cualquier relación con un esquema $R(A_1, A_2, \ldots, A_n)$ en la que existen las tuplas $(a_{11}, a_{21}, \ldots, a_{n1})$, $\ldots$, $(a_{1k}, a_{2k}, \ldots, a_{nk})$ su representación XML canónica tiene la siguiente estructura:

<R>
<Tuple><A_1>a_{11}</A_1><A_2>a_{21}</A_2>...<A_n>a_{n1}</A_n></Tuple>
...
<Tuple><A_1>a_{1k}</A_1><A_2>a_{2k}</A_2>...<A_n>a_{nk}</A_n></Tuple>
</R>

relación, excepto aquéllos que identifiquen los operadores + ó * ya que el modelo relacional no puede representar atributos con un conjunto de valores. En estos casos se crean otras nuevas relaciones manteniendo mediante restricciones del tipo *FOREIGN KEY* la referencia a su elemento padre.

Todas estas relaciones serán las tablas en donde se almacenen los datos contenidos en los documentos XML que posteriormente se registrarán en la herramienta.

Además de crear las tablas, la herramienta crea una nueva vista que reconstruye a partir de los datos almacenados, los documentos XML registrados. Esta vista, al igual que las otras, también podrá ser consultada por los usuarios finales, con lo que indirectamente se está consiguiendo consultar simultáneamente datos relacionales y datos XML.

El principal inconveniente de este planteamiento es la falta de sincronización entre el documento original y el almacenado en la base de datos. Si el documento original cambia, habría que actualizar también la base de datos, con lo que sería preciso conocer la ubicación exacta de de cada uno de los datos almacenados al ser preciso el uso de sentencias SQL, lo cual hace que la actualización sea una tarea excesivamente compleja.

SilkRoute: Su funcionamiento es similar a XTABLES salvo que esta herramienta sólo define una única vista XML y además no permite el registro de documentos XML en la base de datos. En [5][6] la vista XML se crea empleando el lenguaje RXL y los usuarios la consultan a través del lenguaje XML–QL [8]. Posteriormente en [7] los autores hicieron que XQuery fuese el lenguaje empleado tanto para definir la vista XML[4], como por los usuarios para consultarla.

XBD: [9] Al igual que las otras también define una vista que, a diferencia de las anteriores, muestra de forma arborescente la estructura de la base de datos siguiendo el siguiente planteamiento: si en un esquema relacional existen dos tablas T_1 y T_2 en donde T_2 define una relación *foreing key* respecto a la tabla T_1, la representación XML de cada registro de T_1 contiene además de los campos definidos en la tabla, todos los registros de la tabla T_2 relacionados con él. Los lenguajes que emplean los usuarios para acceder a la vista son *XSL adaptado* y *XQuery adaptado*, similares en apariencia a XSL y XQuery respectivamente, pero donde las expresiones XPath contenidas en ellos pueden, teniendo en cuenta la estructura de la vista, acceder fácilmente a registros de la base de datos relacionados entre sí.

La principal virtud de todas estas herramientas es que emplean un lenguaje de consulta basado en XML para acceder a la información y, por tanto, pueden

[4] Esta vista se denomina *public XML view* y se define a través de una consulta XQuery sobre una vista canónica denominada *canonical View* que automáticamente crea la herramienta y que engloba toda la información de la base de datos.

considerarse como una solución global ya que independientemente que la información resida en un RDBMS o en documentos XML independientes, se emplea siempre el mismo lenguaje de consulta. Sin embargo ésta es también su mayor desventaja ya que, en el caso de que la fuente de información sea una base de datos, dicho lenguaje tiene que ser traducido finalmente a SQL. Durante este proceso de traducción, estas herramientas se orientan únicamente en el modelo relacional de la información impidiendo por tanto hacer uso de las extensiones o facilidades que pueda ofrecer cada RDBMS (nuevos tipos de datos soportados, utilización de funciones estándares o definidas por el usuario) y que exigirían una sintaxis concreta y específica de SQL propia del RDBMS empleado.

Por otra parte, sólo SilkRoute define qué información concreta de la base de datos pueden ver los usuarios, ya que es necesario crear manualmente la vista XML que se les presenta. Esto puede considerarse una ventaja o un inconveniente, ya que la seguridad establecida se aplicará a todos los usuarios. La Tabla 2.1 resume las carácterísticas de estas herramientas.

Herramienta	**Lenguaje de vistas**	**Lenguaje de consulta**	**Se define la estructura del resultado XML**	**Permite consultar XML**	**Permite acotar la información accesible del RDBMS**
XTABLES	XQuery	XQuery	Si	Si	No
SilkRoute	{ RXL XQuery	{ XML-QL XQuery	Si	No	Si
XBD	– –	{ XSL adaptado XQuery adaptado	No	No	No

Table 1. Características de herramientas que transforman el modelo relacional a XML

2.2 Herramientas que transforman el resultado de consultas SQL a XML.

El otro patrón de diseño consiste en la utilización directa de SQL como lenguaje de consulta en las peticiones los clientes. El esquema general de funcionamiento de estas herramientas se representa en la Figura 3. Puede observarse cómo a la herramienta le llegan las peticiones de los clientes, de ellas extrae las sentencias SQL embebidas y las envía a la base de datos, transformando su resultado a XML.

La diferencia fundamental con respecto al anterior tipo de herramientas es que aquí las peticiones de los clientes incluyen las sentencias SQL con las que recuperar la información necesaria de la base de datos. Esta es precisamente su principal ventaja, ya que se puede emplear el SQL nativo del gestor de base de

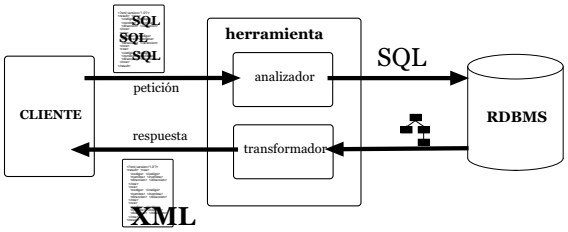


Fig. 3. Esquema general de herramientas que transforman SQL a XML

datos, pudiendo así solventar una de las desventajas analizadas en el anterior modelo.

Las herramimentas transforman los datos relacionales a XML ya sea siguiendo un modelo canónico, o bien según las indicaciones que especifique el usuario en sus peticiones. Según la estructura del XML que se desee obtener, esta transformación puede ser un inconveniente que no se encontraba en las otras herramientas puesto que empleaban lenguajes de consulta orientados a XML. XML es una estructura jerárquica (y como tal puede poseer varios niveles de anidamiento entre sus elementos), sin embargo la información a transformar, es decir, el resultado de consultas SQL, es relacional.

Según se indica en [29] "*En NF^2 las tuplas siguen una estructura jerárquica, lo que resulta útil para representar objetos que por su naturaleza están estructurados jerárquicamente*". Teniendo en cuenta esta afirmación, se puede concluir que la mejor opción para transformar los datos relacionales a XML es que éstos se encuentren siguiendo el modelo relacional anidado NF^2. Este modelo elimina la restricción de la primera forma normal (1NF) del modelo relacional básico (también conocido como modelo relacional plano) según el cual se prohíbe la definición de relaciones en las que estén presentes atributos multivaluados y/o atributos compuestos. Es decir, todos los atributos deben tener un único valor de entre los que define un dominio atómico (un dominio es atómico si sus elementos están considerados como unidades indivisibles[5]).

Por lo tanto, no habría tales problemas si las consultas SQL pudiesen obtener vistas NF^2. Sin embargo, la mayor parte de los RDBMS definen una base de datos como un conjunto de relaciones planas o tablas en primera forma normal. La base de datos Oracle es una excepción ya que permite el uso de UDT (User Defined Types), tablas anadidas, VARRAYS y además también permite obtener vistas NF^2 en las sentencias SQL mediante el uso de CURSORES.

Por lo tanto, las herramientas deberían o bien soportar vistas NF^2, o bien proporcionar la operación NEST[6], o bien facilitar cualquier otra opción que finalmente lo consiga.

[5] Por ello, a este modelo anidado también se le conoce como *modelo relacional no en primera forma normal*, *No-1NF*, *NFNF* ó *NF^2*.

[6] Se ha propuesto extensiones del álgebra relacional [27] [28] [30] y del cálculo relacional que permiten realizar la conversión de relaciones planas a vistas NF^2. Todas

Con respecto a lo expuesto anteriormente, se puede hacer una primera clasificación dentro de las herramientas que transforman SQL a XML atendiendo al criterio de anidamiento. De este modo, este tipo de herramientas se puede agrupar en:

(1) Aquéllas que permiten tratar directamente con vistas NF^2. *Oracle XML SQL Utility (XSU)* y *Oracle XSQL Pages* [10] son herramientas que siguen este patrón.
(2) Aquéllas que permiten convertir una vista plana (que será obtenida a través de la consulta SELECT embebida en la petición del usuario) a una vista NF^2 mediante el operador NEST: ejemplos de este tipo son *XML/SQL* [11] y *uR2X* [12].
(3) Aquéllas que permiten anidamiento definiéndolo en la propia sentencia SELECT: *SQLXML XML for SQL Server 2000* [14].
(4) Aquéllas en las que el anidamiento se simula ejecutando una sentencia SQL por cada una de las tuplas devueltas por otra. Por ejemplo: se ejecuta una sentencia SQL que devuelve la información de todos los alumnos, por cada uno de ellos se ejecuta otra sentencia SQL que devuelve las asignaturas de que está matriculado dicho alumno. Este tipo de herramientas deben permitir la ejecución de más de una sentencia SQL y además admitir que en la definición de las consultas SELECT se puedan incluir parámetros (con el único objeto de no tener que rescribir dichas consultas): *DB2XML* [21], *Net.Data* [22] [23] [22], *ODBC2XML / JDBCXML* [24], *JSP* [25] son herramientas que siguen este patrón.
(5) Aquéllas que no permiten trabajar con vistas NF^2, no tienen definida la operación de anidamiento NEST y tampoco pueden simular el anidamiento descrito anteriormente. Un ejemplo es *Extensiones SAX y DOM para JDBC* [26].

---

coinciden básicamente en la definición de dos operaciones básicas: anidar (NEST) y desanidar (UNNEST). Básicamente NEST, que se denota como: $\nu_{(A_1,A_2,\ldots,A_n)}(r)$, crea una nueva relación r' agrupando las tuplas de r que tienen el mismo valor en todos sus atributos excepto en los indicados por A_i.

Sea r una relación con un esquema $R = (A_1, A_2, \ldots, A_n)$, y $\nu_{(B_1,B_2,\ldots,B_m)}(r)$ la operación NEST sobre la relación r, donde cada B_i es un A_j distinto para algún j. Sea $C_1, C_2, \ldots, C_{n-m}$ las A_j que no son iguales a ningún B_i. Entonces, el resultado de la expresión anterior, es una relación r' basada en un esquema R'. Para calcular este esquema se siguen los siguientes pasos:

1. Se añade la regla $B = (B_1, B_2, \ldots, B_n)$.
2. Se añade la regla $R' = (C_1, C_2, \ldots, C_{n-m}, B)$.
3. Finalmente, la relación r' es el resultado de los siguientes pasos:
   (a) Dividir r en grupos de tuplas que coinciden en $C_1, C_2, \ldots, C_{n-m}$. Estos grupos están identificados por $G_1, G_2, \ldots, G_p$.
   (b) Incluir en r' una tupla t_i por cada grupo G_i, donde t_i toma sus valores de los $C_1, C_2, \ldots, C_{n-m}$ comunes a todas las tuplas en G_i y $t_i[B]$ es el conjunto de valores $(B_1, B_2, \ldots, B_n)$ de las tuplas en G_i.

Dentro de esta clasificación, se podría realizar una segunda atendiendo a otra serie de características enumeradas a continuación:

(a) Si la petición que realiza el usuario a la herramienta es a su vez un documento XML.
(b) Si la estructura del documento respuesta que genera la herramienta no está prefijada por ésta. Es decir, que sea el propio usuario quien, de alguna forma, pueda incluir en la petición la estructura con la que finalmente la herramienta genere el XML resultado, sin necesidad de realizar una transformación XSL posterior.
(c) Si la petición que realiza el usuario no tiene limitado el número de consultas SQL que se pueda incluir.
(d) Si las consultas SQL embebidas en la petición pueden contener parámetros cuyos valores también serán incluidos dentro de la petición.
(e) Si permite la consulta simultánea de distintas fuentes de datos (DBMSs)
(f) Si permite el acceso a UDTs (User Defined Types) y colecciones (VARRAYS o tablas anidadas)

Esta clasificación y subclasificación se resume en la Tabla 2.2:

Herramienta	NF^2					Otras características					
	(1)	(2)	(3)	(4)	(5)	(a)	(b)	(c)	(d)	(e)	(f)
DB2XML				⊙		⊙					
Net.Data				⊙		⊙	⊙	⊙	⊙		
SQLXML			⊙			⊙	⊙	⊙	⊙		
ODBC2XML/JDBC2XML				⊙		⊙	⊙	⊙	⊙		
Oracle XML SQL Utility	⊙								⊙		⊙
Oracle XSQL Pages	⊙					⊙		⊙	⊙	⊙	⊙
SAX and DOM Extensions					⊙						
JSP				⊙		⊙	⊙	⊙	⊙	⊙	
XML/SQL		⊙				⊙	⊙	⊙	⊙	⊙	
uR2X		⊙				⊙	⊙	⊙	⊙	⊙	

Table 2. Características de herramientas que transforman SQL a XML

3 Fuentes de datos XML

En la actualidad las empresas disponen de varios tipos de bases de datos que permiten el almacenamiento y consulta de información XML. Éstas se pueden clasificar en dos grandes grupos que se analizarán a continuación: las bases de datos relacionales habilitadas para XML y las bases de datos nativas XML.

3.1 RDBMS habilitados para XML:

Los RDBMS habilitados para XML (Oracle 9i Release 2 [10], DB2 XML Extender [15] y SQLServer 2005 [13]) son bases de datos tradicionales que definen un nuevo tipo de dato que permite el almacenamiento de información en formato XML. En todos ellos la información XML a almacenar sufre algún tipo de transformación, completamente transparente para el usuario, que en algunos sistemas implica la fragmentación del documento XML a fin de que éste pueda ser almacenado en tablas relacionales que posteriormente podrán ser indexadas y por tanto mejorar el rendimiento de la base de datos durante el proceso de consulta y extracción de este tipo de información. Esta transformación tiene dos planteamientos diferentes.

Uno es el aportado por Oracle y DB2. En ellos el tipo de dato XML (XMLType en el caso de Oracle y XMLVARCHAR en el de DB2) posee una tabla adjunta en donde se almacena la información contenida en el documento XML (posteriormente se puede crear índices en estas tablas para que su acceso sea más eficiente). Esto exige una previa asociación entre el contenido de los elementos y atributos XML con los campos de dicha tabla. En el caso de DB2 esta asociación se realiza a través de un *DAD file* en el que se describe la tabla con sus campos y, a través de expresiones XPath, el origen de esa información dentro del documento. En el caso de Oracle, se debe registrar previamente una versión extendida del XML Schema del documento XML, este proceso crea UDTs por cada complexType contenido en aquél (sus campos almacenarán la información de cada elemento o atributo XML) y finalmente se crea la tabla del UDT correspondiente al *root element* del documento.

El otro planteamiento es el utilizado por SQLServer 2005. En éste, el documento XML se almacena en un formato binario en el que los elementos se identifican a través de un número, que actúa a modo de índice, y la información embebida en el documento XML se convierte previamente al correspondiente tipo de dato en virtud de la naturaleza de dicha información.

El primer planteamiento es mejor, puesto que el usuario decide qué información en concreto desea indexar a fin de optimizar la búsqueda dentro de los datos XML. Además Oracle es quizá el mejor sistema ya que, a diferencia de lo que ocurre en DB2, no se produce una redundancia en la información almacenada (en DB2 además de almacenar la información fragmentada en tablas, también guarda el original y aunque mantiene ambas informaciones sincronizadas, esto redunda en emplear un mayor espacio de almacenamiento y un menor rendimiento en las actualizaciones).

En cuanto al lenguaje, embebido en las sentencias SQL, utilizado para consultar la información XML, sin duda SQLServer 2005 es el más potente. Esto se debe a que utiliza XQuery, en lugar de XPath como sucede en los otros sistemas.

En todos, además de permitir el almacenamiento XML, también se puede obtener este mismo formato a partir de su información puramente relacional. En este sentido, Oracle es quien mejor implementa esta característica ya que ofrece un conjunto de funciones, que se incluirán en la sentencia SELECT, encargadas de definir la estructura del documento XML de salida. En los otros sistemas, la

estructura del XML obtenido depende por completo de la estructura de la sentencia SQL (orden en que se seleccionen las tablas y los campos en la consulta). Se observa, por tanto, que no sólo la sintaxis de las sentencias SELECT es completamente diferente y particular en cada sistema (no siendo portables entre las distintas bases de datos) sino que también su formulación es muy compleja de definir.

La Tabla 3.1 muestra de forma resumida las principales características de los RDBMS analizados.

RDBMS	**Tipo de dato**	**Se emplea el lenguaje XML**	**La generación de XML a partir de datos relacionales se realiza empleando**
Oracle 9i R2	XMLType	XPath	Las funciones { XMLElement, XMLAttributes, XMLForest, XMLAgg, XMLConcat
SQLServer 2005	XML	XQuery	`SELECT ...` `FROM ...` `WHERE ...` `FOR XML, TYPE;`
DB2 XML Extender	XMLVarchar XMLCLOB XMLFile	XPath	{ *Compose XML*, *DAD file*

Table 3. Características de los RDBMS habilitados para XML

3.2 Bases de datos nativas XML

Permiten el almacenamiento, consulta y actualización de información XML. La diferencia fundamental entre ellas (Tamino [16], X-Hive/DB [17], dbXML [18], eXist [19], Xindice [20]) es el lenguaje empleado tanto para la consulta como para la actualización de la información almacenada. Se observa que en todos se emplea XQuery, XPath o ambos lenguajes para consultar la información. Sin duda en los sistemas donde se utilice XQuery podrá plantearse consultas mucho más complejas que en aquéllos que empleen XPath, al ser aquél un lenguaje más potente y versátil. En cuanto al lenguaje empleado para las actualizaciones, todavía no hay uno estandarizado aunque de momento XUpdate [1] es el más extendido. Sin embargo, el hecho de emplear dos lenguajes diferentes, uno para la consulta (XQuery) y otro para la actualización de la información (XUpdate) no parece la solución más acertada, teniendo en cuenta además que el formato del segundo es XML y el del primero no. De ahí que sea preferible utilizar, como sucede en la base de datos Tamino, una extensión no estandarizada de XQuery para realizar las actualizaciones.

En la Tabla 4 se muestra una comparativa entre los lenguajes de consulta y actualización que utiliza cada base de datos.

Gestor de base de datos	Lenguaje de consulta	Lenguaje de actualización
Tamino	XQuery, X–Query (XPath extendido)	extensión de XQuery
X-Hive/DB	XQuery, XPath	XUpdate
dbXML	XPath	XUpdate
eXist	XQuery, XPath	XUpdate
Xindice	XPath	XUpdate

Table 4. Lenguajes empleados por bases de datos nativas XML

4 Conclusiones y una nueva propuesta

Como puede observarse, no existe una herramienta que permita consultar simultáneamente fuentes de datos XML y no XML. En la actualidad, la única forma de conseguirlo sería emplear una única fuente que permitiese tanto el almacenamiento relacional como el XML. Sin embargo, la posible migración de la información podría ser una tarea mucho más costosa que emplear otro tipo de solución.

Para solucionar estos inconvenientes, se propone una herramienta, en la que estamos investigando, que permita realizar consultas a cada una de las fuentes de información analizadas, empleando para ello el lenguaje nativo de cada una. Estas consultas podrían estar o no parametrizadas. Debería también poder transformar a una representación XML el resultado de consultas a bases de datos relacionales permitiendo lógicamente las vistas NF^2. Al mismo tiempo y puesto que el intercambio de información exige que ésta cumpla con una determinada estructura o DTD, también sería conveniente que la herramienta permitiese al usuario especificar la estructura del XML resultante a fin de no tener que realizar transformaciones posteriores. Sería conveniente finalmente, que las peticiones que formulase el usuario a la herramienta estuviesen en formato XML, con el objetivo de poder implementar ésta como un servicio web y, de esta forma, conseguir que fuese el único punto de información de la empresa.

References

1. Laux A. and Martin L.: XUpdate Working Draft. Available at http://exist-db.org/xmldb/xupdate/xupdate-wd.html. (2000)
2. J. E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, C. Wei: XTABLES: Bridging relational technology and XML. IBM Systems Journal. (2002)
3. M. J. Carey, D. Florescu, Z.G. Ives, Y. Lu, J. Shanmugasundaram, E.J. Shekita, S. Subramanian: XPERANTO: Publishing Object-Relational Data as XML. IBM Research Report. (2001)
4. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D.J. DeWitt, J.F. Naughton: Relational Databases for Querying XML Documents: Limitations and Opportunities. The VLDB Journal. (2001) 302–314
5. M. Fernández, W. Tan, D. Suciu: Silkroute: Trading between relations and XML. Proceedings of the Ninth International World Wide Web Conference. (2000)
6. M. Fernández, A. Morishima, D. Suciu, W.C. Tan: Publising relational data in XML: the SilkRoute approach. IEEE Data Engineering. (2001)

7. M. Fernández, Y. Kadiyska, A. Morishima, D. Suciu, W.C. Tan: SilkRoute: a framework for publising relational data in XML. ACM Transactions on Database Systems (TODS), 27(4) (2002)
8. A. Deutsch, M. Fernández, D. Florescu, A. Levy, D. Suciu: XML-QL: A Query Language for XML. In Proceedings of WWW The Query Language Workshop (QL) (1998)
9. A. Fermoso: XBD: Sistema de consulta basado en XML a bases de datos relacionales. PhD thesis, Facultad de Ingeniería E.S.I.D.E. Universidad de Deusto. (2003)
10. Oracle: Oracle 9i Release 2. XML Database Developers's Guide - Oracle XML DB. Oracle Corp. (2002)
11. C.M. Vittory, C.F. Dorneles, C.A. Heuser: Creating XML documents from relational data sources. In Proceedings of EC-WEB (Electronic Commerce and Web Technologies) (2001)
12. V. Braganholo: Updating Relational Databases through XML Views. Instituto de Informática. Univerdidade Federal Do Rio Grande do Sul (2002)
13. S. Pal, M. Fussell, I. Dolobowsky: XML Support in Microsoft SQL Server 2005. MSDN Library. Available at http://msdn.microsoft.com/xml/default.aspx?pull=/library/en-us/dnsql90/html/sql25xmlbp.asp (2004)
14. A. Conrad: A survey of Microsoft SQL Server 2000 XML Features. MSDN Library (2001)
15. IBM: IBM DB2 Universal Database. XML Extender Administration and Programming. Version 8. IBM Corp. (2002)
16. Software AG: Introducing Tamino. Tamino version 4.1.4. Software AG. (2003)
17. X-Hive Corporation: X-Hive/DB. Available at http://www.x-hive.com (2004)
18. The dbXML Group: dbXML. Available at http://www.dbxml.com/index.html (2004)
19. W. Meier: eXist. Available at http://exist.sourceforge.net (2004)
20. Apache Software Foundation: Xindice. Available at http://xml.apache.org/xindice/ (2004)
21. V. Turau: Making Legacy Data Accessible for XML Applications. Available at http://www.ti5.tu-harburg.de/Staff/Turau/pubs/legacy.pdf (1999)
22. J. Cheng and J. Xu: IBM DB2 XML Extender: an end-to-end solution for storing and retrieving XML documents. In Proceedings of ICDE'00 (2000)
23. IBM: IBM Net.Data for OS/2 Windows NT, and UNIX Administration and Programing Guide Version 7. IBM Corp. Available at http://www.ibm.com/software/netdata
24. Intelligent Systems Research: Merging ODBC Data into XML ODBC2XML. Available at http://www.intsysr.com/odbc2xml.htm (2003)
25. Apache group: Jakarta Proyect: DBTags Tag library. Available at http://jakarta.apache.org/taglibs/doc/dbtags-doc/index.html (2003)
26. R. Laddad: XML APIs for databases: blend the power of XML and databases using custom SAX and DOM APIs. Java World (2000)
27. M.A. Roth, H.F. Korth, A. Silberschatz: Extended algebra and calculus for nested relational databases. ACM Trans. Database Syst., 13(4):389–417 (1988)
28. P.C. Fischer, D. Van Gucht: Weak multivalued dependencies. In Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems (1984)
29. R. Elmasri, S. Navathe: Fundamentals of database systems. Addison Wesley (2002)
30. A. Silberschatz, H. Korth, S. Sudarshan: Database System Concepts. McGraw-Hill (1998)

O repositório XML do SIGDiC

João Paulo Rodrigues[1], José Correia[1]

INESC Porto, R. Dr. Roberto Frias, 4200-465 Porto
http://www.inescporto.pt/
{jpr,jcorreia}@inescporto.pt

Resumo O projecto SIGDiC pretende implementar um sistema integrado de gestão e difusão da conteúdos textuais (por exemplo, notícias ou SMS), constituindo o repositório o elemento central deste sistema.
A solução escolhida para o armazenamento dos conteúdos procura conciliar as vantagens das bases de dados relacionais, com a estruturação associada ao XML.
Sobre cada conteúdo foi possível definir informação adicional de metadata, utilizando o conjunto de elementos básicos da Dublin Core Metadata initiative.
O ambiente de edição é gerado, dinamicamente, a partir da informação definida no XML Schema que descreve o conteúdo. A partir deste ficheiro é possível inferir a maioria das regras de negócio e gerar, dinamicamente, um ambiente de trabalho Web que, através de formulários, permite a edição da informação.

1 Introdução

O trabalho aqui apresentado foi realizado no âmbito de um projecto financiado pelo PRIME - Sistema Integrado de Gestão e Difusão de Conteúdos (SIGDiC)[8] - cujo objectivo é desenvolver um sistema capaz de integrar diferentes fontes de conteúdos e disponibilizá-los, automaticamente ou semi-automaticamente, em diferentes canais de comunicação (SMS, Teletexto, WAP, Web, etc.).

Tendo em vista a definição de requisitos, o projecto tem como parceiro uma entidade líder em Portugal na área da produção, edição, gestão e difusão de conteúdos - a RTP. Contudo, o SIGDiC foi concebido tendo em mente as necessidades da generalidade das organizações que desenvolvem o seu "core-business"nesta área e, deste modo, são potenciais utilizadores dos resultados do projecto, todas as empresas nacionais e internacionais que produzem, editam e geram conteúdos para os sectores do audiovisual, Internet e telecomunicações.

O SIGDiC irá ter uma arquitectura modular, baseada em sistemas abertos, e está a ser desenvolvido recorrendo, tanto quanto possível, a ferramentas de domínio público. É neste contexto que surge a opção pelo XML [11], como forma de anotar e estruturar conteúdos, armazenados no repositório.

Em termos da arquitectura SIGDiC, o repositório assume uma função centralizadora da informação a disseminar pelos vários módulos. [1]

[1] Por altura da elaboração deste artigo, está desenvolvida apenas a primeira versão.

O objectivo da versão actual do repositório é suportar os serviços mínimos, que permitam o desenvolvimento, em paralelo, da restante arquitectura, bem como, a validação e teste das funcionalidades já implementadas.

2 Noção de Conteúdo

Antes de avançarmos mais, é oportuno definir a noção de conteúdo, tal como é utilizada no âmbito do projecto SIGDiC.

O que é um "conteúdo"para o SIGDiC?

É óbvio que uma notícia de imprensa é um conteúdo, assim como uma imagem, um vídeo, ou um ficheiro de audio poderão ser um conteúdo. Porém, mais importante é perceber a relação que têm entre si, isto é, saber que conteúdos existem relacionados com um determinado acontecimento ou facto.

Esta é a questão de fundo que se pretende tratar com um sistema como o SIGDiC. A noção de conteúdo abrange todos os diferentes conteúdos que se referem a um mesmo evento, acontecimento ou facto que é identificado como noticia ou informação.

O conteúdo relacionado com um acontecimento poderá incluir uma notícia extensa (para ser disponibilizada via Web), uma mensagem SMS (que será enviada a todos os utilizadores que subscreveram o serviço de notícias SMS), o teleponto que será lido pelo pivot durante o telejornal, a referência para um conjunto de ficheiros com imagens ou a referência para um ficheiro de vídeo com a reportagem. Assim, um conteúdo poderá ser constituído por vários tipos de conteúdos que documentam um dado evento, ou facto.

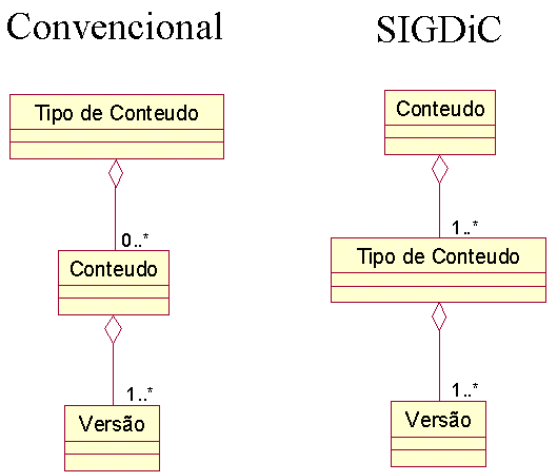


Figura 1. Noção de Conteúdo

No âmbito do projecto SIGDiC, a notícia, a versão WAP, o SMS e a informação do teleponto, são todos tipos de conteúdos que, na sua forma agregada e compilada, constituem o conteúdo SIGDiC.

Os sistemas de gestão de conteúdos convencionais apostam na separação dos vários tipos de conteúdos, de acordo com o meio de difusão.

A noção de conteúdo adoptada no SIGDiC tem por objectivo promover a reutilização de conteúdos, pelo que a informação relativa a uma noticia - o conteúdo - reúne todas as diferentes versões, destinadas a ser difundidas nos vários meios - os tipos de conteúdos.

O objectivo do projecto SIGDiC visa tratar e manter conteúdos baseados em texto. Deste modo, os objectos multimédia, como imagens, vídeo ou áudio, são armazenados mas não são alvo de qualquer tratamento ou processamento, embora seja possível a definição de metadata associada a estes objectos.

3 O modelo de dados

A necessidade de armazenar conteúdos heterogéneos, sem uma estrutura bem definida, requer cuidados especiais na concepção do modelo de dados.

A abordagem inicial perspectivava um modelo de dados algo complexo, onde toda a informação teria que ser armazenada segundo uma estrutura rígida. Ora, a diversidade de conteúdos que o projecto pretende tratar, fazia com que esta abordagem dificultasse sobremaneira o desenvolvimento aplicacional, isto porque, o modelo de dados estaria em constante mutação, de forma a integrar novos tipos de conteúdos, ir de encontro a refinamentos que se mostrassem necessários, ou a englobar novos requisitos.

À medida que fomos percebendo melhor a realidade em questão, tornou-se claro que tínhamos de optar por um modelo de dados muito mais flexível, que permitisse armazenar conteúdos de diferentes características. Veja-se, por exemplo, dois tipos de conteúdos que o sistema poderá tratar e armazenar: a informação de temperaturas nas capitais de distrito e mensagens SMS. Enquanto uma mensagem SMS pode ser vista como uma porção de texto, com uma extensão limitada, a informação das temperaturas é, provavelmente, definida como uma tabela, de uma forma perfeitamente estruturada.

A resposta para esta problemática foi encontrada no XML.

Recorrendo ao XML, foi possível definir uma forma de armazenar informação muito mais flexível que a proporcionada por um modelo relacional. Por outro lado, a estrutura definida pode ser facilmente estendida, indo de encontro a tecnologias emergentes, a novos tipos de conteúdos, ou a necessidades futuras das áreas já abordadas, que o futuro venha a ditar.

Mas, a estruturação da informação em XML apresenta outras vantagens.

Assim, dentro de uma filosofia que privilegia sistemas abertos, a adopção de XML como tecnologia de suporte permite que outras pessoas, ou entidades, a partir dos resultados do projecto, facilmente implementem novas extensões e funcionalidades sobre a plataforma SIGDiC.

Outra vantagem da adopção do XML é a possibilidade de realizar processamento sobre a informação do conteúdo. Recorrendo a transformações XSLT [10] é possível, por exemplo, exportar conteúdos sob a forma de ficheiro Word, ou PostScript/PDF, bem como, gerar páginas Web ou WAP com base em partes do conteúdo original.

A flexibilidade proporcionada pelo XML não deve (não pode!) comprometer a integridade da informação. Deste modo, a informação armazenada é validada por XML Schemas. A utilização de XML Schemas é uma mais valia, quando é considerada a possibilidade de novas funcionalidades serem desenvolvidas directamente por quem delas necessita. Através de XML Schemas, é possível estender, ou refinar, a estrutura e a semântica de uma parte do documento. O próprio XML Schema constitui um documento que, de uma forma clara e inequívoca, documenta as regras que devem ser verificadas, sem necessidade de informação adicional.

Independentemente de estar escolhida uma tecnologia de suporte, resta a necessidade de armazenar a informação.

A escolha da tecnologia de armazenamento tenta conciliar as vantagens das bases de dados relacionais, com a estruturação associada ao XML. Assim, a solução adoptada utiliza uma base de dados relacional, para permitir a optimização do processo de indexação e pesquisa, e o XML para albergar o grosso da informação de conteúdos.

A informação armazenada pela base de dados relacional é, basicamente, informação de metadata associada ao conteúdo e informação acessória para operação da aplicação de gestão. Contudo, é também armazenada alguma informação, de forma redundante com a definida em XML, com o objectivo de acelerar processos de indexação e pesquisa.

No que diz respeito ao armazenamento da informação em XML, foram analisadas duas alternativas: 1) gravar, fisicamente, a informação XML como um documento, mantendo uma referência para o ficheiro; 2) armazenar o documento XML na própria base de dados relacional, num campo específico, do tipo CLOB. A escolha recaiu sobre a segunda opção, por permitir armazenar informação de forma centralizada. Porém, a primeira opção já é considerada uma opção fiável, pelo que, se for verificada uma degradação de performance, é possível, em qualquer altura, a migração de uma solução para a outra, de forma simples e rápida.

4 A estrutura do documento XML

Para que seja possível armazenar em XML toda a informação relacionada com um conteúdo, é necessário que seja definida uma estruturação básica de suporte, que permita o armazenamento dos vários tipos de conteúdos que se pretendem tratar.

Tal como foi referido anteriormente, cada conteúdo poderá incluir vários tipos de conteúdos, com estruturas de dados próprias.

Deste modo, um documento XML que define um conteúdo é constituído por um único nó raiz, designado de "content", e como seus descendentes são definidos os vários tipos de conteúdos, bem como, a informação adicional de metadata.

Como atributo do nó raiz deverá ser obrigatoriamente incluído o código do conteúdo, que funciona como identificador único do mesmo em toda a extensão da arquitectura SIGDiC. O código do conteúdo deve, também, ser representado de forma redundante na informação de metadata.

Sob o nó raiz do documento deverão ser definidos, como filhos, os vários tipos de conteúdos. Cada um poderá ser definido de uma forma totalmente independente dos restantes. Assim, para cada filho deverá ser definida uma estrutura de dados adequada e o corresponde XML Schema, que valide a estrutura e o conteúdo da informação introduzida. É desta forma possível definir tipos de conteúdos novos, bem como, realizar alterações, ou correcções, sobre os já definidos.

O tratamento de conteúdos multimédia, como imagens, vídeo ou áudio, está fora do âmbito do projecto submetido ao PRIME, mas é necessário poder referenciá-los no sistema de informação. Assim sendo, os objectos multimédia são considerados como informação acessória, sendo os ficheiros carregados para o servidor, transparentemente, sem alterações ou processamento.

A referenciação dos objectos multimédia é efectuada no próprio documento XML, sob o elemento "media", sendo definidos vários elementos "assets", que representam e referenciam todos os objectos carregados para o servidor.

O projecto prevê, também, a ligação entre conteúdos que, de alguma forma, estejam relacionados. O estabelecimento de referências cruzadas entre conteúdos é feito através do seu código identificador, sendo suportado pelo elemento "xreference". Estes relacionamentos são bidireccionais, o que implica que se no conteúdo A for referenciado o conteúdo B é, automaticamente, criada em B uma referência para A.

Para além dos conteúdos propriamente ditos, é também necessário incluir informação adicional de metadata, com o objectivo de permitir a classificação, pesquisa e indexação dos conteúdos. Assim, para cada conteúdo e tipo de conteúdo, é incluída informação adicional de metadata, referente à autoria e criação do conteúdo.

A metadata identificada para versão actual do repositório, inclui: utilizador que criou o conteúdo, data de criação, utilizador responsável pela última alteração, data da última alteração, título e resumo.

A opção adoptada para a representação da metadata, recaiu sobre o conjunto de elementos Dublin Core V1.1[3], os quais permitem cobrir as necessidades de metadata actualmente definidas para o projecto. Porém, esta opção não limita que a informação de metadata seja estendida, incorporando, por exemplo, elementos qualificados do Dublin Core. Adicionalmente, por ser XML, a informação de metadata pode ser incluída no conteúdo e não como um conjunto de referências externas.

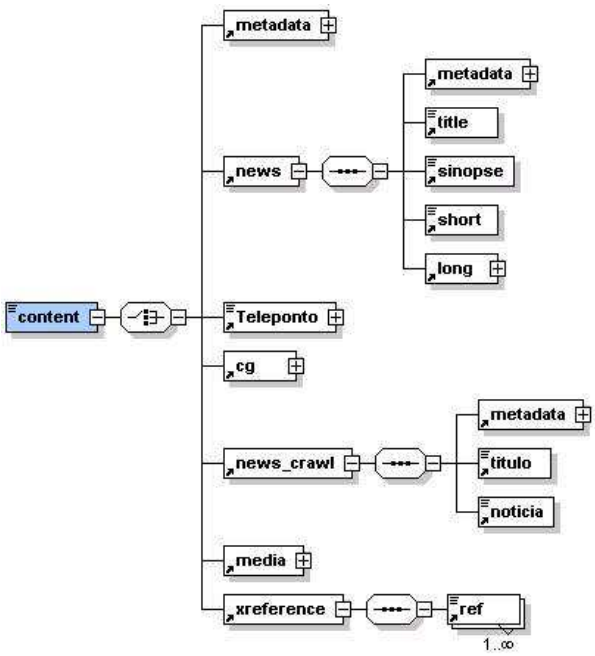


Figura 2. Exemplo de estrutura do documento

A aplicação do conjunto de elementos Dublin Core V1.1, foi realizada segundo as recomendações [4] do Dublin Core Metadata Initiative, para documentos estruturados XML, tendo todas as recomendações referidas sido implementadas.

A metadata incluída no documento XML, deve abranger, quer informação geral de metadata do conteúdo, quer informação específica de metadata directamente relacionada com cada um dos tipos de elementos. Adicionalmente, poderá ainda existir informação de metadata associada a cada referência para objectos multimédia. Desta forma, quer o elemento raiz do documento, quer qualquer dos seus filhos, representando os diversos tipos de conteúdos, deverão incluir o elemento Metadata, com o conjunto básico de elementos Dublin Core V1.1.

Qualquer gestor de conteúdos ficaria incompleto sem um sistema de controlo de versões. Neste domínio, a opção tomada foi a de implementar a funcionalidade de controlo de versões directamente em XML, em que, associado a cada instância do tipo de conteúdo, existe um atributo que identifica a versão.

O processo adoptado é o seguinte: na criação de uma nova instância de cada tipo de conteúdo, o número da versão é inicializado com o valor 1; posteriormente, qualquer alteração de algum tipo de conteúdo origina a criação no documento XML, de um novo elemento correspondente ao tipo de conteúdo alterado, com o número da versão incrementado.

Este processo faz com que seja sempre possível a recuperação de uma versão mais antiga. Por outro lado, é possível manter o controlo de versões ao nível de cada tipo de conteúdo e não ao nível macro do conteúdo.

Uma ferramenta como o SIGDiC deverá possibilitar que a difusão de alguns tipos de conteúdos esteja condicionada por aprovação editorial de um responsável (editor). Esta funcionalidade de aprovação, poderá ser obtida através da adição de um elemento específico "aprovedBy", o qual, sempre que esteja presente, confirma a existência de aprovação. O conteúdo deverá ser preenchido com o código do utilizador que aprovou ou, caso se mostre necessário constituir prova, a assinatura digital, obtida a partir do certificado digital.

5 O ambiente de edição

Com o objectivo de alargar a gama de utilização do SIGDiC a todos os sistemas operativos e, principalmente, permitir o acesso a partir de qualquer ponto, interno ou externo, sem necessidade de software específico, todo o sistema foi desenvolvido de forma a ser suportado por tecnologias Web.

Utilizando a tecnologia disponível pelo pacote DB[6] do PEAR - PHP Extension and Application Repository[5], foi possível criar a camada de isolamento, entre a camada aplicacional desenvolvida sobre PHP[7], ao mesmo tempo que era mantida a performance pela utilização de funções nativas de PHP, para acesso à base de dados.

Adicionalmente, é necessário integrar no PHP o suporte para XML, mais propriamente, o suporte da tecnologia XML DOM[1] e transformações XSLT[12][10].

Para permitir uma melhor organização dos conteúdos, o repositório obedece a uma estrutura em árvore, a qual é suportada por uma base de dados relacional.

O ecrã que representa a organização do repositório tem um layout semelhante ao de muitas outras aplicações de gestão de ficheiros (por exemplo, Windows Explorer), ou seja, do lado esquerdo é representada a árvore que corresponde à hierarquia de pastas, enquanto que do lado direito se pode visualizar os conteúdos disponíveis na pasta activa (título do conteúdo e alguma informação de metadata).

Evidentemente, nesta árvore é possível realizar operações básicas, como criar, alterar ou apagar pastas.

No repositório existe uma pasta que desempenha uma função semelhante ao "Recycle Bin"dos sistemas operativos, isto é, permite que conteúdos e pastas sejam apagados, mas que possam ser recuperados, visto não serem fisicamente apagados.

A questão realmente complexa no que concerne ao repositório é a edição de conteúdos. Dado que o repositório foi modelizado de forma a ser flexível e extensível, e em virtude de não existir uma estrutura predefinida, como se poderá criar uma interface de gestão sobre informação tão mutável e heterogénea?

A solução adoptada passa pela geração dinâmica dos formulários de edição, a partir da informação do XML Schema.

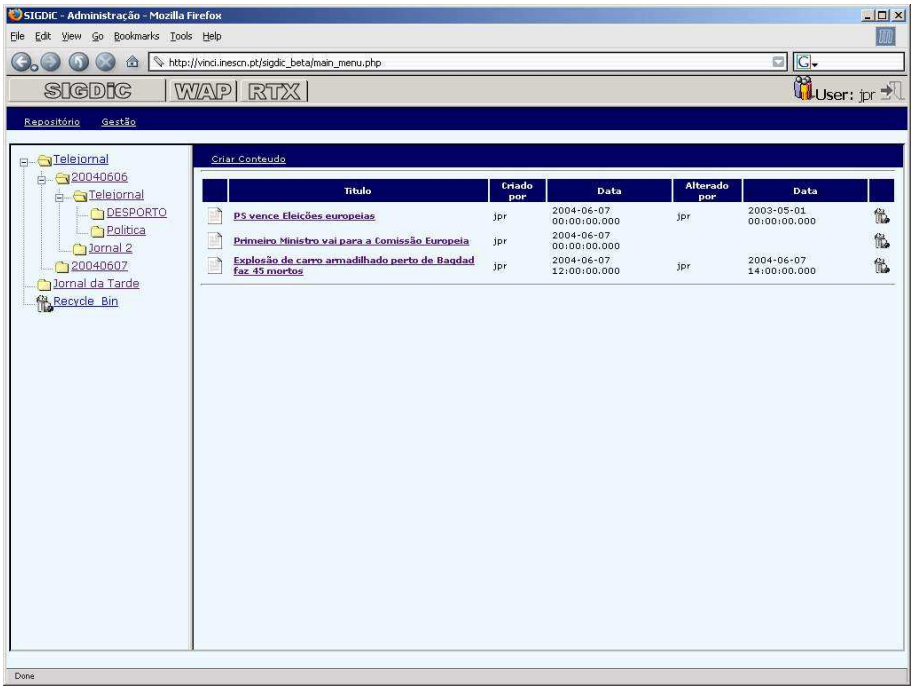


Figura 3. Repositório

Uma das funcionalidades implementadas no SIGDiC é a possibilidade de editar qualquer tipo de conteúdo. Ou seja, além dos tipos de conteúdos já definidos, que constituem um conteúdo, é possível definir novos tipos, de forma a poder incluir novos meios de difusão.

Estruturalmente, qualquer novo tipo de conteúdo tem que ter um ficheiro XML Schema e cumprir três regras:

- O elemento que define o novo tipo de conteúdo, tem que ser representado na árvore do documento XML como filho directo do elemento "content".
- O elemento que define um tipo de conteúdo, tem que ter como filho um elemento do tipo "metadata", para incluir meta-informação de acordo com as recomendações Dubin Core.
- O elemento que define um tipo de conteúdo, tem que ter um atributo "version", para permitir a implementação do sistema de controlo de versões.

A primeira regra é de caracter funcional, pois permite, no XML Schema, identificar todas as partes constituintes de um conteúdo, através da definição dos descendentes do elemento "content".

A segunda e terceira regra são de carácter mais operacional, pois garantem o suporte para as funcionalidades de metadata e controlo de versões do repositório.

A edição de um conteúdo pode implicar editar um, ou mais, tipos de conteúdos. A figura seguinte mostra um ecrã de edição de um conteúdo exemplo, com instâncias definidas para quatro tipos distintos de conteúdos.

O ecrã inclui o título do conteúdo no cabeçalho e informação de metadata, que inclui: o criador do conteúdo e a data de criação, e o responsável pela última alteração em qualquer instância de qualquer tipo de conteúdo.

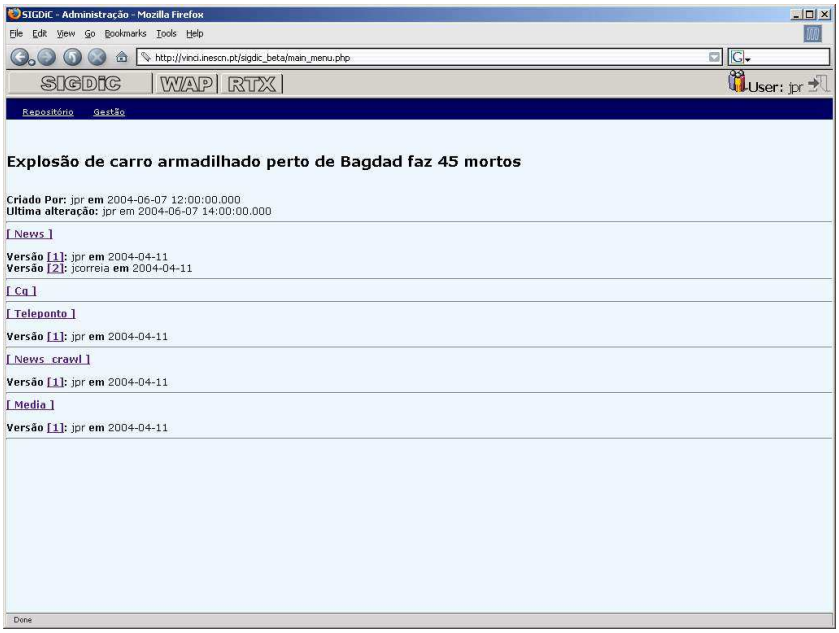


Figura 4. Tipos de conteúdos

No corpo do ecrã surge cada um dos tipos de conteúdos, já definidos, com a informação associada das versões já criadas. Para cada tipo de conteúdo está definida uma área específica.

Adicionalmente, surge também a informação de referências para objectos multimédia e a informação que relaciona, de forma cruzada, o conteúdo com outros existentes.

A acção sobre a designação, ou número da versão, de um qualquer tipo de conteúdo, origina a expansão da área associada, de forma a apresentar o formulário de edição.

Conforme já foi referido, o XML Schema que define o documento XML é usado para, dinamicamente, construir a interface de edição. Ora, dado que o schema não inclui normalmente informação que permita limitar o tamanho de cada elemento, não é possível criar um formulário perfeitamente adaptado à informação a carregar.

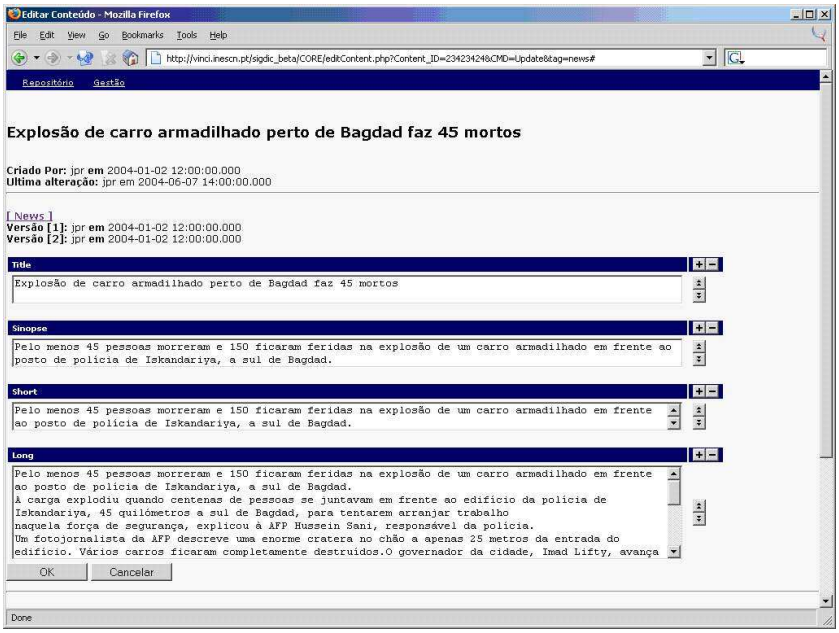


Figura 5. Edição de conteúdo

A solução passa por tratar todos os elementos de um tipo de conteúdo da mesma forma, permitindo situações "esquesitas"como, por exemplo, que o título seja mais extenso que a própria notícia. Não havendo a noção do comprimento máximo, ou pelo menos médio, para a informação de um dado elemento, a solução passou por definir uma dimensão, por defeito, que permita uma utilização confortável para a maioria dos elementos. Nos casos em que a dimensão da caixa de texto não permita uma edição confortável, sem que seja necessário recorrer frequentemente a deslocações verticais do texto, é possível ajustar a dimensão da caixa de edição, através de dois botões dispostos do lado direito. Nos casos de edição de instâncias de tipos de conteúdos já existentes, a dimensão da caixa de texto é automaticamente ajustada de acordo com a informação existente.

Outra questão importante prende-se com a possibilidade de repetição de elementos. Para certos conteúdos é possível a existência repetida de mais do que um elemento. Se permitido pelo XML Schema, para esse elemento será mostrado um conjunto de botões adicionais, que permitem criar mais um elemento no documento.

Para permitir uma melhor adequação da interface de edição, é possível, para casos particulares de conteúdos, definir uma interface de edição específica. Caso esta exista, a interface específica substitui a standard. Esta interface define apenas um front-end de edição, já que todo o processamento de informação é assegurado pelo processo standard.

6 Conclusões

Apesar de se tratar de um projecto em desenvolvimento, é já possível retirar algumas conclusões acerca da adequação de algumas das opções efectuadas.

A estruturação dos conteúdos no repositório, recorrendo a XML, permitiu garantir a adequação futura do sistema, ao permitir adicionar novos tipos de conteúdos, sem a necessidade de desenvolver uma nova versão do repositório.

A opção por uma solução de armazenamento híbrida, composta por uma base de dados relacional e documentos XML, não representou, até ao momento, qualquer problema.

A opção por desenvolver toda a aplicação de gestão numa arquitectura Web de três camadas, tem vantagens operacionais, mas apresenta alguns problemas de performance e implementação. Do ponto de vista operacional, apresenta vantagens no que diz respeito a custos de licenciamento de aplicações e manutenção do parque informático. Porém, implicou um custo e tempo de desenvolvimento superior e uma menor usabilidade, quando comparada com uma aplicação em ambiente desktop equivalente.

Foi possível garantir, através da tecnologia PEAR DB, o isolamento entre o sistema de base de dados e a camada aplicacional, não tendo sido notada qualquer degradação de performance face à utilização de métodos nativos de acesso à base de dados.

A estruturação da informação em XML no repositório, e a possibilidade de definição de novos tipos de conteúdos, impede a criação de um ambiente aplicacional perfeitamente adequado à informação a tratar. Apesar deste aspecto, conseguiu-se uma usabilidade muito próxima da normalmente obtida em aplicações Web, especialmente desenhadas para lidar com informação perfeitamente estruturada.

Foi possível desenhar uma aplicação para lidar com os tipos de conteúdos já identificados, que permite tratar, virtualmente, qualquer novo tipo de conteúdo textual.

Referências

[1] PHP DOMXML. PHP DOMXML Extension . `http://www.php.net/domxml`.

[2] Miguel Mira da Silva Hugo Alhandra. Abordagens para armazenamento de metadata em Data Warehouse usando o XML. Artigo, IST, 2003.

[3] Dublin Core Metadata Initiative. Dublin Core Metadata Initiative, 2004. `http://dublincore.org/`.

[4] Dublin Core Metadata Initiative. Guidelines for implementing Dublin Core in XML, 2004. `http://dublincore.org/documents/dc-xml-guidelines/`.

[5] PEAR. PHP Extension and Application Repository , 1999. `http://pear.php.net`.

[6] PEAR.DB. Database Abstraction Layer , 2002. `http://pear.php.net/package/DB`.

[7] PHP. Andi Gutmans, Zeev Suraski (PHP3), 1997. `http://www.php.net`.

[8] João Paulo Rodrigues. SIGDiC - Sistema Integrado de Gestão e Difusão de Conteúdos. Tese de mestrado, INESC Porto, 2004.

[9] Miguel Mira da Silva Rui Cerveira Nunes. Comparação de várias estratégias para armazenamento de XML. Artigo, IST, 2003.
[10] W3C. XSL Transformations, 1999. `http://www.w3.org/TR/xslt`.
[11] W3C. Extensible Markup Language (XML), 2003. `http://www.w3.org/XML/`.
[12] PHP XSLT. PHP XSLT Extension . `http://www.php.net/xslt`.

eABC : um repositório institucional virtual

Johnny Santos, Cláudio Teixeira, Joaquim Sousa Pinto

Departamento de Electrónica e Telecomunicações,
Instituto Engenharia Electrónica e Telemática de Aveiro
Universidade de Aveiro
3810-193, Aveiro
Email: johnny@ieeta.pt, claudio@ieeta.pt, jsp@det.ua.pt

Abstract. O eABC surge no contexto dos repositórios institucionais desenvolvidos pela Universidade de Aveiro. Neste artigo descreve-se resumidamente em que consistem estes repositórios e algumas dificuldades que advêm aquando da sua implementação. Com este artigo pretende-se também apresentar um caso prático do uso da tecnologia XML e de algumas das tecnologias que lhe estão associadas como o XPath, o XQuery e o XSLT, como suporte ao armazenamento, pesquisa e visualização dos documentos do eABC.

1 Introdução

O eABC (http://abc.ii.ua.pt/) é um projecto que vem sendo desenvolvido desde 2001 na Universidade de Aveiro (UA) com o objectivo "*de proporcionar à comunidade científica lusófona, um sistema de publicação e armazenamento de documentos científicos, proporcionando a disponibilização electrónica de periódicos científicos e académicos, bem como a criação de um banco de teses e dissertações*".

O sistema foi pensado inicialmente com um duplo objectivo: por um lado pretendia-se que os utilizadores (elementos da comunidade académica lusófona) pudessem fazer a gestão das suas publicações académicas através deste sistema e, por outro lado, também as Empresas e o tecido produtivo lusófono encontrassem aqui um ponto de encontro de quadros e valências técnicas relevantes para a sua actividade de negócio.

O sistema entretanto desenvolvido foi adoptado pelo Instituto de Investigação da Universidade de Aveiro (II-UA) como uma base para suporte à investigação científica desenvolvida na UA. Pretendia o II-UA com este sistema criar um repositório onde pudesse ser armazenado todo o trabalho intelectual produzido pela Universidade e,

posteriormente, obter relatórios de Gestão Científica com as listas de publicações quer dos seus investigadores, quer das diversas Unidades de Investigação integradas neste Instituto. Este sistema seria um auxiliar dos Serviços do Instituto que deste modo teriam uma maior facilidade no preenchimento dos relatórios periódicos para as instituições que superintendem a Investigação Científica, quer a nível nacional quer internacional.

Ao longo do tempo, e fruto das necessidades sentidas pelos utilizadores do sistema, o eABC foi sofrendo alguns melhoramentos, o último dos quais, com a inclusão de um "Registo de Autoridade" com o objectivo de resolver problemas relacionados com a associação dos autores aos seus artigos científicos. Este problema pode, resumidamente, ser explicado com um exemplo simples. A grande maioria dos elementos da comunidade científica internacional utiliza de modo sistemático um nome como o seu "nome científico". No entanto, quer conferências quer revistas possuem o seu formato próprio de representação dos autores. Esses formatos nem sempre vão de acordo com o interesse dos próprios autores. De modo a poder introduzir no sistema a publicação, não fugindo à fonte impressa, esses artigos são inseridos com os nomes que lá estão inscritos mas posteriormente não são identificados com o "cientista" que o publicou. Esse problema afecta a manutenção de uma lista actualizada de publicações por parte dos cientistas e afecta também directamente a lista das publicações da Unidade de Investigação à qual ele pertence. A introdução de um índice de autoridade permite, justamente, a possibilidade de manter o rigor relativamente à fonte impressa e ainda a atribuição dos créditos ao verdadeiro autor da obra.

Essas alterações, parciais e localizadas, não resolveram contudo alguns problemas estruturais que ainda persistem. Por isso, neste momento, o eABC está novamente em mudança. Esta nova fase vai implicar grandes alterações, sobretudo no que diz respeito à tecnologia sobre qual vai ser desenvolvido o sistema.

2 Repositórios institucionais

A necessidade de preservar digitalmente documentos, que outrora se encontravam apenas em papel, e permitir que os mesmos estejam acessíveis a qualquer pessoa, têm levado a que determinadas organizações como universidades, bibliotecas, entre outras, tenham criado os seus próprios repositórios. Ao nível nacional temos, por exemplo, a Biblioteca Nacional, e o RepusitoriUM (Universidade do Minho). Ao nível internacional temos, por exemplo, o projecto SHERPA da universidade de Nottingham, e o projecto SPACR, que tem a cooperação de várias universidades, librarias e organizações.

Nestes últimos anos, o aparecimento de repositórios institucionais têm vindo a aumentar, sobretudo dentro da comunidade universitária. Estes repositórios têm como objectivo, a gestão, o armazenamento, a preservação e a divulgação do trabalho intelectual produzido pela universidade.

É da responsabilidade e interesse das universidades, que os membros da sua comunidade disponibilizem os seus trabalhos nestes repositórios institucionais. Esse interesse é também partilhado pela comunidade científica pois ao disponibilizarem os

seus trabalhos estão a contribuir para que a sua instituição possa ganhar prestigio, estatuto, e credibilidade no meio da comunidade científica, mas também para o progresso científico.

Muitas vezes um dos maiores problemas com que as universidades se deparam quando pretendem criar um repositório institucional, não se prende tanto com o desenvolvimento tecnológico do próprio repositório, mas sim com a possibilidade de colocar alguns dos conteúdos nos mesmos. A título de exemplo, imagine-se o caso de um investigador que publica o seu trabalho numa revista conceituada da sua área de trabalho, para que este possa ser lido e citado em outros trabalhos científicos. Normalmente, ao publicar nestas revistas, o autor está imediatamente a prescindir dos seus direitos sobre o mesmo, ficando deste modo impedido de disponibilizar o seu trabalho de modo aberto no repositório da sua instituição de acolhimento. Muitas vezes isso não é compreensível ou mesmo aceitável porque esses trabalhos são desenvolvidos com recurso a financiamento público proveniente das universidades e de fundações. Os mais lesados com este facto são os demais membros da instituição, pois ficam impedidos de aceder à informação, que poderia permitir um avanço mais rápido do seu trabalho.

De modo a resolver os problemas associados com a pertença física e intelectual de muitos dos conteúdos institucionais, pretende-se que no eABC fique catalogado todo o trabalho científico desenvolvido pelos membros desta instituição e, sempre que possível, uma referência (apontador) para a localização física de onde este trabalho se encontra.

Esta metodologia difere do procedimento anterior onde se pretendia, para além do registo bibliográfico, a criação de um acervo digital das fontes geradoras das referências bibliográficas. Embora isso ainda seja possível, neste momento este objectivo foi abandonado pois já não se pretende armazenar no sistema as fontes em formato digital. Isto permite mudar a tipificação do sistema de repositório digital para um repositório institucional virtual.

O eABC não se destina exclusivamente aos membros da UA; qualquer investigador pode lá catalogar as suas publicações. Ao permitir que o trabalho desenvolvido quer pelos membros da sua instituição, quer por outros investigadores, esteja acessível (sempre que possível) a outros investigadores, o eABC pode, ainda que modestamente, estar a contribuir para o avanço da ciência na lusofonia.

3 UNIMARC, como norma de codificação de dados

É pretensão da Universidade que o eABC possa armazenar os vários tipos de publicações científicas produzidas pelos seus investigadores, professores, alunos: artigos, dissertações de mestrado, teses de doutoramento, partituras, patentes, etc. Para que o sistema pudesse albergar estes diferentes tipos de publicações era necessário escolher uma norma que fosse o mais abrangente possível. É ainda objectivo que o repositório institucional da UA possa ser exportado para outras instituições ou para um repositório nacional e que possa incorporar dados vindos de outros repositórios.

Na primeira versão deste sistema não foi utilizada nenhuma norma internacional, quer para a recolha, quer para o armazenamento dos dados. Embora os dados entretanto introduzidos possam ser recuperados, a não obediência a uma norma internacional vai obrigar à elaboração de uma aplicação de conversão de dados.

Nesta nova fase, a norma escolhida para armazenar a catalogação dos documentos foi a norma "Universal MARC format" (UNIMARC) [6], [7]. Esta norma foi publicada pela International Federation of Library Associations (IFLA) em 1967, e é a norma que tem sido utilizada pela maior parte dos países da Europa. Para além da norma UNIMARC utilizada na catalogação dos registos bibliográficos, usou-se também a norma UNIMARC/Authorities [8], para a criação e manutenção de um registo de autoridade, com o objectivo de resolver os problemas relacionados com a associação dos autores aos seus documentos.

4 Implementação do eABC

O eABC é um dos projectos que está inserido no contactUA. O contactUA é um dos projectos do Programa Aveiro Digital 2003-2006 [16] e pretende ser o *front-end* entre a comunidade científica e a comunidade em geral. Sendo eABC um repositório institucional contendo a produção científica da UA, numa perspectiva também de prestação de serviço à comunidade e ao tecido produtivo regional, foi naturalmente englobado no Projecto contactUA.

O eABC não é um sistema novo. É um projecto em desenvolvimento desde 2001 e que está disponível em http://abc.ii.ua.pt/. Tem vindo desde o seu início a passar por algumas fases de melhoramento, de maneira a poder responder às necessidades dos utilizadores do sistema. No entanto, essa alterações, parciais e localizadas, não resolveram contudo alguns problemas estruturais que ainda persistem. Na altura em que se iniciou o seu desenvolvido recorreu-se a bases de dados relacionais, para armazenar os dados provenientes de catalogação. Recorreu-se ainda à tecnologia "*Active Server Pages*" (ASP) [17] como suporte aos formulários de inserção dos dados e de pesquisa.

Neste novo desenvolvimento recorreu-se à tecnologia mais actual. É utilizada a tecnologia XML[9] como suporte ao armazenamento e pesquisa nos dados armazenados, e à tecnologia .NET [18] para o desenvolvimento da interface com os utilizadores do sistema.

4.1 Uso da tecnologia XML no eABC

Ao longo de todas as fases da implementação do eABC, o uso da tecnologia XML é uma constante, isto porque, é usada tanto para o armazenamento dos dados, como para a pesquisa e a visualização dos registos bibliográficos. Uma das maiores vantagens de trabalhar com esta tecnologia, esta relacionada com o facto de facilmente podermos definir uma estrutura para o armazenamento dos dados. Estando os dados armazenados em XML, podem ser também facilmente exportados, visto que cada vez mais, o XML é usado para o "transporte" de dados.

Como já foi referido anteriormente, a norma de catalogação escolhida para o armazenamento dos vários tipos de documentos, foi a norma UNIMARC. OS vários tipos de documentos que são possíveis guardar no eABC são:

- Livros
- Capítulos de Livros
- Publicação de revista com júri de avaliação
- Publicação em acta de congresso simpósio e conferência, com júri de avaliação
- Contributos para organismos para normalização
- Artigos de divulgação
- Estudos e relatórios
- Acta de congresso, conferência e simpósio
- Textos de ensino
- Dissertações de mestrado
- Teses de doutoramento
- Patentes

Foi utilizada uma abordagem XML *document centered* [15]. Isso implica que a informação referente a cada um registos bibliográficos foi armazenada num ficheiro XML separado. Estes documentos teriam que armazenar os dados dos documentos segundo a norma UNIMARC. A Figura 1 representa o *schema* (XSD – acrónimo de XML Schemas) que foi utilizado para armazenar os documentos. Este *schema* foi cedido pela Biblioteca Nacional (BN), e baseia-se, tal como já referido, na norma UNIMARC definida pela IFLA.

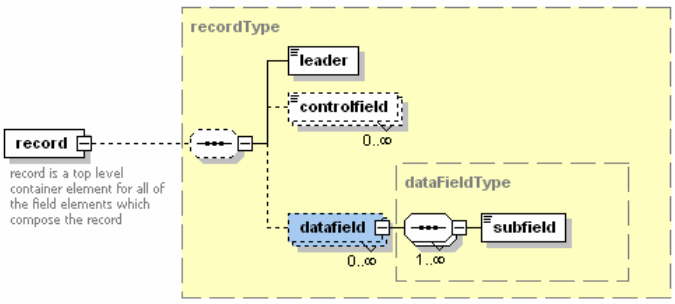


Figura 1 - Schema usado para o armazenamento dos documentos e para o registo de autoridade

Este *schema* apesar de ser simples é suficiente para armazenar qualquer tipo de documento. Na Figura 2 é mostrado um pequeno exemplo de como um documento, neste caso em particular uma publicação em acta de conferência com júri de avaliação, é guardado no ficheiro XML segundo a norma UNIMARC.

```
- <record>
    ...
    <controlfield tag="001">D_PAC</controlfield>
  - <datafield tag="700" ind1="" ind2="1">
      <subfield code="a">Teixeira</subfield>
      <subfield code="b">Claudio</subfield>
    </datafield>
  - <datafield tag="701" ind1="" ind2="1">
      <subfield code="a">Santos</subfield>
      <subfield code="b">Johnny</subfield>
    </datafield>
  - <datafield tag="701" ind1="" ind2="1">
      <subfield code="a">Pinto</subfield>
      <subfield code="b">Joaquim Sousa</subfield>
    </datafield>
  - <datafield tag="701" ind1="" ind2="1">
      <subfield code="a">Martins</subfield>
      <subfield code="b">Joaquim Arnaldo Carvalho</subfield>
    </datafield>
  - <datafield tag="200" ind1="1" ind2="">
      <subfield code="a">VILLAGES, TOWNS AND CITIES: AN E-GOVERNMENT CASE STUDY</subfield>
    </datafield>
  - <datafield tag="101" ind1="0" ind2="">
      <subfield code="a">eng</subfield>
    </datafield>
  - <datafield tag="215" ind1="" ind2="">
      <subfield code="a">p. 635-640</subfield>
    </datafield>
  - <datafield tag="330" ind1="" ind2="">
      <subfield code="a">Traditionally, the interaction between government agencies and citizens or companies was made in governmental buildings, where a
        public employee attended all the requests. This scenario is gradually changing and evolving because of the rapid growth of use of new communication
        technologies. Thanks to this, the government-citizens interaction has been moving to considerably closer to the citizen, and sometimes it takes place
        on his home or working place via his personal computer and the internet.</subfield>
    </datafield>
  - <datafield tag="710" ind1="" ind2="0">
      <subfield code="a">Proceedings of the 2004 International Conference on Computational & Experimental Engineering & Science</subfield>
      <subfield code="f">26 Julho, 29 Julho, 2004</subfield>
      <subfield code="e">Portugal</subfield>
    </datafield>
    ...
  </record>
```

Figura 2 - Exemplo de um documento guardo segundo a norma UNIMARC no ficheiro XML

O registo bibliográfico mostrado na Figura 2 não está completo e é apenas apresentado como exemplo. Pode verificar-se que se trata de uma publicação em acta de conferência com júri de avaliação (valor “D_PAC” guardado no marcador “001”), em que o primeiro autor foi o Cláudio Teixeira (marcador “700”), em que os co-autores foram: Johnny Santos, Joaquim Sousa Pinto e Joaquim Arnaldo Carvalho Martins (marcador “701”). O título da publicação é guardado no marcador “200”, o idioma em que o documento está escrito é guardado no marcador “101”. As páginas onde se encontra a publicação no livro da conferência, é guardada no marcador “215”, o resumo é guardado no marcador “330”, o nome da conferência, a data e local, onde se realizou a mesma, é guardada no marcador “710”.

Como foi mencionado anteriormente, um dos últimos melhoramentos do eABC foi a criação de um registo de autoridade. Para essa funcionalidade se mantivesse, era necessário guardar a informação segundo a norma UNIMARC/Authority. O *schema* utilizado para armazenar a informação sobre os autores foi o mesmo que o utilizado para o armazenamento dos registos bibliográficos (Figura 1). Para além de armazenar informação relativa à identificação do Autor (nome, email), foi ainda necessário armazenar os dados referntes à instituição e/ou Unidade de Investigação à qual o autor está associado. Este procedimento permite garantir que caso um autor mude de Instituição ou de Unidade de Investigação(UI), que as suas publicações, continuarão

no futuro, e dentro de um mesmo período temporal, a pertencer a essa Unidade de Investigação. Se este princípio não for acautelado, no caso de um autor mudar de UI "carrega" para a nova UI todas publicações efectuadas durante o período em que pertencia à antiga. Este tipo de informação pode ainda ser útil no caso de se pretender fazer uma pesquisa mais elaborada, em que se pretenda saber os registos referentes a um determinado autor numa dada Instituição pela qual tenha passado. Na Figura 3 é mostrado um exemplo de um registo de autoridade.

```
- <record>
    <controlfield tag="001">1</controlfield>
  - <datafield tag="200" ind1="" ind2="1">
      <subfield code="a">Jesus</subfield>
      <subfield code="b">Johnny Enrique Santos</subfield>
    </datafield>
  - <datafield tag="390" ind1="0" ind2="">
      <subfield code="a">johnny@ieeta.pt</subfield>
    </datafield>
  - <datafield tag="210" ind1="0" ind2="2">
      <subfield code="a">Universidade de Aveiro</subfield>
      <subfield code="z">1/9/1998-31/9/2003</subfield>
    </datafield>
  - <datafield tag="210" ind1="0" ind2="2">
      <subfield code="a">Universidade de Aveiro</subfield>
      <subfield code="b">Instituto de Engenharia Electrónica e Telemática de Aveiro - IEETA</subfield>
      <subfield code="z">1/10/2003-</subfield>
    </datafield>
  </record>
```

Figura 3 - Exemplo de um registo de Autoridade armazenado segundo a norma UNIMARC/Authorities

No exemplo mostrado na Figura 3 é possível ver qual o identificador interno que foi atribuído ao autor (armazenado no marcador "001"), o nome do autor (marcador "200"), o e-mail (marcador "390"), e as instituições pelas quais já passou. Neste caso em particular pode ver-se que o autor passou pela Universidade de Aveiro durante o período que medeia entre 1/9/1998 e 31/9/2003 e que, actualmente, se encontra também na Universidade de Aveiro, mas agora numa das Unidades de Investigação, mais precisamente no "Instituto de Engenharia Electrónica e Telemática de Aveiro - IEETA"

Para além do *schema* anterior usado para armazenar os documentos, houve a necessidade de criar mais três schemas, dois deles funcionando como dicionários de dados. Estão representados na Figura 4 e na Figura 5.

Sendo este sistema destinado ao registo de publicações científicas, há a necessidade/possibilidade de associar áreas científicas às publicações. A título de exemplo este procedimento permite, por exemplo, associar a um registo uma área Científica, "Ciências da Tecnologia e da Engenharia", e, se necessário, uma ou mais sub-áreas como "Engenharia Electrotécnica e Informática". Na Figura 4 está representado o *schema* que foi criado para armazenar o dicionário referente às várias áreas científicas e às suas respectivas sub-áreas. A informação da área e sub-área fica associada ao documento no marcador "615".

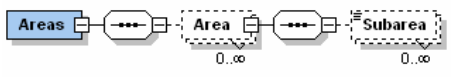


Figura 4 - Schema usado para o dicionário de áreas científicas e suas subáreas

Uma das funcionalidades que se pretende disponibilizar para os utilizadores do sistema no modo de pesquisa, é a possibilidade de pesquisar pelos registos de uma determinada instituição ou unidade de investigação. Para cumprir tal requisito, houve a necessidade de criar um dicionário que pudesse armazenar as várias instituições e as suas unidades de investigação. Na Figura 5 está representado o *schema* que suporta esse dicionário. Os dados armazenados neste dicionário estão de acordo com acordo com a estrutura da Fundação para a Ciência e Tecnologia. Neste dicionário as instituições pertencem a Distritos, e cada Instituição pode ter uma ou mais Unidades de Investigação.

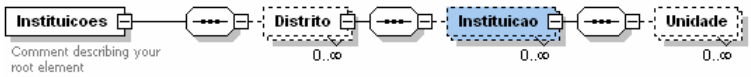


Figura 5 - Schema usado para armazenar as instituições e suas unidades de investigação

Como o eABC é um sistema que pretende servir utilizadores, foi necessário criar um "registo de utilizadores" do sistema. Na Figura 6 está representado *schema* desenhado de modo a suportar essa informação. Os dados constantes neste registo devem ser mínimos, daí a existência apenas de um identificador de utilizador, uma senha (guardada de acordo com algoritmos de encriptação/hash), e alguma informação pessoal para contacto com a pessoa.

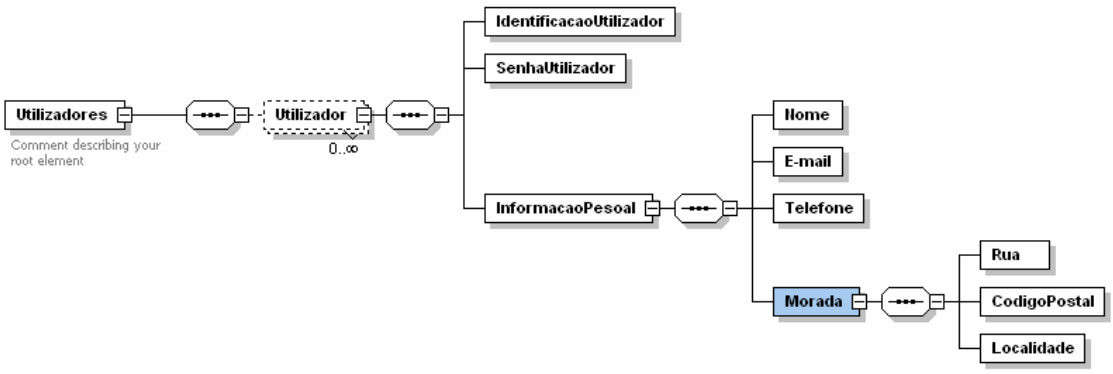


Figura 6 - Schema usada para armazenar os utilizadores

4.2 Pesquisa sobre os documentos

O eABC não pretende apenas ser um repositório onde se possa armazenar os vários tipos de registos bibliográficos, mas pretende também que os mesmos possam ser pesquisados, de maneira a que a comunidade científica possa ter acesso aos mesmos. As pesquisas a efectuar sobre estes registos podem ser pesquisas das mais simples, como uma pesquisa por texto livre em todos os campos (titulo, autor, ano, área temática, ...) dos documentos, até pesquisas mais elaboradas como pesquisas por uma dada área temática, ou de todas as publicações de uma dada unidade de investigação.

De maneira a que os registos, que estavam armazenados em documentos XML no sistema de ficheiros, pudessem ser pesquisados houve a necessidade criar um sistema de indexação dos mesmos. Para tal foi utilizado o Serviço de Indexação de Ficheiros do Sistema Operativo Windows 2003. Este sistema de indexação possibilita a indexação de sistemas de ficheiros ou de servidores *web*. Tem como principal atractivo fazer uma pré-indexação dos ficheiros, o que permite pesquisas muito rápidas. A principal desvantagem deste sistema de indexação é não indexar correctamente ficheiros XML. De modo alargar as funcionalidades da tecnologia deste Serviço de Indexação foi-lhe adicionado um novo filtro (IFilter) capaz de indexar ficheiros XML [19] [20]. Este filtro permite aos utilizadores indexar e pesquisar ficheiros XML de estrutura arbitrária baseada no seu conteúdo.

O uso da tecnologia XML, nesta camada, foi também necessária para poder efectuar pesquisas sobre os registos bibliográficos. Neste caso, em particular, foi necessário usar XPath [10] e XQuery [11].

4.3 Visualização dos resultados das pesquisas

Uma das preocupações do eABC, diz respeito à maneira como o resultado da pesquisa é mostrada ao utilizador. Não se pretendia que os dados dos vários tipos de documentos fossem extraídos de qualquer maneira dos ficheiros XML, mas sim que estes pudessem sair formatados segundo as normas ISBD [12], IEEE, NP405 [13] e SPQS.

Umas das vantagens dos resultados saírem formatados segundo estas normas, é o facto de poderem ser directamente incorporadas, por exemplo, num currículo académico.

De maneira a que os resultados da pesquisa pudessem sair formatados segundo as normas mencionadas anteriormente, foi necessário recorrer mais uma vez à tecnologia XML, neste caso, a tecnologia XSL Transformations (XSLT) [14].

Foram definidas um conjunto de transformações distintas, uma para tipo de registo bibliográfico (artigos, dissertações de mestrado, teses de doutoramento, ...) e para cada um dos diferentes formatos de representação implementados (ISBD, IEEE, NP405 e SPQS).

5 Conclusões

Os repositórios institucionais têm vindo a aumentar cada vez mais ao longo destes últimos anos, sobretudo no meio da comunidade académica. No entanto, a criação de um repositório com o objectivo de guardar o trabalho intelectual das universidades deve obedecer a normas internacionalmente aceites de modo a ser possível a sua troca a e a incorporação de dados vindos de outros repositórios institucionais. O eABC pode dizer-se que é um repositório institucional *virtual*, isto porque não pretende armazenar no seu *corpus* os documentos em formato digital, mas sim um apontador (url) para os documentos (sempre que possível). A publicitação do apontador do documento, ao invés do armazenamento de uma cópia do documento, tem como vantagem ultrapassar os problemas associados com a cedência dos direitos de autor a revistas e jornais científicos.

A utilização das tecnologias associados ao XML, por se tratar de tecnologias neutrais de formato aberto, fornecem também ao sistema a versatilidade desejada em repositórios institucionais em que a mobilidade e troca de conteúdos são importantes.

Referências bibliográficas

[1] Lynch, Clifford A. “Institutional Repositories: Essential Infrastructure for Scholarship in the Digital Age”, ARL, no. 226 (February 2003): 1-7. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.arl.org/newsltr/ 226/ir.html> .

[2] Pinfield, Stephen, “Open Archives and UK Institutions: An Overview”, D-Lib Magazine, Volume 9 Number 3, ISSN: 1082-9873. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.dlib.org/dlib/march03/pinfield/ 03pinfield.html>.

[3] Drake, Miriam A., “Institutional Repositories: Hidden Treasures”, Searcher: The Magazine for Database Professionals, Information Today, Inc., Vol. 12 No. 5 — May 2004, ISSN: 1070-4795. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.infotoday.com/searcher/may04/drake.shtml>.

[4] “Appendix 51: Memorandum from Hallward Library, University of Nottingham, SHERPA Project”, Committee on Science and Technology - Written Evidence, United Kingdom Parliament. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.publications.parliament.uk/pa/cm200304/cmselect/ cmsctech/399/399we62.htm>.

[5] Hubbard, Bill, “SHERPA and Institutional Repositories”, Serials, The University of Notthingan, 16, 3, November 2003, pp 243-247. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://eprints.nottingham.ac.uk/archive/ 00000095/01/sherpa&instrep.pdf>

[6] “UNIMARC: An Introduction”, Universal Bibliographic Control and International MARC Core Programme, International Federation of Library

Associations and Institutions, Latest Revision: March 3, 1999. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.ifla.org/VI/3/p1996-1/ unimarc.htm>.

[7] "UNIMARC Manual: Bibliographic Format 1994", IFLA Universal Bibliographic Control and International MARC Core Programme (UBCIM), International Federation of Library Associations and Institutions, Latest Revision: 6 April 2000. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.ifla.org/VI/3/p1996-1/sec-uni.htm>.

[8] "UNIMARC / Authorities (Concise version)", IFLA Universal Bibliographic Control and International MARC Core Programme (UBCIM), International Federation of Library Associations and Institutions, Latest Revision: May 25, 1999. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.ifla.org/VI/3/p1996-1/ucaf.htm>.

[9] Bray, T., et al (editors), "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation 6 October 2000, W3C. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.w3.org/TR/2000/REC-xml-20001006>

[10] Clarck, James, DeRose, Steve (editors), "XML Path Language (XPath) – Version 1.0", W3C Recommendation 16 November 1999, W3C. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.w3.org/TR/xpath>.

[11] BOAG, Scott, et al, "XQuery 1.0: An XML Query Language" W3C, Disponível em <http://www.w3.org/TR/xquery>

[12] Saur, K.G. "ISBD(G): General International Standard Bibliographic Description", International Federation of Library Associations and Institutions, 1991, March, UBCIM Publications - New Series Vol 6, München · London · New York · Paris 1992. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.ifla.org/VII/s13/pubs/isbdg.htm>.

[13] Instituto Português da Qualidade, "NP 405-1 : 1994 : informação e documentação : referências bibliográficas : documentos impressos", Instituto Português da Qualidade, Lisboa, 1995, 49p..

[14] Clarck, James (editor), "XSL Transformations (XSLT) – Version 1.0", W3C Recommendation 16 November 1999, W3C. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.w3.org/TR/xslt>.

[15] Daum, Berthold, Merten Udo, "System Architecture with XML", Morgan Kaufmann Publishers, 2002, 458 pages. ISBN 1-55860-745-5.

[16] "Projectos Aveiro Digital 2003 - 2006 :: ContactUA", Programa Aveiro Digital 2003 – 2006. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.aveiro-digital.pt/default.asp?func=proj-24>.

[17] Walther, Stephen "Active Server Pages 2.0 unleashed", Sams Publishing, corp. 1999. XXIII, 856 p.+1 CD-ROM. ISBN 0-672-31613-7

[18] Liberty, Jesse, Hurwitz, Dan "Programming ASP.NET", O'Reilly & Associates, corp. 2002. XIII, 944 p.. ISBN 0-596-00171-1

[19] "XML IFilter for easy XML file indexing – White Paper", QuiLogic Inc., Austria. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.quilogic.cc/QLXFilterWhiteP.pdf>.

[20] "XML IFilter for easy XML file indexing – User's Guide – V 2.19" ", QuiLogic Inc., Austria. Em linha, acedido em 19 de Novembro de 2005, disponível em <http://www.quilogic.cc/QLXFilter.pdf>.

XML Topic Maps e Mapas de Conceitos

Francisco Paz, Paulo Teixeira, Giovani Rubert Librelotto, Sandra Cristina Lopes, and Pedro Rangel Henriques

Universidade do Minho, Departamento de Informática
4710-057, Braga, Portugal
prh@di.uminho.pt

Abstract. O desafio que os programadores de software educativo enfrentam é o de terem que implementar adequadamente uma determinada concepção da aprendizagem, que não é necessariamente a sua. O que é hoje conhecido sobre os processos de aprendizagem dos indivíduos não deriva de investigação realizada com tecnologia, nem da aprendizagem apoiada no computador, decorre sobretudo dos contributos da psicologia do desenvolvimento e da psicologia da aprendizagem. Assim cabe ao programador compreender a metodologia que o pedagogo quer seguir e espelhá-la no software que desenvolve, em vez de ser a tecnologia que ele domina a impor o estilo de aprendizagem implícito nesse software.
Nesse sentido, este artigo sugere o uso de Mapas de Conceitos como forma de organizar o conhecimento que constitui o conteúdo de uma disciplina. No artigo defende-se, então, o uso da norma Topic Maps, com base na sintaxe XTM (XML Topic Maps), para descrever o mapa de conceitos de um domínio em particular que é o tema da dita disciplina. A argumentação apresentada a favor desta escolha tecnológica assenta na possibilidade de derivar automaticamente da descrição XTM um visualizador (acessível via Web) que permite navegar através dos conceitos encontrados neste mapa. Essa navegação permite, por um lado, ir de conceito em conceito analisando as várias relações que os ligam e, por outro lado, possibilita o acesso à informação final associada aos conceitos básicos.
Neste contexto pretendemos mostrar as potencialidades da tecnologia Topic Maps e como esta pode ser uma parceira que providencia oportunidades de aprendizagem.

1 Introdução

Os Mapas de Conceitos (MC) têm sido usados em inúmeras áreas científicas, da educação à política, da filosofia às ciências ditas exactas, para construir representações visuais de estruturas de conhecimento, de fenómenos científicos complexos e até de ideias em fase embrionária.

No seguimento de experiências efectuadas, concluiu-se [13] que a organização de conteúdos curriculares com apoio em MC é uma das abordagens que tem, nos dias de hoje, mais sucesso para esquematizar o tipo de ensino que se pretende. Como se dirá à frente, os MC têm outras importantes vantagens tanto para o professor como para o aluno. Apesar de tudo isso, constata-se que o seu uso

e exploração está muito aquém do que seria de esperar e desejável. Na nossa opinião, tal deve-se, principalmente, ao esforço requerido para desenhar no papel mapas reais, que tendem a ser grandes e complexos, e à inerente dificuldade em navegar posteriormente nesse mapa acedendo aos materiais didácticos associados aos conceitos. Recentemente assistiu-se ao aparecimento de vários livros em que surgem MC no início de cada capítulo, como forma de sintetizar a matéria aí incluída, mas como é fácil de perceber, não há uma interligação directa entre os nós do mapa e os locais do livro onde os conceitos são explicados, ...

Vivendo-se numa era em que existe uma preocupação constante na integração da novas tecnologias no processo ensino/aprendizagem, a tecnologia XML Topic Maps (XTM) vem permitir que os mapas de conceitos tradicionais saltem do papel para o écran do computador, trazendo as tais capacidades de exploração que no mundo físico eram desejadas mas impossíveis de criar!

Neste artigo vai-se mostrar que Mapas de Conceitos e Topic Maps (TM) foram criados com a mesma finalidade, *representar o conhecimento*, mas por correntes diferentes de investigação[1], e que a integração de ambos, além de possível e sistemática, aporta inúmeras vantagens ao sistema de ensino/apredizagem.

Pretende-se, então, sugerir que se use a notação XTM — criada no mundo do XML para descrever TM — e todo o arsenal tecnológico associado, para representar MC de modo a criar uma interface Web que permita, simultaneamente, visitar os conceitos percorrendo as ligações que os unem e aceder aos materiais didácticos associados a alguns desses conceitos e também disponíveis na Web.

Nas duas secções seguintes serão apresentados os dois temas de base: na Secção 2 introduzem-se os Mapas de Conceitos, do ponto de vista dos agentes educativos; e na Secção 3 fala-se de Topic Maps do ponto de vista dos tecnólogos. Na Subsecção 3.1 refere-se, sucintamente, a linguagem XML que permite expressar os Topic Maps, a XTM. A ponte entre Mapas de Conceitos e Topic Maps será feita na Secção 4, onde será, ainda, apresentado um caso de estudo (4.1). O gerador de interfaces Web a partir dos Topic Maps será apresentado na Secção 5, bem como a sua aplicação concreta ao caso de estudo introduzido na Secção precedente, discutindo-se prós e contras das alternativas que foram exploradas (Omnigator e Ulisses). Por último, na Secção 6, será feita uma breve reflexão sobre os benefícios que se retiram de projectos multi-disciplinares como este que se descreve ao longo do artigo e comentam-se os próximos passos para lhe dar o prosseguimento desejado.

2 Mapas de Conceitos

Os Mapas de Conceitos (MC) têm a sua origem no movimento da teoria construtivista da aprendizagem, de Ausubel [1]. A ideia central desta teoria é da aprendizagem significativa, que surge em oposição à de aprendizagem mecânica, em que o conhecimento é memorizado sem que o aluno estabeleça relações entre a nova informação e aquela que já existe na sua estrutura cognitiva. Assim, a

[1] Por um lado temos os agentes educativos por outro os tecnólogos.

aprendizagem significativa é o processo através do qual uma nova informação se relaciona com um aspecto relevante da estrutura do conhecimento do sujeito [1]. O aluno ao confrontar-se com novos conhecimentos deve ser capaz de distinguir os principais conceitos e de os relacionar com os conhecimentos anteriores. Os Mapas de Conceitos foram usados pela primeira vez, há mais de 4 décadas, por Novak o qual defendia que o mapa de conceitos pode ter várias funções em simultâneo: como recurso de auto-aprendizagem ao dispor dos alunos e professores; como método para encontrar e explicitar significado para os materiais de estudo; e como estratégia que estimula a organização dos materiais de estudo. Os MC são, para Joseph Novak [11], uma ferramenta para organizar e representar conhecimento.

Hoje em dia, os mapas de conceitos são utilizados com variados fins pedagógicos:

- como uma técnica de planificação;
- como forma de estruturar conteúdos para a leccionação de determinada unidade didáctica (que será o exemplo apresentado ao longo deste artigo);
- para sintetizar informação;
- para consolidar informação a partir de diferentes fontes de pesquisa;
- como meio de simplificar a abordagem a problemas complexos;
- para ajudar o aluno a fixar a sua atenção nos conceitos mais importantes, ajudando-o também a aprender a representar ideias de uma maneira gráfica;
- como suporte à avaliação formativa;
- como instrumento de apoio durante a exposição de um conteúdo;
- como meio diagramático de esclarecer, ou descrever, ideias que as pessoas têm sobre um determinado assunto;
- etc.

Um MC vai muito além de um esquema convencional desenhado ad-hoc, informalmente; ele tem uma semântica precisa que pode ser descrita formalmente por um grafo que é constituído por nós, onde se inscrevem os conceitos (substantivos no singular), e ramos (ligações), que representam as relações (verbos transitivos) entre conceitos[2]. Assim, o MC pode ser lido caminhado no grafo e formando proposições numa linguagem explícita e concisa. De igual modo, também as frases da nossa linguagem comum podem facilmente ser traduzidas para o MC; por exemplo, na frase *squash é um desporto*, "*squash*" e "*desporto*" são dois conceitos ligados através de um relacionamento "*é-um*". O mapa de conceitos que representa esta frase é o mais simples possível, sendo constituído por dois nós conectados por uma ligação, conforme se pode ver na Figura 1. Estes relacionamentos são nominativos, ou seja, cada relacionamento entre dois conceitos forma uma proposição.

Já a Figura 2 ilustra um caso bem mais complexo, com mais de 2 dezenas de nós e cerca de 10 relações; trata-se de um mapa de conceitos concreto que pretende descrever o que é um MC. O termo *Mapa de Conceitos* é o nó de topo, ou inicial, do mapa apresentado, sendo caracterizado à custa do conceito

[2] Estas ligações cruzadas ajudam-nos a percepcionar como alguns domínios de conhecimento se relacionam entre si.

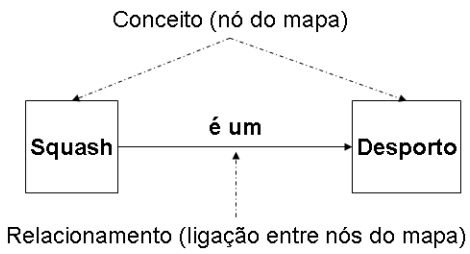


Fig. 1. Exemplo de um Mapa de Conceitos simples.

Conhecimento no contexto do *Ensino* e da *Aprendizagem*, o qual se exprime em termos de mais 3 conceitos: *Conceito*, *Proposição* e *Contexto*.

Ao construir um MC há total liberdade para a escolha do tipo de relações que se quer mostrar a ligar os conceitos. Contudo uma das relações sempre presente, e que é fundamental, é a relação *é-um* que permite criar uma hierarquia, com conceitos mais gerais e inclusivos no topo do mapa e os mais específicos, portanto, os conceitos menos gerais, dispostos hierarquicamente por baixo. Esta relação cria a noção de *inclusão de classes*, ou seja de *subclasse* (à frente designada por *instância*) e *classe*. No MC da Figura 2 podem ver-se 7 ocorrências desta relação.

Outra relação que é vulgar adicionar aos MC é a relação *é-exemplo-de* que permite introduzir exemplos específicos de acontecimentos ou objectos para ajudar a clarificar o significado de um determinado conceito. Uma vez mais a Figura 2 pode ser usada como referência pois contém 2 instâncias desta relação para ilustrar os conceitos *Acontecimento* e *Objecto*.

Em resumo, MC são instrumentos capitais para o professor programar uma disciplina, para recolher os materiais didácticos de apoio e para preparar cada aula. São também importantes para o aluno entender os objectivos da aula e aferir os conhecimentos adquiridos; segundo vários autores, a avaliação é precisamente um processo de comparação entre mapas de conceitos: o proposto pelo professor e o construído pelo aluno.

3 Topic Maps

Paralelamente à criação e aplicação de MC no âmbito das Ciências da Educação, surgiu muito mais recentemente no seio das Ciências da Informação um outro instrumento de representação do conhecimento designado por Topic Maps (TM). Os TM surgem num contexto onde a comunidade informática e arquivística (ligada às ciências documentais) estava preocupada com a modelação e manipulação do conhecimento envolvido em sistemas de informação com recursos variados e heterogéneos, tendo necessidade de criar e manusear índices sobre essas várias fontes.

Diferentes abordagens podem ser seguidas para conseguir a integração de recursos heterogéneos de informação, porém o problema principal será conquistar a interoperabilidade semântica entre as fontes, garantindo a manipulação do

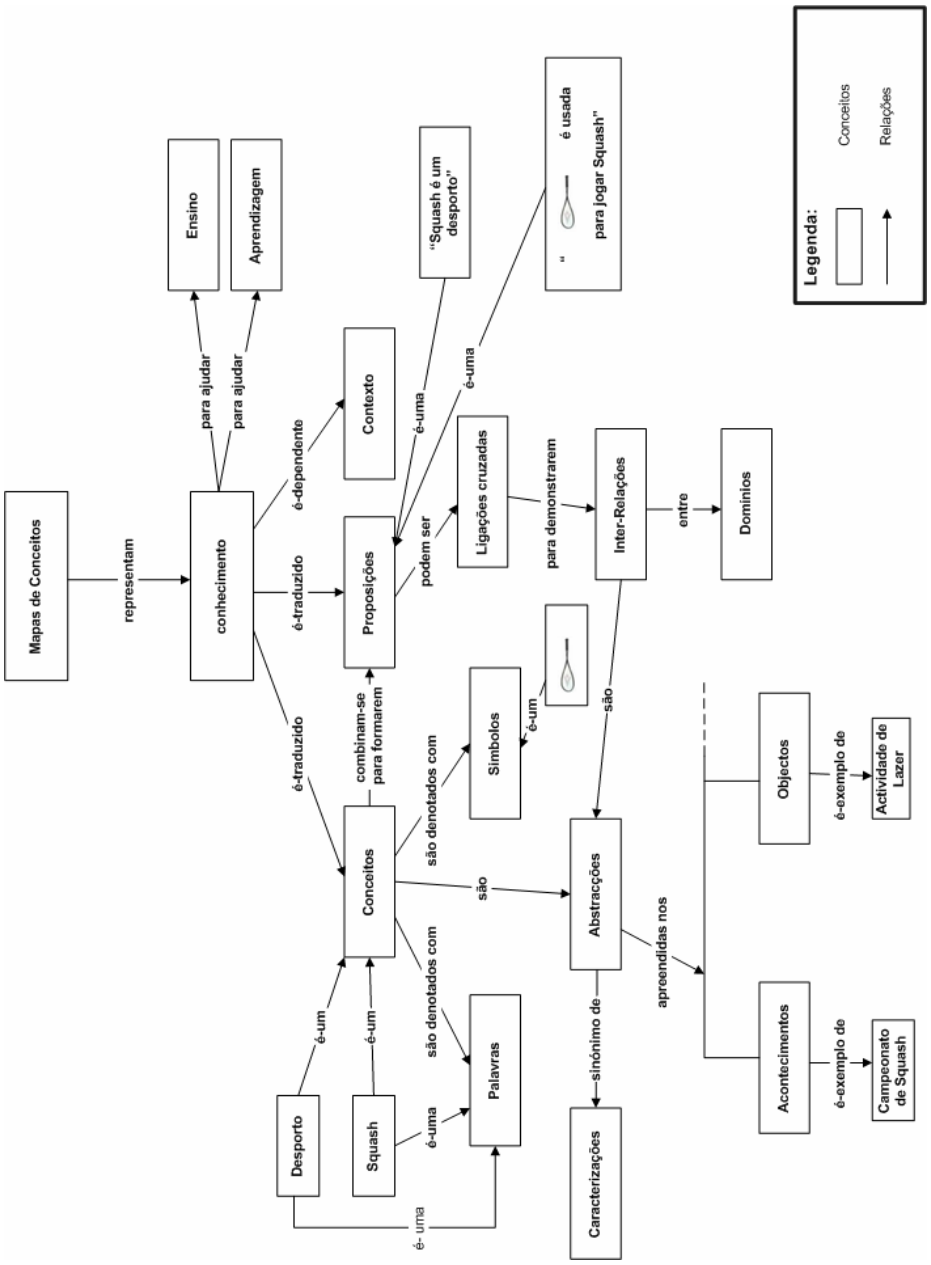


Fig. 2. Um Mapa de Conceitos mostrando as ideias e princípios-chave que caracterizam a noção de "Mapa de Conceitos"

conjunto sem forçar a conversão das partes em um formato único, como é o caso da interoperabilidade sintáctica.

Historicamente, o paradigma Topic Maps (uniformizado universalmente através da norma ISO 13250 [3]) foi definido para facilitar a fusão de diferentes esquemas de índices. Um formato comum para a anotação, usado para a indexação, é um passo crucial em direcção ao objectivo da interoperabilidade entre esquemas de índices. O que é necessário ainda é a interoperabilidade semântica. Enquanto que a especificação Topic Maps garante interoperabilidade sintáctica, ontologias provém interoperabilidade semântica. Uma ontologia é uma especificação, ou formalização, de um determinado universo de discurso (universo de conhecimento) [6]; alternativamente, uma ontologia pode ser entendida como uma teoria lógica que fornece uma explicação de uma conceptualização (um conjunto de conceitos e suas relações entre si), projectada para ser compartilhada por diversos agentes com vários objectivos [7]. Se forem construídos a partir de uma ontologia válida, os Topic Maps oferecem, às aplicações que os usam, interoperabilidade semântica entre os vários recursos descrito por cada topic map.

TM é um formalismo para representar conhecimento acerca da estrutura de um conjunto de recursos de informação, organizando-o em *tópicos*. Esses tópicos têm ocorrências em recursos de informação e associações que representam e definem os relacionamentos entre os tópicos. A informação sobre os tópicos pode ser inferida ao examinar as associações e ocorrências ligadas ao tópico. Desta forma, um *topic maps* fornece-nos um ponto de vista sobre uma colecção de recursos [14], disponibilizando conhecimento sobre determinado assunto organizado formalmente por tópicos que se ligam entre si e às partes relevantes dessas fontes de informação.

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. *Charles Goldfarb* (o pai das linguagens de anotação) geralmente compara topic maps com GPS (*Global Positioning System*) aplicado ao universo da informação [5]. Falar sobre Topic Maps é falar sobre estrutura de conhecimento.

Formado na sua essência por um conjunto restrito e muito intuitivo de conceitos—*tópico, associação, papel, ocorrência*—o TM tem uma característica muito importante: *permitindo criar um mapa virtual da informação, o TM mantém as fontes na sua forma original, sem modificação*. Então, o mesmo recurso de informação pode ser usado de diferentes formas, por diferentes mapas de tópicos, conquistando-se assim, facilmente, a *reutilização* dos recursos de informação.

Introduzida a definição e os ingredientes que compõem um TM, há necessidade de encontrar uma notação precisa e universal para o descrever. Conforme será defendido em [9], há de momento várias alternativas plausíveis, tais como XTM – XML Topic Maps [15], HyTM [10], AsTMa= [2], ou LTM [4]. Destas só a XTM é um standard (as outras foram criadas por distintos fabricantes para os seus sistemas específicos). Por isso e pelos vários argumentos apresentados na próxima subsecção, essa tem sido a sintaxe adoptada noutros e neste projecto.

3.1 XML Topic Maps

Como a maioria das vezes os recursos de informação acima referidos estão anotados em XML, é fácil verificar que os Topic Maps podem também ser expressos usando XML, retirando-se daí inúmeras vantagens. Para isto, um grupo de investigadores, liderados por Steve Pepper e Graham Moore, definiu a linguagem XTM, *XML Topic Maps*, criando o *TopicMaps.org*. *TopicMaps.Org*[3] é uma associação independente, formada por vários grupos de pesquisa nesta área, para desenvolver e divulgar a linguagem e procurar a sua aplicabilidade à Internet, tirando partido das características da família de especificação XML. Assim surgiu, portanto, XTM como linguagem XML para descrição, manipulação e intercâmbio de Topic Maps, onde diferentes *elementos* são usados para representar tópicos, ocorrências de tópicos, e associações entre os tópicos, conforme a sintaxe estabelecida formalmente por um DTD [16]. Abaixo mostra-se o esqueleto de um possível documento XTM:

```
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
          xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="id-topic">
        ...
    </topic>
    ...
    <association>
        ...
    </association>
</topicMap>
```

A validação sintáctica de um topic map, escrito no formato XTM, é realizada por um qualquer parser XML porque a sua estrutura é regulada por um DTD (definido, como se disse acima, no documento oficial [16]). Esta definição cria um dialecto XML que contempla todos os conceitos estabelecidos na norma Topic Maps [3].

4 Mapas de Conceitos em Topic Maps

A semelhança entre um mapa de conceitos e uma ontologia e o facto de os Topic Maps estarem, actualmente, a ser usados para exprimir ontologias (conforme se afirmou na secção anterior), levou-nos a pensar em *usar TM para mostrar e explorar MC na Internet*.

Para fazer a transposição de Mapas de Conceitos (apresentados na secção 2) para Topic Maps (introduzidos na Secção 3), usando a sintaxe XTM (também discutida na Secção anterior, ver 3.1), procedemos de modo sistemático como se vai explicar através da apresentação de um Caso de Estudo (CE): *Topic Map para representar o Mapa de Conceitos de uma disciplina de Introdução à Informática*.

O CE escolhido está associado a um projecto, no contexto concreto da disciplina de Introdução à Informática de uma licenciatura em Ciências Sociais, que nasceu da necessidade de criar um sítio WWW moderno para apoio eficiente e eficaz ao ensino presencial tradicional. Pretendia-se usar a clássica "*página WWW*

[3] http://www.topicmaps.org

da disciplina" (elaborada para satisfazer os requisitos oficiais[4] do "Dossier de Disciplina") de modo a construir esse suplemento educativo com suporte na Web. O novo sítio seguirá integralmente, em termos de interface e funcionalidade, o actualmente existente[5] apenas diferindo do actual pelo facto de vir a ser gerado dinamicamente a partir de uma base de dados com toda a informação sobre a disciplina. Nesse âmbito, decidiu-se integrar no sitio um mapa de conceitos descrevendo o conteúdo curricular da disciplina. Para isso necessitávamos de um visualizador que permitisse navegar sobre o mapa e, a partir dele, aceder aos recursos de informação disponibilizados para suportar o dito conteúdo curricular.

4.1 Caso de Estudo

Para criar o *Topic Map para representar o Mapa de Conceitos de uma disciplina de Introdução à Informática*, o primeiro passo consistiu na análise paralela do mapa de conceitos fornecido[6] e do esquema da sintaxe XTM a usar.

Depois, procedeu-se metodicamente a um processo de produção formado por 4 etapas, que se discutem a seguir:

1. escolha da raiz (ou símbolo inicial) do TM;
2. definição de todos os tópicos que correspondem aos conceitos do MC dado;
3. definição de todos os tópicos necessários (papéis e tipos de associações) para depois explicitar as associações que correspondem às relações do MC dado;
4. definição das ocorrências dos tópicos que ligam os conceitos do MC aos recursos de informação disponíveis[7].

Para começar, o conceito principal (no topo da hierarquia), `IIpCS` – *Introdução à Informática para as Ciências Sociais*, que se pretende caracterizar com o MC dado, foi representado como raiz do Topic Map (identificador `first`); esse será o tópico inicial que irá aparecer como título da Página de Entrada, e em destaque, no navegador que vai ser gerado. Dá-se então inicio ao documento XTM da seguinte forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topicMap SYSTEM "xtm1.dtd">
<topicMap id="first" xmlns="http://www.topicmaps.org/xtm/1.0/"
                     xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="IIpCS">
        <subjectIdentity>
            <subjectIndicatorRef xlink:href="#first"/>
        </subjectIdentity>
        <baseName>
            <baseNameString>Introdução à Informática para Ciências Sociais</baseNameString>
        </baseName>
    </topic>
    ......................
</topicMap>
```

[4] Há vários anos em vigor na Universidade do Minho.
[5] Acessível em www.di.uminho.pt/ prh/IIA04/.
[6] Na forma de um grafo desenhado em Visio, ocupando 5 páginas A4 cheias.
[7] A ordem entre esta 4ª e a 3ª fases é arbitrária.

De seguida entra-se na 2ª fase: definição dos tópicos. Para isso percorremos o MC para identificar todos os conceitos e analisar as relações entre cada um e seus adjacentes, pois já nesta fase é preciso verificar qual o tipo da relação que existe. Se um tópico é pai de outro—noção de classe/subclasse criada pela relação *é-um*—o tópico filho terá de ser introduzido como *instância* do outro, conforme se ilustra abaixo com o tópico `unidade1`, *Processamento de Documentos*, que *é-um* tópico `unidade`, *Unidade Lectiva*.

Note-se que neste caso particular da relação hierárquica entre conceitos, não se usa uma associação para a materializar, mas sim recorrer-se ao elemento `instanceOf`.

```
<topic id="unidade1">
    <instanceOf>
        <topicRef xlink:href="#unidade"/>
    </instanceOf>
    <baseName>
        <baseNameString>Processamento de Documentos</baseNameString>
    </baseName>
</topic>
<topic id="unidade">
    <baseName>
        <baseNameString>Unidade Lectiva</baseNameString>
    </baseName>
</topic>
```

Para todos os tópicos que não sejam subclasse de outro (não tenham pai), como é o caso do tópico `unidade`, apenas se define o seu identificador único, através do atributo `id` do elemento `topic` e o respectivo `basename` que contem a *string* que será visualizada no Web Browser; tal como `unidade`, os tópicos `documento` e `computador`, definidos abaixo, ilustram esta situação:

```
<topic id="documento">
        <baseName>
            <baseNameString>Documento</baseNameString>
        </baseName>
</topic>
<topic id="computador">
        <baseName>
            <baseNameString>Computador</baseNameString>
        </baseName>
</topic>

```

Quando todos os conceitos estiverem representados por tópicos, passa-se à 3ª fase: definição das associações. Agora procede-se em 3 passos:

1. primeiro definem-se os tópicos correspondentes aos papéis de cada elemento de uma associação;
2. depois definem-se os tópicos que correspondem aos tipos de associações que vão ser usadas;
3. por fim declaram-se as associações como instâncias dos tópicos anteriores

Suponha-se, então, que o tópico `unidade1`, *Processamento de Documentos*, envolve dois outros tópicos, `computador` e `documento` definidos acima, e fixemo-nos na relação que une estes dois últimos.

Para definir a associação `é-processado`, que liga o tópico `documento` ao tópico `computador`, começamos por identificar os papéis de cada um dos membros: `processado` desempenhado por `documento`; e `processa` desempenhado por `computador`. Os papéis também são vistos como tópicos, e assim sendo terão também de ser definidos.

```
<topic id="processado">
        <baseName>
            <baseNameString>é processado por</baseNameString>
        </baseName>
</topic>

<topic id="processa">
        <baseName>
            <baseNameString>processa</baseNameString>
        </baseName>
</topic>
```

O tipo que caracteriza a associação em causa será designado por `assoc_doc_comp` e, visto que também é um tópico, tem de ser igualmente definido:

```
<topic id="assoc_doc_comp">
    <baseName>
        <baseNameString>
            Associação entre documento e computador
        </baseNameString>
    </baseName>
    <baseName>
        <scope>
            <topicRef xlink:href="#processado"/>
        </scope>
        <baseNameString>é processado por</baseNameString>
    </baseName>
    <baseName>
        <scope>
            <topicRef xlink:href="#processa"/>
        </scope>
        <baseNameString>processa</baseNameString>
    </baseName>
</topic>
```

Agora é, finalmente, possível introduzir a associação `é-processado` como instância do tipo `assoc_doc_comp` anterior:

```
<association>
    <instanceOf>
        <topicRef xlink:href="#assoc_doc_comp"/>
    </instanceOf>
    <member>
        <roleSpec>
            <topicRef xlink:href="#processado"/>
        </roleSpec>
        <topicRef xlink:href="#documento"/>
    </member>
    <member>
        <roleSpec>
            <topicRef xlink:href="#processa"/>
        </roleSpec>
        <topicRef xlink:href="#computador"/>
    </member>
</association>
```

Por fim, definem-se as ocorrências dos tópicos, realizando-se assim a 4ª etapa que colmata o processo. Esta etape é fundamental visto que serão estas definições que vão permitir, mais tarde a nível do navegador produzido, aceder aos recursos de informação a partir dos conceitos representados no mapa.

Porém, para concretizar esta tarefa é necessário mais um cuidado: é preciso criar um novo tópico por cada tipo de recurso que se queira usar como ocorrência.

No CE em apreço, os recursos que se utilizaram foram todos documentos em HTML (na integra, ou suas partes) pertencentes aos sumários e material de apoio disponibilizado, pela equipa docente, no sítio WWW da disciplina. Por esta razão, decidiu-se designar este tipo de fontes de informação por *URL*, sendo então necessário definir o tópico respectivo, `url`:

```
<topic id="url">
    <baseName>
        <baseNameString>URL (Uniform Resource Locator)</baseNameString>
    </baseName>
</topic>
```

Agora a definição das ocorrências pode ser feita sem mais dificuldades, como se exemplifica abaixo para o caso do tópico `bit` que é conectado à respectiva entrada no documento `Dicionario.htm`[8].
Note-se que esta definição de ocorrência é acrescentada à definição do tópico respectivo.

```
<topic id="bit">
    <baseName>
        <baseNameString>Bit</baseNameString>
    </baseName>
    <occurrence>
        <instanceOf>
            <topicRef xlink:href="#url"/>
        </instanceOf>
        <resourceRef
         xlink:href="http://www.di.uminho.pt/~gepl/IIA04/Dicionario/Dicionario.htm#bit"/>
    </occurrence>
</topic>
```

Seguindo esta linha sistemática de pensamento, construiu-se toda a especificação XTM que traduz o Mapa de Conceitos fornecido à partida para a disciplina de Introdução à Informática, tendo-se obtido uma descrição com 5360 linhas e um total de 353 tópicos, 122 associações e 55 ocorrências.

5 Geração do Navegador sobre o Mapa de Conceitos

Uma vez criada a especificação XTM é altura de gerar automaticamente o navegador pretendido (consoante se explicou na introdução ao caso de estudo, na secção anterior). Para o efeito usou-se, numa 1ªfase, o Omnigator—a escolha mais directa e óbvia. Contudo, para geração da solução final recorreu-se a uma

[8] O qual contém uma definição sucinta para cada um dos conceitos básicos do conteúdo curricular da disciplina em causa.

outra ferramenta (criada no seio do nosso grupo de investigação[9], o Ulisses, por razões que serão justificadas à frente.

Actualmente, o Ontopia Omnigator talvez seja o processador de XTM mais difundido para manipular Topic Maps. O Omnigator é uma aplicação que permite carregar e navegar sobre qualquer topic map, usando um browser para a Web. O objectivo do desenvolvimento do Omnigator foi incentivar o uso de Topic Maps, ensinando os princípios básicos do paradigma.

O Omnigator faz parte do Ontopia Knowledge Suite (OKS) [12]. Isto implica que algumas funcionalidades do Omnigator foram projectadas para trabalhar em conjunto com outras ferramentas, tais como o Ontopia Navigator Framework. Contudo o Omnigator é o único módulo integrante do OKS fornecido livremente[10]; os demais módulos devem ser adquiridos.

Apesar deste inconveniente comercial, o Omnigator é considerado uma ferramenta completa e eficiente, que produz um navegador com uma interface muito agradável e simples de manusear. Contudo, o Omnigator é um interpretador—processa o topic map quando carrega a especificação XTM indicada—o que requer a sua presença na máquina do utilizador final, ou então o recurso a um servidor que terá de estar activo numa determinada máquina.

Por esta razão constata-se, na prática, que é uma óptima solução para a fase de desenvolvimento, quando o topic map está constantemente a ser alterado[11]. Nessa fase, a navegação proporcionada pelo Omnigator pode ser utilizada para certas verificações, como por exemplo, a correcção dos nomes dos tópicos e dos seus contextos. Porém, numa situação de produção, em que o topic map está estável, é preferível usar uma solução em que o visualizador possa ser aberto por qualquer *browser* sem requerer a presença do processador.

Foi precisamente este requisito que nos levou a usar o Ulisses que, sendo comparável a um compilador, gera um conjunto de páginas HTML estáticas. Assim, o Ulisses é invocado uma vez (pela equipa responsável pelo desenvolvimento do navegador) e o resultado[12] permite efectuar navegações com qualquer browser que o utilizador final escolha.

O Ulisses [8] é um gerador de navegadores conceptuais desenvolvido no contexto do Metamorphosis[13], mas que pode ser usado isoladamente (tal como sucede com o Omnigator, como acima explicado). O Metamorphosis é formado por um conjunto de linguagens de especificação e ferramentas que permitem criar uma interface para integração de informação oriunda de diversas fontes, através do uso de uma ontologia que será representada em Topic Maps. A partir da descrição das fontes heterogéneas de informação e da especificação da ontologia, uma das ferramentas extraí automaticamente o respectivo topic map. Depois de guardado (no formato de um documento XTM, ou numa base de dados)

[9] Ver http://www.di.uminho.pt/ gepl/

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

[11] como ele interpreta a especificação na hora em que o serviço é requisitado, qualquer alteração que tenha sido produzida no ficheiro XTM é logo reflectida no visualizador.

[12] Experimentável em http://www.di.uminho.pt/ gepl/IIpCS/.

[13] Descrição completa e detalhada em: http://www.di.uminho.pt/ gepl/metamorphosis.

este topic map será validado sintáctica e semanticamente (face a um conjunto de restrições especificadas numa linguagem apropriada) por uma segunda ferramenta. Por fim, a componente Ulisses gera, a partir da descrição XTM válida, uma interface Web para manipular o topic map extraído.

Como se disse, o Ulisses não se limita ao contexto do Metamorphosis; isto significa que ele não está restrito aos topic maps obtidos através do processo de extracção ocorrido no Metamorphosis.

Assim, o Ulisses é visto como um gerador de *sítios Web* completos (em HTML com algumas componentes em Javascript), desenvolvido em XSL e usando, para esse propósito, qualquer topic map que siga a sintaxe XTM. Apesar do seu processo sistemático de transformação, através de um processador standard de XSLT, o Ulisses oferece opções que permitem rápidas modificações em todo o website, como por exemplo, alterações em termos de layout (cores e tamanho das fontes), comentários a serem inseridos nas páginas criadas e a inserção de imagens.

Usando a especificação XTM, que se obteve conforme descrito na secção anterior (ver 4.1, o gerador Ulisses produziu automaticamente e com eficiência o conjunto de páginas HTML que formam o visualizador pretendido para o Mapa de Conceitos do caso de estudo descrito. As páginas geradas propiciam, em qualquer browser da Web, uma navegação conceptual fácil através dos conceitos apresentados e dos recursos ligados. Na Figura 3 vê-se uma das páginas desse navegador.

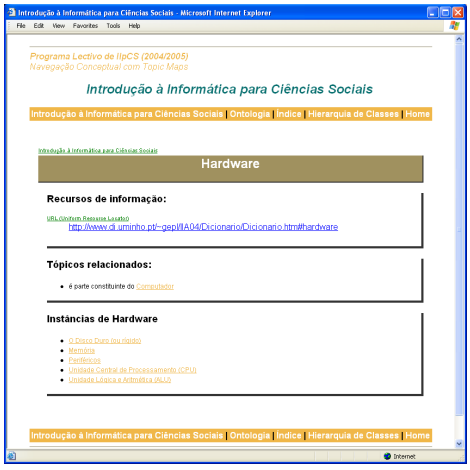


Fig. 3. Uma página do navegador gerado pelo Ulisses, apresentando o conceito "Hardware".

A página ilustrada pela Figura 3 diz respeito ao conceito *Hardware* apresentando, por baixo do tópico, os três grupos de elementos principais que o caracterizam:

– as ocorrências, neste caso apenas um recurso de informação acessível através do URL indicado;
– tópicos associados (por coincidência também só um), indicando o seu (*Hardware*) papel de actuação em relação ao outro tópico;
– e as suas instâncias, neste caso cinco.

Ainda se encontra, na parte superior da página, referência ao tópico do qual *Hardware* é uma instância. Qualquer tópico referenciado permite aceder directamente à sua própria definição (página com a mesma estrutura desta).

Para além desta informação, específica do nosso Mapa de Conceitos particular, qualquer página criada pelo Ulisses inclui navegadores genéricos que permitem aceder: à Página Principal do navegador; à Ontologia implícita no topic maps (sendo ainda disponibilizadas várias opções de consulta); a um índice alfabético de todos os tópicos; e à visualização gráfica do topic map (neste caso coincidiria com o Mapa de Conceitos).

6 Conclusão

Com o intuito de apetrechar um sítio WWW dinâmico de apoio a uma disciplina de Introdução à Informática com a descrição do seu conteúdo curricular feita através de um Mapa de Conceitos, decidimos lançar mão aos Topic Maps e às tecnologias associadas associadas ao seu processamento.

Para isso, estudámos com cuidado a própria definição de MC (bem semelhante à noção de ontologia), analisámos a informação nele contida e identificámos os objectivos que se pretendem atingir ao associar tal instrumento ao dossier de uma disciplina.

Confrontando os resultados dessa análise com o conceito de TM (actualmente em voga para descrever ontologias) e com a funcionalidade disponível, concluímos que o caminho a seguir, na prossecução do intuito didático acima exposto, passaria pelo uso de norma XTM para descrever o MC criado.

Procurámos então sistematizar a tarefa de transcrição dos conceitos e relações do MC para tópicos e associações do TM, como única via para obter rapidamente, com segurança e sem grande custo, a descrição completa do mapa de conceitos da Introdução à Informática, fornecido esquematicamente (na forma de um grafo).

Uma vez obtida a especificação XTM, recorremos a ferramentas com que tínhamos grande familiaridade, nomeadamente o Omnigator e o Ulisses, para construir mecanicamente o navegador Web pretendido.

Os resultados, alcançados em metade do tempo previsto, mostram o sucesso da solução tecnológica.

O impacto pedagógico da ideia e a exploração de caminhos diversos de utilização deste objecto de ensino na sala de aula, constituem o trabalho futuro.

References

1. David P. Ausubel. *Educational Psychology, A Cognitive View.* New York: Holt, Rinehart and Winston, Inc, 1968.

2. Robert Barta. AsTMa= Language Definition. Bond University, TR., 2004. http://astma.it.bond.edu.au/astma=-spec-xtm.dbk.
3. Michel Biezunsky, Martin Bryan, and Steve Newcomb. ISO/IEC 13250 - Topic Maps. ISO/IEC JTC 1/SC34, December, 1999. http://www.y12.doe.gov/sgml/sc34/document/0129.pdf.
4. Lars Marius Garshol. LTM – The Linear Topic Map Notation. Ontopia, 2002. http://www.ontopia.net/topicmaps/ltm.html.
5. Charles F. Goldfarb and Paul Prescod. *XML Handbook*. Prentice Hall, 4th edition, 2001.
6. Thomas R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.
7. Nicola Guarino and P. Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. Ed. Amsterdan: ISO Press, 1995.
8. Giovani R. Librelotto, José C. Ramalho, and Pedro R. Henriques. Ontology driven Websites with Topic Maps. In *The International Conference on Web Engineering*, Oviedo, Spain, 2003.
9. Giovani Rubert Librelotto. *XML Topic Maps: da Sintaxe à Semântica*. PhD thesis, Departamento de Informática, Universidade do Minho, 2005. A ser publicado em breve.
10. Steven R. Newcomb, Michel Biezunski, and Martin Bryan. The HyTime Topic Maps (HyTM) Syntax 1.0. ISO/IEC JTC 1/SC34 N0391, 2003. http://www.jtc1sc34.org/repository/0391.htm.
11. Joseph D. Novak. *A Theory of education*. Ithaca, N.Y., Cornell. University Press, 1977.
12. Ontopia. The Ontopia Knowledge Suite. http://www.ontopia.net/solutions/products.html, February, 2004.
13. José Augusto Pacheco, Maria Palmira Alvez, Maria Assunção Flores, João M. Paraskeva, José Carlos Morgado, Ana Maria Silva, and Isabel Carvalho Serra. *Componentes do Processo de Desenvolvimento do Currículo*. Colecção Minho Universitária, Livraria Minho, 1999.
14. Steve Pepper. The TAO of Topic Maps - finding the way in the age of infoglut. Ontopia, 2000. http://www.ontopia.net/topicmaps/materials/tao.html.
15. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/.
16. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0 - Annex D: XTM 1.0 Document Type Declaration (Normative). TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/#dtd.

The use of Taxonomies as a way to achieve Interoperability and improved Resource Discovery in DSpace-based Repositories

Miguel Ferreira, Ana Alice Baptista
(mferreira@dsi.uminho.pt, analice@dsi.uminho.pt)

Department of Information Systems,
University of Minho,
4800-Guimarães
Portugal

Abstract

In this paper, we present the ongoing work being developed at University of Minho in the context of Institutional Repositories, more precisely the ones based on the DSpace platform (developed jointly by the Massachusetts Institute of Technology and Hewlett-Packard). As part of our research, we have created an add-on for DSpace to ensure authority control over the keywords that human cataloguers may use to describe their items of information. These keywords are used by the visitors of the repository for searching and browsing the catalogue. The keywords are organised in a taxonomy that results from the combination of several specialised thesauri, one for each community of users (e.g., Computing, Engineering, Architecture, etc.). Each of these off-the-shelf thesauri is described by a simple XML file. The first thesaurus we have imported into our system was the publicly-available *Association for Computing Machinery (ACM) Computing Classification System (CCS)*. The success of the add-on exceeded our expectations given that there was a broad acceptance by the community of users, probably driven by its simplicity and ease of use. ACM contributed to our work by validating our XML version of the CCS and by publishing it on their Web site. Our most recent endeavour is centred on the conversion of the ACM CCS to OWL (Web Ontology Language). Through the use of taxonomies we expect to achieve better interoperability between analogous systems as well as improve the discovery of resources in our repository. Future work will be focused on the improvement of the add-on to support more complex structures, such as thesauri or ontologies.

Keywords: Controlled Vocabularies, Taxonomy, Thesaurus, Ontology, Institutional Repositories, DSpace.

Introduction

As the amount of globally accessible information grows, so does the inherent difficulty in finding desired items of information[1] [1, 2]. In the context of document archives, one of the most applied techniques to facilitate the discovery of items, both in traditional manual systems as in newer computerized systems, has been *indexing* [2].

Indexing consists in the assignment of values to predefined attributes to serve as a basis for searching [2, 3] and resource discovery. The combination of these attributes and values should constitute sufficient information to successfully characterise the contents of a document and enable the future retrieval of that document by solely looking at this information [2]. Examples of commonly found attributes are: *author*, *title*, *subject*, *abstract*, etc. These are generally referred to as *metadata*.

Metadata can be attained either automatically or manually. Fully automated methods generally result in the creation of a full-text catalogue, i.e., an index containing the most frequent words found in the document [4]. Other automatic procedures analyse the contents of the document and attempt to assign pre-established concepts to metadata attributes [2, 5].

Manual approaches generally consist in scanning a document for keywords that are considered relevant or more adequate to describe the subject of the document. These keywords may be selected from a predefined set of keywords, also know as controlled vocabulary, or self-thought by the person who is indexing document. Manual indexing usually results in better quality metadata which will later reflect the precision and recall of search results [2]; however, manual indexing, especially the one carried out by trained professionals is highly expensive, time-consuming and evermore unfeasible given the amount of information being produced today. Some repository systems combine the two approaches [2] and use manually inserted metadata as well as automatically generated full-text indexes to facilitate the discovery of items.

Given the ever-growing call for quality metadata and the amount of information being produced today combined with the insufficient funding necessary to finance professional indexing, a new paradigm has emerged: the *self-archiving* [2]. Self-archiving means that the producer of a document is the main responsible for the creation of its metadata. Doing it so, the task of

[1] An information item is defined broadly as an object that contains information. The representation can be in different media types such as text, image, video, etc. An information item is also referred to as a *document* inside the corpus of this paper.

creating quality metadata resorting to human expertise becomes less onerous as the task is distributed by several people. This change in the paradigm implies that documents will now be indexed by amateurs who know little or nothing about indexing [2].

The keywords chosen by users to describe their works can be extremely variable resulting in a poor inter-indexer consistency [2]. This results in an increased difficulty in finding similar items. Furnas, et al [6] showed that the probability of two amateur indexers using the same keywords to describe a specific object is less than 20%. It is crucial that both manual and automatic indexing procedures use controlled vocabularies to standardise document descriptions and to simplify subsequent searches by establishing a common language in a given domain.

In this paper we describe an ongoing project that consists in the implementation of a cross-domain controlled vocabulary used to describe self-archived items in DSpace-based repositories.

This paper is organised as follows: section 2 attempts to shed light on some of the concepts that will be used throughout the corpus of this paper; section 3 provides a quick overview on the genesis of institutional repositories; section 4 describes some of the work being developed at University of Minho in the context of institutional repositories; section 5 explains in detail how we implemented an add-on that enhances DSpace with controlled vocabularies; on section 6 we draw some conclusions about our work and discuss some proposals for future work.

Taxonomies, Thesauri and Ontologies

For the sake of clarity, we felt important to include a section in this paper to describe some of the concepts that will be used throughout the corpus of this piece.

Taxonomy

Taxonomy consists in a subject-based classification that arranges the terms in a controlled vocabulary into a hierarchy without doing anything further [7].

Almost anything – animate objects, inanimate objects, places, and events – may be classified according to some taxonomic scheme [8].

Mathematically, a taxonomy is a tree structure of classifications for a given set of objects. At the top of this structure is a single classification – the root node – that applies to all objects. Nodes below this root are more specific classifications that apply to subsets of the total set of classified objects [8].

Hence, a taxonomy is a collection of terms used to describe *things* that are grouped together in a tree structure. We are able to identify parent-child relationships between the terms in the controlled vocabulary.

Thesaurus

Thesauri basically take taxonomies as described above and extend them to make them better able to describe the world by not only allowing subjects to be arranged in a hierarchy, but also allowing other statements to be made about the subjects [7].

The following properties and relationships are incremented by thesauri:

- *Scope Note* – A string property attached to the term explaining its meaning within the thesaurus.
- *Use* – Refers to another term that is to be preferred instead of a certain term; implies that the terms are synonymous.
- *Related Term* – Refers to a term that is related to a given term, without being a synonym or a broader/narrower[2] concept.

Ontology

With ontologies the creator of the subject description language is allowed to define the language at will. Ontologies in computer science came out of artificial intelligence, and have generally been closely associated with logical inferencing and similar techniques, but have recently begun to be applied to information retrieval [7].

Ontologies extend the concept of thesaurus by enabling the creator of the controlled vocabulary to define new properties and relationships between terms.

Institutional repositories

In recent years, the development of institutional repositories has emerged as a new strategy for the preservation, publishing and dissemination of scholarly communications [9]. Universities and other research institutions throughout the world are actively planning and implementing institutional repositories aiming at providing its members a set of new services such as the archiving of research results (article preprints and post-prints, theses, and dissertations);

[2] Parent-child relationship.

management of digital collections; preservation of digital documents; housing of teaching materials and electronic publishing of journals and books [10]. Institutional repositories also fulfil the important task of levering scholars from the burden of administering their own publishing system (i.e. personal Web site) [9].

The digital repository genesis has been short, beginning in the late 2000 when the UK's University of Southampton released a software package called EPrints [10]. Since then, the movement to establish digital repositories has gained momentum, encouraged by a convergence of dropping costs for online storage, the proliferation of broadband networking technologies and the development of metadata standards to describe repository content [9, 10].

Other initiatives, such as the Open Archives Initiative (OAI) [11] sustain the general acceptance and proliferation of institutional repositories. The OAI is a collaborative effort towards the development and promotion of standards and solutions such as the OAI Protocol for Metadata Harvesting [12], which allows an institution to create descriptive metadata for the items in its custody and making it available to others who wish to use it [9, 10].

As well as EPrints, other repository systems have been developed by different organisations. DSpace is a good example of a general-purpose repository designed to capture the intellectual output of research organizations that is rapidly gaining wide-range acceptance, especially by universities and research institutes.

The development of DSpace is a responsibility of the MIT Library and Hewlett-Packard. Unlike EPrints, DSpace supports the ingestion of a wide range of digital material types [10]. The system itself does not present any restrictions on the formats that should be accepted; although, such restrictions can be set up by the administrator of the repository. In addition, DSpace is open-source allowing everyone in the community to contribute by building original enhancements and customisations.

DSpace Development at University of Minho

The University of Minho (UMinho) was the first institution in the Portuguese speaking world to use a translated version of DSpace. DSpace related activities at University of Minho started in April 2003 and since then many developments have been made.

The first step has been taken by the UMinho Documentation Services (SDUM) with the translation to Portuguese of the entire DSpace system and

its implementation in the RepositóriUM[3]. This version of DSpace has been downloaded for use in many other institutions in Portugal and Brazil.

This same version was used as a basis for the Papadocs[4] system. Papadocs was first created to provide access to all assignments made by students of the Department of Information Systems. Some changes had to be made to the original version, in particular to what concerns the metadata. Additional fields in the area of education were appended to the basic Dublin Core [13] element set. Examples of newly created fields are:

- *Creator.Identifier* – i.e., student's id number;
- *Contributer.Teacher* – i.e., the identification of the main teacher responsible for a particular assignment or discipline.
- *Grade* – i.e., assignment classification;

This customized version of DSpace served as test-bed for a series of add-ons that enhance the platform in many ways. A Web site called *DSpace-Dev @ University of Minho*[5] was created with the purpose of sharing these add-ons with the interested community and generate discussion about other research projects currently in hands. All the source-code can be downloaded on this Web site.

The following sections provide a short description of some of the add-ons that resulted from this project.

Commenting Add-on

The Commenting Add-on consists in a set of classes, servlets and custom tags that bring informal communication capabilities to the DSpace environment. The informal communication is assured by a threaded forum that can be attached to any DSpace resource: web-page, community, collection, submitted item or e-person.

Recommendation Add-on

The Recommendation Add-on consists in a set of custom-tags that provide suggestions of resources related to a given selected resource. The most relevant custom-tag receives a parameter identifying the selected resource (type and id) to which the suggestions/recommendations shall be given. The

[3] http://repositorium.sdum.uminho.pt
[4] http://papadocs.dsi.uminho.pt
[5] http://dspace-dev.dsi.uminho.pt

add-on iterates through the database collecting items, e-persons and comments that are considered to be relevant.

Web of Communication Add-On

The 3D Web of Communication allows the user to discover hidden relationships between items, comments and people. It works by displaying a VRML 3D web of resources involved in a communication process. The user is also able to jump to specific items on the environment thus providing a 3D navigational system over DSpace

Controlled Vocabulary Add-on

Many repository systems favour self-archiving and DSpace is no exception. Users are stimulated to submit their works to the repository and generate themselves the appropriate descriptive metadata. Users that submit items to the repository are allegedly rewarded by an increased visibility of their work.

In most archiving scenarios, it is natural that a certain degree of ambiguity and heterogeneity will be found in the metadata provided by different users to documents with similar content. This can also be observed in archives where items have been indexed by trained professionals. To downsize this problem, we have developed an add-on for DSpace that restricts the keywords that users may employ during indexing stages of self-archiving.

During submission, users are asked to enter the keyword(s) that best describe their works. With our add-on in place, users are presented with a taxonomy that displays the terms that are allowed to be used as descriptors. For each community of users that interact with the repository a different taxonomy is presented. Each of these taxonomies is rendered to the user as an expandable tree (see Figure 1).

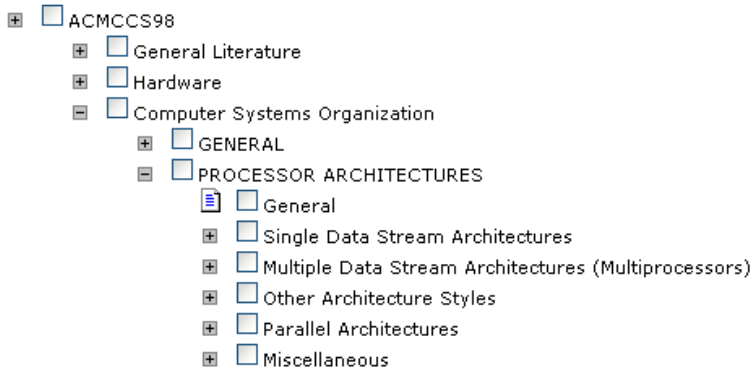


Figure 1: Excerpt of the ACM Computing Classification System taxonomy presented to the user.

The development and maintenance of these domain-specific taxonomies is not our main concern. We have elected publicly available classification systems, one per each scientific community, to be used in the repository. Since these are highly used classification systems, the interoperability between similar repositories is simplified as the probability of finding other systems that use the same controlled vocabularies becomes higher.

DSpace is compatible with the Open Archives Initiative Protocol for Metadata Harvesting [12], a protocol that allows the creation of centralised catalogues of metadata to facilitate the discovery of items in physically distributed repositories. In this context, an agreement on the set of keywords used to describe the items in custody is of considerable importance.

The first controlled vocabulary we have imported into our system was the ACM Computing Classification System (1998 version) [14]. This controlled vocabulary is being used by the students of the Department of Information Systems to describe their academic projects. Recent contacts with other departments also interested in publishing their students' projects have resulted in the opening of the Papadocs repository to the Civil Engineering and Architecture communities. This event conducted to the adoption of two other taxonomies appropriate to describe the items submitted by members of these communities – we are now using a sub-set of the Engineering Index Thesaurus [15] and negotiating the possibility of using the Art & Architecture Thesaurus [16].

How does it work?

The add-on works by loading all the included taxonomies from independent XML files (stored on the server's file system) and rendering them as trees to

the user. The structure of these XML files is very straightforward. We use four different elements to represent the whole structure of the taxonomies: *node*, which contains information about a specific term; *isComposedBy*, a wrapper element that contains a list of child nodes; *isRelatedTo*, an element that contains links to other related nodes in the taxonomy; and *hasNote*, an element that allows the inclusion of a small descriptive note about the term (see Figure 2).

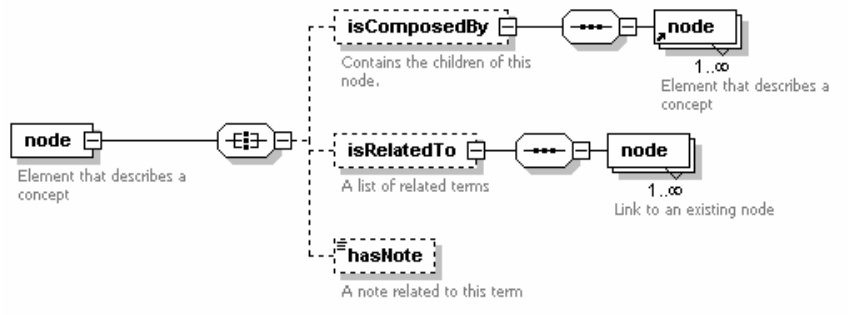


Figure 2: Schema that validates our XML taxonomy.

As we can see, this schema goes beyond the representation of simple taxonomies. The relationship *isRelatedTo* and the property *hasNote* allow the description of thesauri according to the ISO standard 2788:1986 [17]. However, at the moment our rendering system does not process these relationships so the user is presented with a limited view of the thesaurus actually described in the XML file. The reason for this approach is not technological, but social. We wanted our classification system to be simple and user-friendly to students that are not used to manipulate complex structures. Adding further dimensions to the taxonomy would probably steer users away from its use.

In Figure 3 is shown a small sample of the ACM Computing Classification System compliant with our schema.

```
<node id="acmccs98" label="ACMCCS98">
   <isComposedBy>
      <node id="A." label="General Literature">
         <isComposedBy>
            <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
            <node id="A.2" label="REFERENCE (e.g., dictionaries, encyclopedias, glossaries)"/>
            <node id="A.m" label="MISCELLANEOUS"/>
         </isComposedBy>
         <hasNote type="2">
            The classification is no longer used as of January 1998,
            but the item is still searchable for previously classified documents.
         </hasNote>
         <isRelatedTo>
            <node id="D.3.2"/>
         </isRelatedTo>
      </node>
   </isComposedBy>
</node>
```

Figure 3: A sample of the ACM CCS taxonomy in XML.

Finding items in the repository

As stated previously, the keywords used to describe the submitted items are selected from a tree of terms. When a term is picked from this tree, the full-path between the root node and selected node is used to represent the chosen keyword. For example, a book on the programming language Java could generally be described by the following keywords:

- ACMCCS98/Software/PROGRAMMING TECHNIQUES/Object-oriented Programming
- ACMCCS98/General Literature

The advantage of this approach is twofold. It removes the ambiguity inherent to certain concepts by accompanying them with the correct context and allows the realisation of more general queries. For example, the book described above will be included in the list that results from querying the repository for items whose subject is *"ACMCCS98/Software/ PROGRAMMING TECHNIQUES"*. The concept matching is accomplished easily due to the fact that the most general concept is a mere substring of the most specific descriptor.

Conclusions and Future work

In this paper we present an Add-on for DSpace that enables repository administrators to compel its users to use a controlled set of keywords to describe self-archived items of information. The advantages of using controlled vocabularies are enumerated throughout the corpus of this document.

It is our belief that the introduced controlled vocabulary system is adequate to our objectives due its simplicity. Users can easily find the terms they are looking for by expanding just a few branches of the taxonomy. Further research should be performed to make sure our beliefs are truthful.

Much can be done in the future to improve the add-on. First, we could render the XML as a true thesaurus by exposing the *isRelatedTo* relationships as links to the user. Secondly, we could upgrade the thesaurus model to support other types of relationships and/or properties. This would mean start using ontologies to describe concepts and their relationships. In certain contexts this would be very useful, but in the context of our repository, the augmented number of relationships and complexity introduced by this new class of structures could be dissuasive for most users.

Interesting work could also be developed in the area of automatic cataloguing. The add-on could be enhanced to suggest keywords to the user by analysing the contents of the document being submitted or by comparing it with other documents already in the repository.

If the system keeps on being adopted by different communities, we will soon come to a state where some branches of the incorporated taxonomies will overlap. When this happens, we must start using techniques to merge taxonomies [18].

The main purpose of the Papadocs repository was to serve as a test-bed for the research we are performing in the field of institutional repositories. We have come to a point where some of the technology we have produced is being considered to be included in the RepositoriUM – the official institutional repository of University of Minho – where it will be used by a greater number of users. The efficiency and the scalability of our solutions are now being subject to a higher degree of consideration. Furthermore, some of the add-ons we have developed are being considered for inclusion in the official DSpace source-code.

It is also worth noting that our XML version of the ACM Computing Classification System is now being used by ACM it self and is publicly

available for download on their Web site[6]. Our most recent endeavour is centred in the conversion of the ACM CCS from our XML format to OWL (Web Ontology Language) [19] in order to make it suitable for a greater number of users.

References

[1] D. Teixeira, M. Ferreira, and V. Verhaegh, "An Integrated Framework for Supporting Photo Viewing Activities in Home Environments," presented at European Symposium on Ambient Intelligence, Eindhoven, The Netherlands, 2003.

[2] Y.-M. Chung, W. M. Pottenger, and B. R. Schatz, "Automatic Subject Indexing Using an Associative Neural Network," presented at 3rd ACM International Conference on Digital Libraries (DL'98), 1998.

[3] C. Meadow, "Text Information Retrieval Systems," *Academic Press Inc.*, 1992.

[4] D. Cunliffe, C. Taylor, and D. Tudhope, "Query-based Navigation in Semantically Indexed Hypermedia," presented at Conference on Hypertext, 1997.

[5] K. Taghva, T. A. Nartker, and J. Borsack, "Recognize, Categorize, and Retrieve," presented at Symposium on Document Image Understand Technology, 2001.

[6] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary Problem in Human-System Communication," *Communications of the ACM*, vol. 30, pp. 964-971, 1987.

[7] L. M. Garshol, "Metadata? Thesauri? Taxonomies? Topic Maps!," Ontopia, 2004.

[8] Wikipedia, "Taxonomy," 2004.

[9] C. A. Lynch, "Institutional Repositories: Essential Infrastructure for Scholarship in Digital Age," *ARL Bimonthly Report*, 2003.

[10] H. F. Cervone, "The Repository Adventure," *Library Journal*, 2004.

[11] OAI, "Open Archives Initiative," vol. 2004.

[6] http://www.acm.org/class/1998/

[12] OAI, "The Open Archives Initiative Protocol for Metadata Harvesting," vol. 2004, 2002.

[13] DCMI, "Dublin Core Metadata Initiative," vol. 2004.

[14] N. Coulter, J. French, E. Glinert, T. Horton, N. Mead, R. Rada, A. Ralston, C. Rodkin, B. Rous, A. Tucker, P. Wegner, E. Weiss, and C. Wierzbicki, "The ACM Computing Classification System [1998 Version]," vol. 2004: Association for Computing Machinery, 1998.

[15] E. E. I. Inc., "Ei Thesaurus," 1998.

[16] G. R. Institute, "Art & Architecture Thesaurus," vol. 2004, 2000.

[17] ISO, "Guidelines for the establishment and development of monolingual thesauri, International Organization for Standardization - ISO 2788:1986," 1986.

[18] M. Sintichakis and P. Constantopoulos, "A Method for Monolingual Thesauri Merging," *Communications of the ACM*, 1997.

[19] W3C, "Web Ontology Language (OWL)," 2004.

Obtendo Interoperabilidade Semântica em Sistemas Heterogéneos de Informação com Metamorphosis

Giovani Rubert Librelotto, José Carlos Ramalho, and Pedro Rangel Henriques

Universidade do Minho, Departamento de Informática
4710-057, Braga, Portugal
grl@di.uminho.pt

Abstract. Hoje em dia, os dados manipulados pelas instituições estão dispersos nos mais variados recursos de informação, tais como bases de dados e documentos de diferentes tipos. Com isso, a integração da informação contida nessas fontes é uma tarefa árdua.
Diferentes abordagens podem ser seguidas para conseguir a integração, porém o problema principal será conquistar a interoperabilidade semântica entre as fontes de informação, garantindo a manipulação do conjunto sem forçar a conversão das partes em um formato único. Ontologias surgem como uma solução para este problema e Topic Maps, como uma forma de representação de ontologias, afirmam-se como um caminho para alcançar esta solução.
Neste contexto, este artigo apresenta o Metamorphosis – um ambiente orientado a Topic Maps composto por módulos capazes de extrair dados de recursos heterogéneos de informação, construir uma ontologia a partir dos mesmos (baseado em uma especificação), validá-la e gerar um sistema de navegação conceptual – que permite atingir a interoperabilidade semântica desejada.

1 Introdução

Diariamente, uma grande quantidade de dados é produzida em instituições. Para satisfazer os requisitos de armazenamento, estas organizações geralmente utilizam bases de dados relacionais, as quais são eficientes para guardar e manipular dados estruturados. Quando se trata de dados semi-estruturados, o armazenamento é realizado em documentos textuais ou anotados.

Há um problema quando estas organizações requerem uma visão integrada desses sistemas heterogéneos de informação, pois é necessário interrogar/extrair cada fonte de dados. Contudo, o acesso para cada sistema de informação é diferente. Nesta situação, há a necessidade de uma abordagem que permita extrair a informação contida nestes recursos e a disponibilize como uma visão única sobre todo o sistema.

Topic Maps são uma boa solução para organizar conceitos e os relacionar entre si, porque eles seguem uma notação normalizada – ISO/IEC 13250 [2] –

para a representação de conhecimento intercambiável. Topic Maps são compostos por tópicos e associações dando origem a uma rede semântica estruturada que concentra a informação relacionada com um certo domínio. Esta rede hierárquica de tópicos pode representar uma ontologia. Esta é a razão pela qual se usa, há alguns anos, esta tecnologia para classificação e integração de documentos na área de Arquivos Digitais, com sucesso.

Contudo, o processo de desenvolvimento de ontologias baseado em Topic Maps é complexo, consome muito tempo e requer uma quantidade significativa de recursos humanos e financeiros, porque elas podem envolver um enorme conjunto de tópicos (conceitos) e associações (relacionamentos entre os conceitos).

Para resolver este problema, propomos o sistema *Metamorphosis* que torna possível a extracção, validação, armazenamento e navegação de Topic Maps. Ele é composto por três módulos principais: (1) *Oveia*, que extrai dados de sistemas heterogéneos de informação, de acordo com a especificação de uma ontologia e armazena-a em um topic map; (2) *XTche*, que valida o topic map gerado, de acordo com uma especificação de restrições; e (3) *Ulisses*, que navega sobre o topic map, dando uma visão conceptual integrada sobre o domínio e os recursos.

Deste modo, o *Metamorphosis* permite obter a interoperabilidade semântica em sistemas heterogéneos de informação porque os dados relevantes são extraídos e armazenados em um topic map, de acordo com uma especificação da ontologia desejada. O ambiente valida o topic map gerado de acordo com um conjunto de regras definido numa linguagem para descrição de restrições. Este topic map fornece fragmentos de informação (os dados propriamente ditos) conectados por relações específicas para outros conceitos, em diferentes níveis de abstracção. A navegação sobre o topic map é realizada sobre uma rede semântica e proporciona uma visão homogénea sobre os recursos – o que justifica a decisão de chamar a este processo de *interoperabilidade semântica*.

Habilitando a criação de um mapa virtual de informação, os recursos de informação são mantidos em seus estados originais; ou seja, não são modificados. Então, um mesmo recurso pode ser usado pelo *Metamorphosis* de diferentes formas, para a criação de diferentes topic maps. Com isso, a reutilização de fontes de informação é conquistada.

A secção 2 apresenta uma visão geral sobre Topic Maps. A arquitectura proposta e suas características principais encontram-se descritas na secção 3; as seguintes secções descrevem os módulos principais de *Metamorphosis*: *Oveia* (secção 4), *XTche* (secção 5) e *Ulisses* (secção 6). Por fim, a conclusão fornece uma síntese deste artigo.

2 Topic Maps

Topic Maps – ISO/IEC 13250 [2] – é uma norma para organização e representação de conhecimento sobre um domínio, a qual permite a especificação de temas e de relacionamentos entre temas. Steve Pepper [13] define *tema* como um termo usado para designar alguma coisa do mundo real, ou seja, algo que

um tópico possa representar. Portanto, um *tópico* pode ser qualquer coisa: uma pessoa ou objecto, uma entidade ou uma organização, um conceito, etc.

Um topic map é composto por tópicos e associações que dão origem a uma rede semântica estruturada que agrupa informações relacionadas sobre um certo domínio.

Um topic map pode ser visto como um conjunto organizado de tópicos (representação formal de temas), contendo:

- uma estrutura hierárquica de tópicos (definido pelas relações *é-um* ou *contém* – relações classe-instância);
- vários nomes para cada tópico (ou tema de um índice);
- ponteiros (ocorrências) entre tópicos e documentos externos (recursos de informação);
- relacionamentos semânticos (associações) entre tópicos.

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. A informação sobre cada tópico pode ser inferida ao examinar as associações e ocorrências ligadas a esse tópico.

Topic Maps podem ser expressados em XML, através da especificação XML Topic Maps (XTM) 1.0 [14], a qual é um dialecto para escrita de topic maps e foi desenvolvido para aplicar a norma Topic Maps [2] para a web.

3 Metamorphosis

A principal ideia do *Metamorphosis* é integrar a especificação de redes de conceitos ou ontologias, com sua navegação e armazenamento, assim como sua extracção automática e sua validação, a partir de recursos heterogéneos de informação.

A motivação para o desenvolvimento do *Metamorphosis* também veio de duas situações que surgiram no contexto de alguns projectos de desenvolvimento de software:

- Muitas vezes, para se testar algumas funcionalidades de um sistema que se está a desenvolver é necessário criar uma interface Web para realizar esses testes. Normalmente, estas interfaces não têm grandes requisitos quanto à aparência, o que se pretende é que sejam desenvolvidas o mais rapidamente possível.
- Quer-se expor na Web um sistema de informação. Este sistema é composto por bases de dados, documentos XML, documentos PDF, etc. Quer-se que toda a informação esteja acessível via Web e para isso constroem-se os primeiros índices. Estes índices acabam por ser enormes excedendo a capacidade dos browsers actuais. Podia pensar-se em fraccioná-los alfabeticamente, mas há situações em que isso não é possível nem recomendável. Mas,

é sempre possível arranjar um método para fraccionar a informação conceptualmente. É aqui que se começa a discutir a organização dos recursos de informação e é aqui que se introduz o *Metamorphosis*.

O *Metamorphosis* toma como entrada:

Recursos de Informação: compostos por um ou mais recursos de dados: documentos XML, páginas HTML, bases de dados, etc. O *Metamorphosis* não modifica nenhuma fonte, apenas usa parte de seus dados para construir a rede semântica, através de uma ontologia definida para tais fontes de dados;

Especificações XML: a descrição das fontes de dados (escrita em XSDS – *XML Specification for DataSources*); a descrição da ontologia a ser construída (escrita em XS4TM – *XML Specification for Topic Maps*); e a descrição das regras a serem obedecidas por um topic map (estrita em XTche – *Topic Maps Schema and Constraint Language*).

e gera, como saída:

Um Website Conceptual: O website gerado permite a navegação através do sistema de informação dirigido por conceitos organizados em uma rede semântica.

A figura 1 vem reforçar esta ideia e dá uma visão pictórica da arquitectura do *Metamorphosis*.

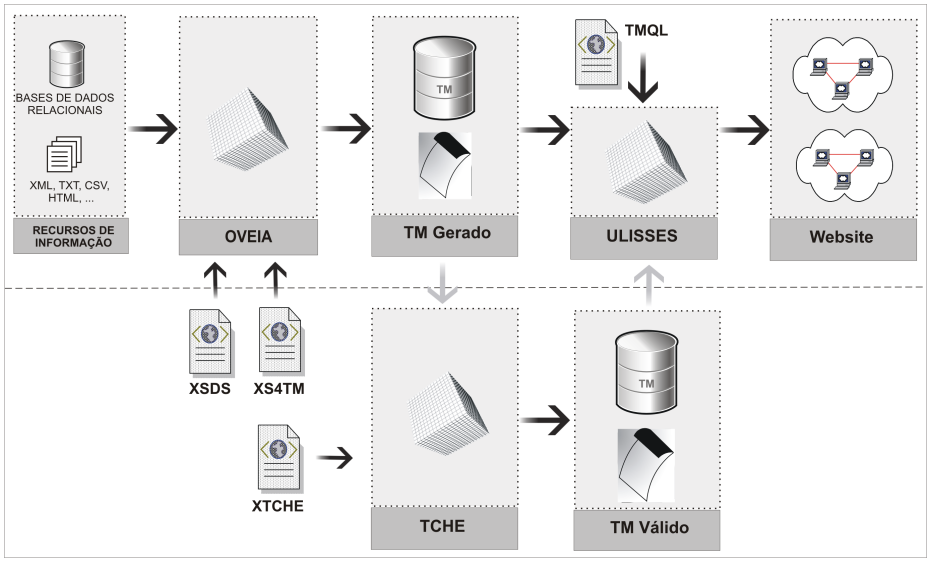


Fig. 1. Metamorphosis

Esta arquitectura pode ser descrita da seguinte forma:

(1) Camada de recursos de informação: Este componente é composto pelos vários recursos de informação: documentos XML, páginas Web, bases de dados, ... O *Metamorphosis* não interfere com nenhum deles, apenas utiliza parte da informação de cada um para construir a ontologia ou rede semântica.

(2) Especificações XSDS e XS4TM: São documentos XML que fornecem informações precisas sobre onde aceder para extrair as partes da informação necessárias para a construção da ontologia; este componente é descrito em detalhes em [9].

(3) Oveia: Este componente utiliza a especificação XS4TM para ir aos recursos de informação buscar a informação de que necessita para construir a ontologia. O *Oveia* [9] é um extractor de topic maps em sistemas heterogéneos de informação.

(4) TM gerado: Este é o topic map de acordo com a sintaxe *XML Topic Maps*. Além da possibilidade de armazenamento dos topic maps em formato XTM, os topic maps gerados pelo *Oveia* também podem ser armazenados em uma representação relacional. Para isso, a norma Topic Maps foi mapeada para um modelo relacional; este modelo é chamado de *BD Ontologia*.

(5) Especificação XTche: É uma linguagem de especificação de restrições para topic maps, baseada nos requisitos de TMCL (*Topic Map Constraint Language*) [10] definidos pela ISO/IEC. Permite especificar regras para a validação semântica de topic maps.

(6) Processador XTche: Este componente é responsável pela validação de topic maps de acordo com um conjunto de restrições especificado em XTche. Se os topic maps a serem validados estiverem de acordo com a especificação XTche, nada acontece; caso apresentem alguma irregularidade, mensagens de erros serão mostradas, indicando os pontos em que o topic map não está correto.

(7) Ulisses: Toma como entrada um *topic map* e produz uma visualização na web, de acordo com algumas regras. O *Ulisses* tanto fornece a navegação conceptual a partir da sintaxe XTM, como a partir do modelo relacional apresentado no *Oveia*.

(8) Website: É o website gerado através do qual é possível navegar semanticamente pelos vários recursos de informação que compõem o sistema de informação original.

Nas próximas secções, os três módulos principais desta arquitectura são discutidos: *Oveia* (secção 4), *XTche* (secção 5) e *Ulisses* (secção 6).

O modo de funcionamento do *Metamorphosis* permite uma interdependência entre tais módulos; assim, quando o utilizador for efectuar o processamento de um conjunto de recursos de informação de acordo com especificações XSDS, XS4TM e XTche, cada etapa é realizada em separado:

1. no *Oveia*, define-se os ficheiros com as especificações das fontes de informação e da ontologia a ser aplicada. Após o seu processamento, um sumário indica o tempo total e a quantidade de tópicos e associações extraídos que serão encontrados no topic map obtido;

2. no *XTche*, indica-se o nome dos ficheiros que possuem o topic map a ser validado e o conjunto de restrições a ser aplicado. O resultado do processamento é a confirmação (ou não) da validação do topic map;
3. no *Ulisses*, basta informar o topic map que será a fonte para a construção do website semântico.

O funcionamento independente dos módulos permite que uma tarefa específica seja refeita, caso necessário.

4 Oveia – Um Extractor de Topic Maps

O *Oveia* é um extractor de ontologias baseadas em Topic Maps a partir de sistemas heterogéneos de informação.

Sua arquitectura é composta por duas especificações e os referentes processadores, conforme apresentada na Figura 2: a primeira, escrita na linguagem XSDS (*XML Specification for DataSources/DataSets*), especifica os dados a serem extraídos das fontes de informação; enquanto que a segunda, escrita na linguagem XS4TM (*XML Specification for Topic Maps*), é responsável por declarar como são construídos os topic maps. Com base nestas especificações, o extractor busca as informações nas fontes de informação e produz um topic map. Este topic map gerado pode ser armazenado em formato XTM (XML Topic Maps) ou em uma base de dados relacional.

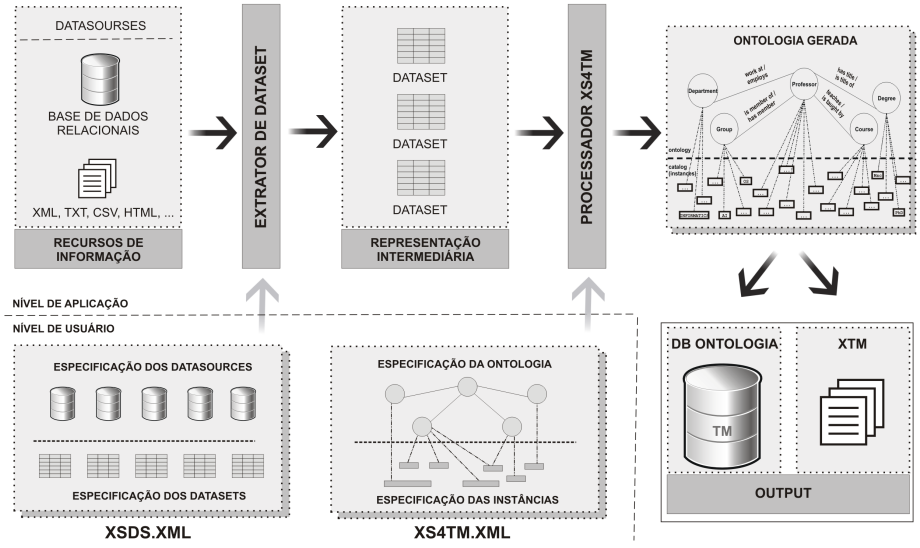


Fig. 2. Oveia

4.1 XSDS – *XML Specification for DataSources*

A linguagem XSDS (*XML Specification for DataSources/DataSets*) – apresentada em detalhes em [9] – foi definida com o intuito de especificar que fontes de informação fornecerão dados para a criação de topic maps, de acordo com uma ontologia posteriormente especificada. Essa especificação fornece todos os mecanismos necessários para especificar as fontes de dados passíveis de extracção de informação. Assim, os dados extraídos são armazenados em *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 registo 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.

Para efeitos de demonstração da sintaxe de XSDS, apresenta-se o código abaixo:

```
<resources>
    <datasources>
        <datasource extratorDriver="br.uneb.dcet.tmbuilder.drivers.XMLFile" name="XML_DI">
            <parameter name="pathDocument">D:\\UMINHO\\DI.xml</parameter>
        </datasource>
    </datasources>
    <datasets>
        <dataset name="Pessoas" database="XML_DI">
            <columns>
                <column identify="true">CodPes</column>
                <column>nome</column>
            </columns>
            <statement>//Pessoas</statement>
        </dataset>
    </datasets>
</resources>
```

Na prática, 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*.

4.2 O Extrator DS2DS

O *Extractor 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 seleccionados nos elementos *datasets* da especificação XSDS em questão.

O *Extrator DS2DS* possui diversos *drivers* de extracção que são os módulos responsáveis pela extracção de informação das fontes de dados; portanto, há um *driver* desenvolvido para cada tipo de recurso de informação.

4.3 XS4TM – XML Specification for Topic Maps

A linguagem XS4TM – também apresentada em detalhes em [9] – tem por objectivo especificar a construção de Topic Maps a partir dos dados contidos nos *datasets*. XS4TM torna a definição da extracção de Topic Maps mais completa e flexível, pois ela caracteriza-se por transformar a norma XTM em um subconjunto da sua especificação; em outras palavras, XS4TM estende a linguagem XTM, como pode ser percebido neste extracto de código XS4TM abaixo:

```
<xstm xmlns="http://www.topicmaps.org/xtm/1.0/" xmlns:xlink="http://www.w3.org/1999/xlink">
    <ontologies>
        <topic id="Pessoas">
            <baseName>
                <baseNameString>Pessoas</baseNameString>
            </baseName>
        </topic>
        ...
    </ontologies>
    <instances>
        <topic dataset="Pessoas">
            <instanceOf>
                <topicRef xlink:href="#Pessoas"/>
            </instanceOf>
            <baseName>
                <baseNameString>@Nome</baseNameString>
            </baseName>
        </topic>
        ...
    </instances>
</xtche>
```

A especificação XS4TM é subdividida em duas partes distintas, as quais seguem o esquema de XTM 1.0 DTD:

Ontologia: nesta primeira parte são declarados os elementos responsáveis pela definição da ontologia, como os tipos de tópicos, os tipos de associações, ou qualquer outra definição de acordo com o modelo XTM que possa ser utilizada para expressar a estrutura de conhecimento a ser extraída – ou seja, a hierarquia de tópicos, relações super-tipo/sub-tipo e associações entre os tópicos;

Instâncias: nesta segunda parte representa as instâncias de tópicos e associações. Nesse momento, os `Datasets` serão utilizados para expressar quais recursos de informação fornecerão dados para a construção de cada tópico e associação.

A linguagem XS4TM pretende tornar a especificação de extracção de Topic Maps mais flexível. Para isso, ela utiliza a sintaxe de XTM para seus dois elementos principais: <Ontologies> e <Instances>; isto significa que tanto a declaração da ontologia, quanto das instâncias, seguem o formato XTM: uma sequência dos elementos <topic> e <association>. Portanto, se o utilizador conhece a sintaxe de XTM, ele terá muita facilidade em definir uma especificação XS4TM.

4.4 Processador de XS4TM

Este componente utiliza a especificação XS4TM para seleccionar 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.

4.5 Base de Dados de Ontologias

Uma das diferenças desta ferramenta é o armazenamento dos Topic Maps extraídos em uma base de dados relacional. Baseado nos métodos de mapeamento de documentos XML para o modelo relacional apresentados em [16], adoptou-se na *BD Ontologia* o modelo de mapeamento por estrutura.

Conforme o mapeamento por estrutura se caracteriza, 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.

A facilidade de compreensão desse modelo é garantida principalmente pelo facto de que este segue a norma XTM, a qual é bastante conhecida pela comunidade académica. 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.

5 XTche – Um Validador Semântico para Topic Maps

Esta secção apresenta uma linguagem para a definição de restrições sobre Topic Maps, chamada XTche [8], a qual permite garantir que um conjunto de topic maps de uma mesma família são semanticamente válidos de acordo com uma especificação. Antes de descrever a linguagem e seu processador (um gerador de validador), será dado uma motivação de seu desenvolvimento, além de uma discussão sobre restrições.

5.1 Uma Linguagem para definição de Esquemas e Restrições em Topic Maps

No desenvolvimento de topic maps reais é altamente conveniente utilizar um sistema para validá-los; isto é, verificar a correcção de uma instância real de acordo com uma especificação formal de uma respectiva família de topic maps (de acordo com a intenção de seu criador).

Adoptando-se o formato XTM, a validação sintáctica de um topic map é garantida por um parser XML porque a estrutura de XTM é definida por um

DTD [15]. Contudo, sabe-se que validação estrutural não significa uma completa correcção – a semântica também deve ser garantida.

Usando um XML Schema para a validação de Topic Maps, ao invés de DTD, aprimora-se o processo de validação porque alguns requisitos de semântica (domínio, número de ocorrências, etc.) podem ser adicionados à especificação estrutural; parsers XML tratam desta tarefa. Contudo, outros requisitos de semântica permanecem não-especificados. Assim, torna-se necessária uma linguagem de especificação que permite a definição de esquema e restrições numa família de Topic Maps.

Uma lista de requisitos para uma nova linguagem foi recentemente estabelecida pelo *ISO Working Group* – projecto ISO JTC1 SC34 – para uma *Topic Map Constraint Language* (TMCL) [11]. Esta lista é exaustiva e cobre praticamente todos os objectos de Topic Maps.

A linguagem XTche atende todos os requisitos desta lista; para este propósito, XTche possui um conjunto de construtores para a descrição de restrições em Topic Maps, o qual será detalhado nas próximas sub-secções.

Porém, a novidade da proposta é que a linguagem permite a definição de estrutura para topic maps em um estilo baseado em XML Schema; portanto, não é mais necessário uma descrição sintáctica em separado. Uma especificação XTche une o esquema (definição da estrutura e da semântica básica) com restrições (descrevendo a semântica contextual) para todos os topic maps de uma família em particular.

Cabe ressaltar que as especificações XTche definidas de acordo com a sintaxe XML Schema não poderão ser usadas directamente por um parser XML para validar um topic map em formato XTM. Isto porque, apesar de usar tal sintaxe, o código XTche possui apenas a definição das restrições a ser aplicadas a topic maps, portanto necessita ser processado para a obtenção de um validador específico.

O uso da sintaxe XML Schema justifica-se, basicamente, por possibilitar uma edição e visualização gráfica da especificação XTche (pois há um vasto conjunto de ferramentas que possibilitam a edição gráfica de XML Schema) e por possuir uma sintaxe bem conhecida pela comunidade XML.

5.2 Processador XTche e TM-Validator

As frases escritas na linguagem XTche, referentes a um conjunto de restrições – listando as condições (envolvendo tópicos e associações) que devem ser verificadas – especificam um processo de validação em topic maps (um *TM-Validator*).

Aliado ao facto de XTche ser uma linguagem XML, o formato de intercâmbio mais utilizado pela comunidade académica é o XTM; portanto, tanto a especificação, quanto os topic maps, são anotados em XML. Com isso, é possível habilitar a codificação sistemática, em XSL, desta tarefa de verificação.

Entende-se que a partir destas circunstâncias, é possível gerar automaticamente um validador para cada especificação. Para este propósito, desenvolveu-se o *Processador XTche* (uma folha de estilos XSL) que transforma uma especificação XTche em um *TM-Validator* (outra folha de estilos XSL).

O *Processador XTche* é o gerador de *TM-Validator*; comporta-se precisamente como um gerador de compilador e é o núcleo desta arquitectura, como pode ser visto na Figura 3. Ele toma uma especificação de esquema e restrições em Topic Maps válida (um documento XML escrito de acordo com a linguagem XTche) e gera o *TM-Validator* (uma folha de estilos XSL) que irá processar o topic map para validá-lo ou gerar mensagens de erros, à medida que os mesmos forem sendo detectados. A resposta do *TM-Validator* é a confirmação de que o topic map fornecido como entrada cumpre, ou não, as restrições definidas na especificação XTche.

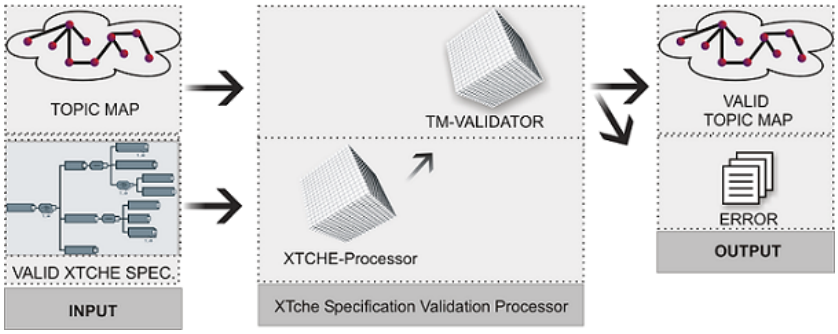


Fig. 3. Arquitectura de XTche. Fonte: [8].

Ambas folhas de estilo XSL (o gerador e o validador) são interpretados por um processador XSL padrão, como o Saxon[1], o que é um dos benefícios desta proposta.

6 Ulisses – Um Navegador Conceptual baseado em Topic Maps

Para navegar em uma ontologia expressada em Topic Maps, esta deve ser vista como um grafo, onde os nodos representam os tópicos, as ligações entre os nodos representam associações e as ligações para os recursos físicos de informação representam as ocorrências. Portanto, a partir de um grafo o processo de criação de um website para a navegação é óbvio: uma página para cada conceito (não somente o nome de cada tópico, mas com todas as suas características) associadas com os nodos relacionados, onde cada ligação representa uma ligação para outra página. Assim, quando uma ligação é escolhida, uma página relacionada é visualizada, apresentando outro conceito ou um recurso de informação.

A ideia principal da navegação conceptual pode ser descrita como: quando se está posicionado sobre um certo conceito, a ferramenta de navegação mostrará

[1] http://saxon.sourceforge.net/

as informações associadas a este conceito em particular; se for escolhido algum dos outros conceitos relacionados, a posição muda para o conceito e a visão muda de acordo com a escolha; se for escolhido algum dos recursos de informação, o sistema mostrará o próprio recurso. Esta, portanto, é a ideia sobre a qual se baseou o *Ulisses*.

Assim como as outras duas ferramentas apresentadas (*Oveia* e *XTche*), *Ulisses* – um navegador conceptual – é um processador genérico que pode ser usado independentemente. O *Ulisses* permite navegação sobre topic maps gerados por ferramentas (como o *Oveia*) ou gerados manualmente. Adicionalmente, estes topic maps podem ser validados semanticamente ou não, de acordo com uma especificação *XTche*.

Este browser ainda suporta navegação tanto sobre documentos XTM, como sobre a *BDOntologia*. Assim, quando o topic map estiver no formato XTM, este componente navegacional é implementado como um conjunto de transformações XML através de folhas de estilo XSL; quando a fonte é a *BDOntologia*, ferramentas baseadas em SQL apropriadas são utilizadas para navegar sobre ela.

7 Conclusão

Este artigo descreve a integração de sistemas heterogéneos de informação usando o paradigma de Ontologias, para gerar uma visão homogénea destes recursos. A proposta é um ambiente, chamado *Metamorphosis*, para a construção automática de Topic Maps com dados extraídos de várias fontes de dados e a disponibilização de uma navegação semântica sobre a informação extraída.

Embora o *Metamorphosis* tenha sido desenvolvido para o uso em nossas áreas de trabalho principais – processamento de documentos XML aplicado a Arquivos Públicos e Museus Virtuais – estamos convencidos que o *Metamorphosis* pode ser aplicado com sucesso na área geral de sistemas de informação para a integração de dados, análise e exploração de conhecimento.

O *Metamorphosis* tem sido utilizado em vários projectos de pequena e média dimensão. As interfaces Web são criadas rapidamente e sem grandes dificuldades – em termos de especificação – por parte dos utilizadores.

Em termos de trabalhos relacionados, até o presente momento não é do conhecimento dos autores uma ferramenta com as mesmas características do *Metamorphosis*. Contudo, pode-se comparar, individualmente, cada um dos seus módulos com ferramentas conhecidas pela comunidade académica. Uma comparação entre o *Oveia*, o TSIMMIS [4] e o KAON REVERSE [5] é apresentada em [9]. O *XTche* foi comparado em [8] com o *AsTMa!* [1] e com uma proposta de Eric Freese [3]. Por fim, uma comparação entre o *Ulisses* e o Ontopia Omnigator [12] será encontrada em [6].

Um caso de estudo sobre um departamento académico foi apresentado em [9], o qual descreveu a geração de um topic map a partir de uma base de dados MySQL de acordo com especificações XSDS e XS4TM. O resultado final, armazenado em uma *BDOntologia* em Microsoft SQL Server 2000, tinha 4420 tópicos e 4223 associações; o mesmo topic map teria aproximadamente 172490

linhas no formato XTM. O *Ulisses* fornece uma navegação completa sobre o topic map gerado. O tempo médio para a extracção de cada elemento (tópico ou associação) foi de 0,1311 segundos.

Para mais detalhes sobre o *Metamorphosis* e seus módulos, recomenda-se a leitura de [7–9].

References

1. Robert Barta. AsTMa! Bond University, TR., 2003. http://astma.it.bond.edu.au/constraining.xsp.
2. Michel Biezunsky, Martin Bryan, and Steve Newcomb. ISO/IEC 13250 - Topic Maps. ISO/IEC JTC 1/SC34, December, 1999. http://www.y12.doe.gov/sgml/sc34/document/0129.pdf.
3. Eric Freese. Using DAML+OIL as a Constraint Language for Topic Maps. In *XML Conference and Exposition 2002*. IDEAlliance, 2002. http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-03-03/05-03-03.html.
4. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. In *Journal of Intelligent Information Systems*, volume 8(2), pages 117–132. Kluwer Academic Publishers, 1997.
5. S. Handschuh, A. Maedche, L. Stojanovic, and R. Volz. KAON, 2001. http://kaon.semanticWeb.org/kaon/white-paper.pdf.
6. Giovani Rubert Librelotto. *XML Topic Maps: da Sintaxe à Semântica*. Tese de Doutoramento, Departamento de Informática, Universidade do Minho, 2005. to be published.
7. Giovani Rubert Librelotto, José Carlos Ramalho, and Pedro Rangel Henriques. Extração de topic maps no oveia: Especificação e processamento. In Mauricio Solar, David Fernández-Baca, and Ernesto Cuadros-Vargas, editors, *30ma Conferencia Latinoamericana de Informática (CLEI2004)*, pages 451–460. Sociedad Peruana de Computación, September 2004. ISBN 9972-9876-2-0.
8. Giovani Rubert Librelotto, José Carlos Ramalho, and Pedro Rangel Henriques. XTche - A Topic Maps Schema and Constraint Language. In *XML 2004 Conference and Exposition*, Washington D.C., U.S.A, 2004. IDEAlliance.
9. Giovani Rubert Librelotto, Weber Souza, José Carlos Ramalho, and Pedro Rangel Henriques. Using the Ontology Paradigm to Integrate Information Systems. In *International Conference on Knowledge Engineering and Decision Support*, pages 497–504, Porto, Portugal, 2004.
10. Mary Nishikawa and Graham Moore. Topic Map Constraint Language (TMCL) Requirements and Use Cases. ISO/IEC JTC 1/SC34 N0405rev, 2003. http://www.isotopicmaps.org/tmcl/requirements.html.
11. Mary Nishikawa, Graham Moore, and Dmitry Bogachev. Topic Map Constraint Language (TMCL) Requirements and Use Cases. ISO/IEC JTC 1/SC34 N0405rev, 2004. http://www.jtc1sc34.org/repository/0548.htm.
12. Ontopia. The Ontopia Omnigator, 2002. http://www.ontopia.net/omnigator/.
13. Steve Pepper. The TAO of Topic Maps - finding the way in the age of infoglut. Ontopia, 2000. http://www.ontopia.net/topicmaps/materials/tao.html.
14. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/.

15. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0 - Annex D: XTM 1.0 Document Type Declaration (Normative). TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/#dtd.
16. Kevin Williams, Michael Brundage, Patrick Dengler, Jeff Gabriel, Andy Hoskinson, Michael Kay, Thomas Maxwell, Marcelo Ochoa, Johnny PaPa, and Mohan Vanmane. *Professional XML Databases*. Wrox Press, 2000.

Constraining XML Topic Maps with XTche

Giovani Rubert Librelotto[1], José Carlos Ramalho[1], and
Pedro Rangel Henriques[1]

Universidade do Minho, Departamento de Informática
4710-057, Braga, Portugal
{grl, jcr, prh}@di.uminho.pt

Abstract. This paper presents a process for specifying constraints on topic maps with a constraint language. This language allows to express contextual conditions on classes of Topic Maps. With XTche, a topic map designer defines a set of restrictions that enables to verify if a particular topic map is semantically valid. As the manual checking of large topic maps (frequent in real cases) is impossible, it is mandatory to provide an automatic validator.
The constraining process presented in this paper is composed by a language and a processor. The language is based on XML Schema syntax. The processor is developed in XSLT language. The XTche processor takes a XTche specification and it generates a particular XSLT stylesheet. This stylesheet can validate a specific topic map (or a set of them) according to the constraints in the XTche specification.
In this paper we will show, in abstract terms and with concrete examples, how to specify Topic Maps schemas and constraints with XTche.

1 Introduction

Topic Maps are a standard for organizing and representing knowledge about a specific domain. They allow us to specify subjects and relationships between subjects. Steve Pepper [9] defines subject as the term used for the real world thing that the topic itself stands in for. A topic, in its most generic sense, can be anything whatsoever - a person/object, an entity/organization, a concept - regardless of whether it actually exists or is a mental abstraction [12].

Besides the simplicity and powerfulness of the topic/association-based model, there are two Topic Maps features that are important in the process of understanding and reasoning about a domain: the hierarchical structure that is represented in a map (defined by the relations is-a or contains); and the complementary topic network (made up of other links that connect topics that are not included in each other but just interact).

The facts above explain the importance of Topic Maps to describe knowledge in general; in particular their application to define ontologies is one of the growing up fields. So Topic Maps are nowadays widely used within XML environments: in archives, for cataloging purposes; or in web browsers, for conceptual navigation.

To build reliable systems, like those referred, it is crucial to be sure about the validation of the underlying semantic network.

Like in other fields, as formal language and document processing, it is wise to validate the syntax and semantics of a topic map before its use. This is precisely the focus of this paper: we propose XTche, a language to define Topic Maps Schema and Constraints. The validation process of a topic map based on a XTche specification will also be under the scope of the paper.

In section 2 there is an overview about the basic concepts in the area of this work: Semantic Web, Ontology, and Topic Maps; it creates the context and motivation for our concern with the precise semantics of Topic Maps. Section 3 describes XTche; before the introduction of XTche specific semantic constructors, we distinguish schema and contextual constraints. Then the automatic analysis of a XTche specification (in order to generate a concrete validator) is discussed. Section 4 compares our proposal with related work and exemplifies the use of our constraint language. A synthesis of the paper and hints on future work are presented in the last part, in the section 5.

2 Semantic Web, Ontology, and Topic Maps

Semantic Web is concerned with the arrangement on the Web of information in such way that its meaning can be understood by computers as easily as by people; that is, the web pages contain not only the concrete information to be shown, but also metadata that allows for its semantic interpretation. Such an organization of information offers new perspectives for the Web [6]:

- Greater efficiency and precision in the search for and comprehension of information by users, humans or machines;
- Automatic treatment of information,
- Transfer of simple tasks like search, selection, updating, and transaction from the user to the system.

Organization, *standardization* and *automatic treatment of information* are the key elements that allowed the transition from the *first Web generation*, which is first of all a vast collection of anarchic information, to the *Semantic Web*, which aims at treating decentralized, sharable, and exploitable knowledge.

The Semantic Web requires the cooperation of various disciplines: Ontologies, Artificial Intelligence, Agents, Formal Logic, Languages, Graph Theory and Topology, etc. Our working area is Ontologies for the Web, more exactly, ontologies represented by Topic Maps to be handled by web applications and browsers.

An ontology is a way of describing a shared common understanding, about the kind of objects and relationships which are being talked about, so that communication can happen between people and application systems [13]. In other words, it is the terminology of a domain (it defines the universe of discourse). As a real example consider the thesaurus used to search in a set of similar, but independent, websites.

Ontologies can be used to:

- Create a structured core vocabulary, to be used and validated by a set of actors in a community;

– Define and use logical relationships and rules between the concepts, allowing an efficient use of intelligent agents;
– Develop, maintain, and publish knowledge (that changes rapidly) about an organization (the whole or a part), easily providing different views.

Topic Maps [8] are a good solution to organize concepts, and the relationships between those concepts, because they follow a standard notation – ISO/IEC 13250 [3] – for interchangeable knowledge representation. Topic Maps are composed of topics and associations giving rise to structured semantic network that gathers information concerned with certain domain. This hierarchical topic network can represent an ontology.

A topic map is an organized set of topics (formal representation of subjects), with:

– several names for each topic (or subject of the index);
– pointers (occurrences) between topics and external documents (information resources) that are indexed;
– semantic relationships, whether they are hierarchical or not, between topics via associations;

It also has the capability of supporting multi-classification (a topic can belong to more than one class), and offers a filtering mechanism based on the concept of *scope* that is associated with names, occurrences, and associations.

According to [13], Topic Maps are very well suited to represent ontologies. Ontologies play a key role in many real-world knowledge representation applications, and namely the development of Semantic Web. The ability of Topic Maps to link resources anywhere, and to organize these resources according to a single ontology, will make Topic Maps a key component of the new generation of Web-aware knowledge management solutions.

On one hand, this section helps to understand our interest on Topic Maps in the actually important area of Semantic Web; on the other hand, the concepts so far introduced pointed out the indubitable need for mechanisms to guarantee the semantic correctness of Topic Maps.

3 XTche – A Language for Topic Maps Schema and Constraints

This section presents a language to define constraints on Topic Maps, called XTche. This language allows the topic map semantic validation. Before describing the language and its processor (a validator generator), we give the motivation behind its development, and discuss what a constraint is in this context.

As shown in section 2, when developing real topic maps, it is highly convenient to use a system to validate them; this is, to verify the correctness of an actual instance against the formal specification of the respective family of topic maps (according to the intention of its creator).

Adopting XTM format [10], the syntactic validation of a topic map is assured by any XML parser because XTM structure is defined by a DTD [11]. However, it is well known that structural validity does not mean the complete correctness – semantics should also be guaranteed.

Using XML Schema instead of DTD improves the validation process because some semantic requirements (domain, occurrence number, etc.) can be added to the structural specification. Still XML parsers will deal with that task.

However other semantic requirements remain unspecified. So, a specification language that allows us to define the schema and constraints of a family of Topic Maps is necessary.

A list of requirements for the new language was recently established by the ISO Working Group - the ISO JTC1 SC34 Project for a Topic Map Constraint Language (TMCL) [7]. XTche language meets all the requirements in that list; for that purpose, XTche has a set of constructors to describe constraints in Topic Maps, as will be detailed in the next subsections. But the novelty of the proposal is that the language also permits the definition of the topic map structure in an XML Schema style; it is no more necessary a separate syntactic description. A XTche specification merges the schema (defining the structure and the basic semantics) with constraints (describing the contextual semantics) for all the topic maps in that family.

A Topic Map Schema defines all topic types, scopes, subject indicators, occurrence types, association types, association roles, and association players. So, it is possible to infer a topic map skeleton (written in XTM) from the schema; the user or an application (like Oveia [LSRH04] , a Metamorphosis [LRH03] module) must only fill it in (with data extracted from the information resources) to obtain the topic map instances. This functionality (skeleton derivation and syntactic validation) will not be more developed in this paper, as it is devoted to the semantic aspects.

3.1 Schema Constraints and Contextual Constraints

XTche is designed to allow users to constrain any aspect of a topic map; for instance: topic names and scopes, association members, topics allowed as topic type, roles and players allowed in an association, instances of a topic (enumeration), association in which topics must participate, occurrences cardinality, etc.

These constraints can be divided in two parts: schema constraints and contextual constraints. The first subset defines the Topic Maps Schema (i.e., the structure of topics, associations, and occurrences); the second one is applied over particular conditions in a topic map.

An extensive list of Topic Maps constraints, classified as *schema* or *contextual constraints*, is presented in [5]. Although all the concerns in this list are constraints, there is actually a slight difference in the way of dealing with the two subsets. So, the wish to have XTche expressing both – contextual constraints and schema constraints – has a direct influence in the design of the language and its processing. We will care about that in the following subsections.

3.2 An XML Schema-based language

Like XTM , XTche specifications can be too verbose; that way it is necessary to define constraints in a graphical way with the support of a visual tool. To overcome this problem, XTche syntax follows the XML Schema syntax; so, any XTche constraint specification can be written in a diagrammatic style with a common XML Schema editor.

It is up to the designer to decide how to edit the constraints and schemas: either in a XML Schema visual editor (that outputs the respective textual description), or in an XML text file according to XTche schema. The XTche specification (in textual format) is taken as input by XTche Processor that analyzes and checks it, and generates a Topic Maps validator (TM-Validator) as output (more details in the Section 3.5).

XTche is an XML Schema-based language. All XTche specifications are XML Schema instances; but, obviously, not all XML Schema instances are XTche specifications.

Section 3.2 describes the skeleton for all the XTche specifications. That skeleton is a generic, but incomplete, XML Schema that must be fulfilled with particular constraints for each case, as detailed in Section 3.3 and Section 3.4. To write those constraints, the basic schema language is extended by a set of domain specific attributes that are defined in a separated file (also presented in Section 3.2) imported by the skeleton.

Like any other schema, before processing an XTche specification (in order to generate a TM-Validator), its correctness should be checked. However, an usual XML Schema-based parser is not enough to do that desired validation; we had to extend it with one more layer (to take care of the above referred domain specific attributes) as will be explained in Section 3.2.

XTche Skeleton An XTche specification has a schema where the <`xtche`> element is the root. This element is composed of a sequence, where two elements are allowed: <`schema-constraints`> – that specifies the schema constraints – and <`contextual-constraints`> – that specifies the contextual constraints. Both subelements are optional; it means a specification can only have one kind of constraints. These subelements are composed of a sequence, where each subelement represents a particular constraint.

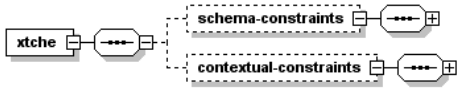


Fig. 1. XTche specification inicial structure

The diagram of Figure 1 represents the code presented below, the generic skeleton above referred (that must be completed in each case). It begins with

root specification, where the namespace `xtche` must be declared with the value http://www.di.uminho.pt/ gepl/xtche.

After that, it is necessary to import the schema that specifies the XTche attributes, as discussed above in the introduction to this section. This schema is available in http://www.di.uminho.pt/ gepl/xtche/xtche-schema.xsd. Finally, a sequence of two non-required elements (contextual-constraints and schema) allows the definition of all the constraints necessary to validate the particular topic maps under definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xtche="http://www.di.uminho.pt/~gepl/xtche" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:import namespace="http://www.di.uminho.pt/~gepl/xtche"
    schemaLocation="http://www.di.uminho.pt/~gepl/xtche/xtche-schema.xsd"/>
    <xs:element name="xtche">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="schema-constraints" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <!-- schema constraint 1 -->
                            <!-- schema constraint 2 -->
                            ...
                            <!-- schema constraint N -->
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="contextual-constraints" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <!-- contextual constraint 1 -->
                            <!-- contextual constraint 2 -->
                            ...
                            <!-- contextual constraint N -->
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The specific XML Schema for XTche attributes (imported by the skeleton above) is found in http://www.di.uminho.pt/ gepl/xtche/xtche-schema.xsd. That schema defines all the attributes required to qualify the elements in an XTche specification.

Those two XML Schemas (the first one incomplete) are all that is necessary to learn the general structure of the new XTche language to define the schema of Topic Maps. To use it, the topic map designer shall also know how to write the constraints he wants to be satisfied by each particular topic map instance. However, before explaining both the schema and contextual constraints, let us just talk about the XTche validation that will guarantee that a particular specification is a well-formed XML-Schema and a valid XTche description.

XTche-Specification Validation Processor XTche-Specification Validation Processor (*XTche-SpecVP*) checks the structure of a XTche-specification in

agreement with the standard schema for XML Schema language and the specific schema for XTche language, presented in last subsection.

About this subject, it is possible to make an analogy between an XTche specification and an XML document: an XTche instance should be a well-formed[1] XML Schema but it also needs to be valid according to XTche schema. So, its correctness is assured by *XTche-SpecVP* that performs separately those two verifications.

Figure 2 depicts that processor, which behavior is: initially, it verifies if the source XTche specification is a valid XML Schema (any XML parser is able to do this simple task); if no errors are found, the processor executes the second step that consists on the verification of its compliance against the rules defined below. Errors are reported as they occur. The XTche specification is correct if no errors are reported.

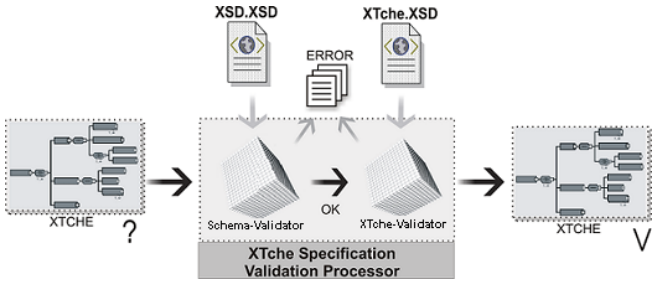


Fig. 2. XTche-Specification Validation Processor

Rules verified by *XTche-SpecVP* in the second phase are described in [5].

3.3 Schema constraint specification

The schema constraint specification follows closely XTM schema. Each schema specification is a subelement of <`schema-constraints`>, the first subelement of <`xtche`>, as shown in the skeleton previously presented. It has several elements structured according XTM schema.

For instance: to specify that topics of type `country` must have occurrence of type `map` in the scope `geography`, we should write the code below:

```
<xs:element name="country">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="map">
            <xs:complexType>
               <xs:sequence>
```

[1] The concept of being well-formed was introduced as a requirement of XML, to deal with the situation where a schema (DTD, XML Schema, or RelaxNG) is not available.

```
                <xs:element name="geography">
                   <xs:complexType>
                      <xs:attribute ref="xtche:occurrenceScope"/>
                   </xs:complexType>
                </xs:element>
             </xs:sequence>
             <xs:attribute ref="xtche:occurrenceType"/>
          </xs:complexType>
       </xs:element>
    </xs:sequence>
    <xs:attribute ref="xtche:topicType"/>
 </xs:complexType>
</xs:element>
```

Figure 3 is the respective diagrammatic view.

Fig. 3. An XTche specification

To compare the XTche specification in Figure 3 and XTM structure, Figure 4 exhibits a part of that schema, where the path to occurrence scope is in contrast.

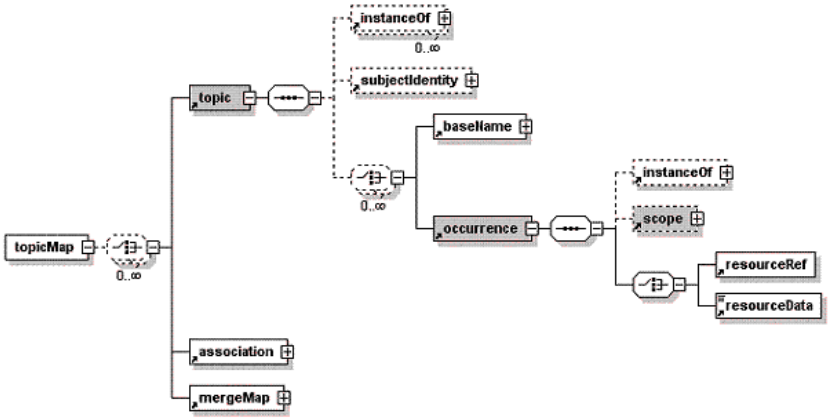


Fig. 4. XTM schema

As shown in Figure 3 one schema constraint is a sequence of concrete topics (`country`, `map`, and `geography`) each one qualified by an associated XTche attribute. A similar description in XTM (Figure 4) uses generic element names (`topic`, `occurrence`, and `scope`) and defines the concrete data via attributes associated to those elements (see code below). This systematic correspondence justifies a previous statement that the XTM code can be inferred from the XTche specification. However, the first contains more semantic information.

```
<topic id="xxx">
    <instanceOf>
        <topicRef xlink:href="#country"/>
    </instanceOf>
    <occurrence>
        <instanceOf>
            <topicRef xlink:href="#map"/>
        </instanceOf>
        <scope>
            <topicRef xlink:href="#geography"/>
        </scope>
    </occurrence>
</topic>
```

A more sophisticated XTche example inspired in the *E-Commerce Application*, subsection 6.1 of [7], is described in [5].

3.4 Contextual constraint specification

Contextual constraints appear in the XTche specification as subelements of <`contextual-constraints`>, the second subelement of <`xtche`>, as explained in subsection 3.2 (see the skeleton shown). They do not have more subelements; they only have attributes.

For instance, to create a topic `person` and say that *it can be used for scoping occurrences and nothing else*, we have to add a <`person`> subelement with an `@occurrenceScope-Exclusive` attribute, as shown in Figure 5.

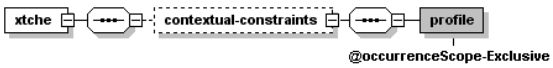


Fig. 5. A contextual constraint specification example

Such a restriction can not be made explicitly in XTM; this is why we call that family of constraints *contextual*, to distinguish from those that can be included in XTM (called *schema-constraints*). This way, to validate the above stated restriction, the TM-Validator needs to check if the topic profile is only used as a `topicRef` element at the end of //occurrence/scope path, as shown in Figure 6.

3.5 XTche Processor and TM-Validator

Each sentence in XTche language – listing all the conditions (involving topics and associations) that must be checked – specifies a *specific topic map validation process* (*TM-Validator*), enabling the systematic codification of this verification task. We understood that those circumstances, it was possible to generate automatically this *TM-Validator*. For that purpose, we developed another XSL stylesheet that translates an XTche specification into the *TM-Validator* XSL code.

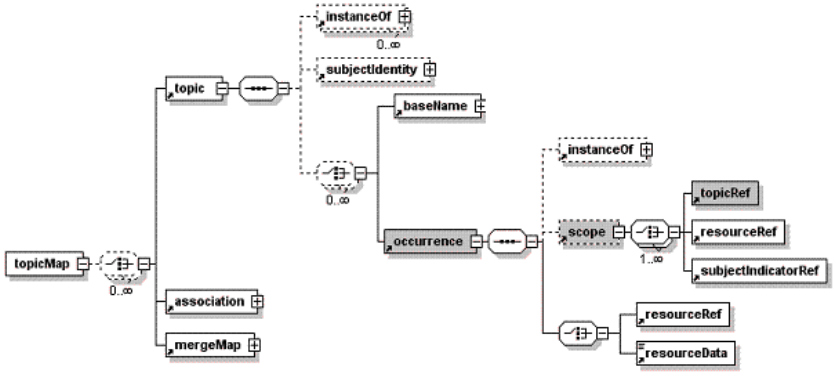


Fig. 6. XTM schema

The XTche processor is the TM-Validator generator; it behaves precisely like a compiler generator and it is the core of our architecture, as can be seen in Figure 7. It takes a valid topic map schema and constraint specification (an XML instance, written according to the XTche schema), verified by the XTche-SpecVP introduced in Section 3.2, and generates an XSL stylesheet (the TM-Validator) that will process an input topic map and will generate an ok/error messages (an ok message states that the topic map is valid according to the XTche specification).

Both XSL stylesheets (the generator and the validator) are interpreted by a standard XSL processor like Saxon[2], what in our opinion is one of the benefits of the proposal.

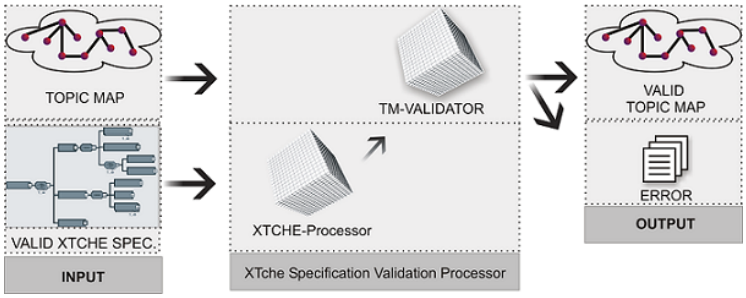


Fig. 7. XTche Architecture

During the development of this generator we found some problems that had a strong impact in the final algorithm. The most important was the ambiguity in constraint selection; until now, we have just said that an XTche specification

[2] http://saxon.sourceforge.net/

is composed of a set of constraints; we did not say that these constraints are disjoint in terms of context; in some cases there is a certain overlap between the contexts of different conditions; this overlap will cause an error when transposed to XSL; XSL processors can only match one context at a time. The solution we have adopted was to run each constraint in a different mode (in XSL each mode corresponds to a different traversal of the document tree).

4 Related Work

AsTMa! [1] is another Topic Maps constraint language, and Robert Barta also proposes a mechanism to validate a topic map document against a given set of rules. This language uses *AsTMa=* [2], the authoring language, and extends it with several new language constructs, and logic operators like NOT, AND and OR, simple logical quantifiers and regular expressions. *AsTMa!* exposes some features of a future TMCL.

In another related work, Eric Freese [4] says that it should be possible to use the DAML+OIL language to provide a constraint and validation mechanism for topic map information. The cited paper discusses how to describe validation and consistency of the information contained in Topic Maps using DAML+OIL and RDF, showing how to extend XTM and how to define PSIs and class hierarchies, as well as assign properties to topics.

Comparing XTche with the other known approaches, some advantages of XTche emerge: XTche has a XML Schema-based language, a well-known format. In addition, XTche allows the use of an XML Schema graphic editor, like XMLSpy. With the diagrammatic view, it is easy to check visually the correctness of the specification. Moreover, XTche gathers in one specification both the structure and the semantic descriptions, and it realizes a fully declarative approach requiring no procedural knowledge for users.

Talking about the constraints covered, XTche and *AsTMa!* have more mechanisms to check the Topic Maps validation than the Eric Freese's proposal.

5 Conclusion

In this paper we introduced a Topic Maps Validation System - XTche Constraint language and its processor. We started with our strong motivation to check a topic map for syntactic and semantic correctness - as a notation to describe an ontology that supports a sophisticated computer system (like the applications in the area of Semantic Web or archiving) its validation is crucial!

Then we assumed XTM and TMCL as starting points and we used our background in compilers and XML validation to come up with our proposal. XTche complies with all requirements stated for TMCL but it is an XML Schema oriented language. This idea brings two benefits: on one hand it allows for the syntactic specification of Topic Maps (not only the constraints), eliminating the need for two separated specifications; and on the other hand it enables the use of

an XML Schema editor (for instance, XMLSpy) to provide a graphical interface and the basic syntactic checker (the first stage of the XTche-SpecVP).

We succeeded in applying this approach to some case studies - E-Commerce Application and a personal video library management system - virtually representative of all possible cases. It means that: on one hand, we were able to describe the constraints required by each problem in a direct, clear and simple way; on the other hand, the Topic Maps semantic validator could process every document successfully, that is keeping silent when the constraints are satisfied, and detecting/reporting errors, whenever the contextual conditions are broken.

References

1. Robert Barta. AsTMa! Bond University, TR., 2003. http://astma.it.bond.edu.au/constraining.xsp.
2. Robert Barta. AsTMa= Language Definition. Bond University, TR., 2004. http://astma.it.bond.edu.au/astma=-spec-xtm.dbk.
3. Michel Biezunsky, Martin Bryan, and Steve Newcomb. ISO/IEC 13250 - Topic Maps. ISO/IEC JTC 1/SC34, December, 1999. http://www.y12.doe.gov/sgml/sc34/document/0129.pdf.
4. Eric Freese. Using DAML+OIL as a Constraint Language for Topic Maps. In *XML Conference and Exposition 2002*. IDEAlliance, 2002. http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-03-03/05-03-03.html.
5. Giovani Rubert Librelotto, José Carlos Ramalho, and Pedro Rangel Henriques. XTche - A Topic Maps Schema and Constraint Language. In *XML 2004 Conference and Exposition*, Washington D.C., U.S.A, 2004. IDEAlliance.
6. Mondeca. Questions and Answers. Mondeca, March, 2004. http://www.mondeca.com/english/faqs.htm.
7. Mary Nishikawa, Graham Moore, and Dmitry Bogachev. Topic Map Constraint Language (TMCL) Requirements and Use Cases. ISO/IEC JTC 1/SC34 N0405rev, 2004. http://www.jtc1sc34.org/repository/0548.htm.
8. J. Park and S. Hunting. *XML Topic Maps: Creating and Using Topic Maps for the Web*, volume ISBN 0-201-74960-2. Addison Wesley, 2003.
9. Steve Pepper. The TAO of Topic Maps - finding the way in the age of infoglut. Ontopia, 2000. http://www.ontopia.net/topicmaps/materials/tao.html.
10. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/.
11. Steve Pepper and Graham Moore. XML Topic Maps (XTM) 1.0 - Annex D: XTM 1.0 Document Type Declaration (Normative). TopicMaps.Org Specification, August, 2001. http://www.topicmaps.org/xtm/1.0/#dtd.
12. H. Holger Rath. White Paper: The Topic Maps Handbook.
13. Ann Wrightson. Topic Maps and Knowledge Representation. Ontopia, February, 2001. http://www.ontopia.net/topicmaps/materials/kr-tm.html.

Produção Dinâmica e Interactiva de Relatórios Baseados na Linguagem XML/RDL, com Foco na Geração de Queries MDX

Paulo Santos [1], Alberto Rodrigues da Silva [2]

[1] INESC-ID e Instituto Superior Técnico,
Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
paulo.m.santos@netcabo.pt
[2] INESC-ID e Instituto Superior Técnico,
Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
alberto.silva@acm.org

Resumo. As aplicações de *Business Intelligence* têm por objectivo satisfazer as necessidades de informação das organizações e auxiliar nos seus processos de tomada de decisão. As capacidades de *reporting* enquadram-se na generalidade de sistemas de informação, mas com maior relevância nesta área do *Business Intelligence*. O aparecimento da nova plataforma de *reporting* da Microsoft, "*Reporting Services*", veio disponibilizar uma plataforma extensível para construção e distribuição de relatórios especificados num formato aberto em XML. Neste artigo descreve-se um método de produção dinâmica e interactiva de relatórios baseados no modelo de objectos dos "*Reporting Services*", definidos numa linguagem XML, mais precisamente na linguagem RDL (*Report Definition Language*).

1 Introdução

Desde os anos 90 as aplicações de *Business Intelligence* sofreram uma evolução enérgica graças ao crescimento exponencial da informação, aliado aos avanços tecnológicos. As aplicações de *Business Intelligence* permitem às organizações obterem o conhecimento necessário à melhoria dos seus processos de negócio, responderem atempadamente aos seus clientes e adaptarem-se a alterações no mercado [1]. Este conjunto de aplicações abrange todas as áreas funcionais de uma organização, com especial ênfase nos níveis estratégico e de gestão. Estas aplicações servem no auxílio à tomada de decisões, pelo que, se podem classificar em sistemas de suporte à decisão (*Decision Support Systems*), embora, também possam existir a nível estratégico [2][3]. Existem diversos tipos de aplicações *Business Intelligence*. Há quem defenda que são cinco os seus tipos de aplicação [4]. A área de *reporting* é uma dessas aplicações do *Business Intelligence*.

No âmbito das ferramentas e plataformas de *reporting* a Microsoft lançou recentemente a sua plataforma, "*SQL Server Reporting Services*". Dentro desta área, existem diversos produtos no mercado, tais como, o MicroStrategy, o Business Objects,

ou o Cognos. Porém, esta é uma solução inovadora que permite a criação, gestão e distribuição de relatórios sob diferentes formatos (e.g., DOC, PDF, HTML, Excel) [5]. Ao passo que a maioria dos produtos existentes no mercado definem os relatórios segundo um formato proprietário, os Reporting Services definem-nos segundo um formato aberto. Uma característica fundamental desta plataforma consiste no facto dos seus relatórios serem definidos com base na linguagem RDL (*Report Definition Language*), que é um esquema XML. Este facto permite a disponibilidade de uma plataforma extensível para a criação e distribuição de relatórios especificados num formato aberto em XML.

Neste artigo descreve-se um método de produção dinâmica e interactiva de relatórios que tiram partido das capacidades da plataforma "*Reporting Services*", e em particular da linguagem RDL.

Na Secção 2 introduz-se a arquitectura da plataforma "*Reporting Services*". A Secção 3 introduz sucintamente a linguagem RDL (*Report Definition Language*) descrevendo o respectivo modelo de objectos. Nas Secções 4 e 5 é apresentado, respectivamente, a arquitectura do sistema proposto no âmbito da nossa investigação para a produção dinâmica e interactiva de relatórios definidos na linguagem RDL, e descrito o método para produção dinâmica e interactiva dos relatórios. Finalmente, na Secção 6 são apresentadas as conclusões.

2 Arquitectura da Plataforma "*Reporting Services*"

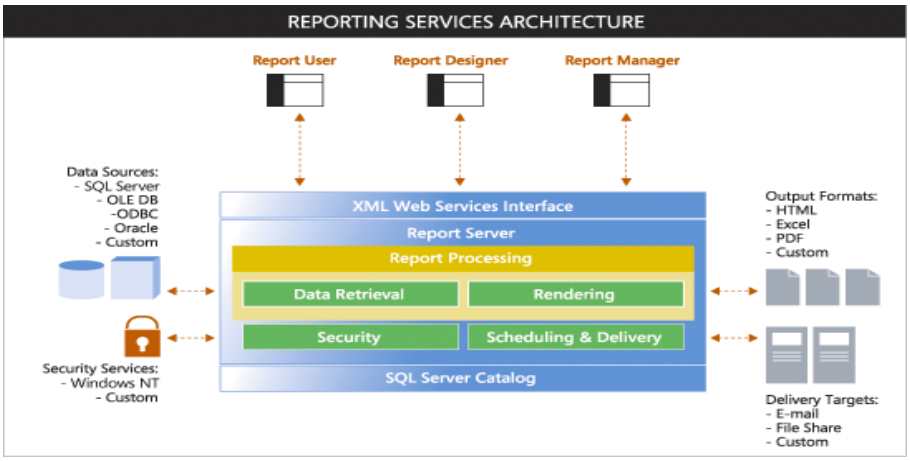


Figura 1 - Arquitectura dos Reporting Services (adaptado de [5])

Os *Reporting Services* possuem uma arquitectura modular, como sugerido na Figura 1. A plataforma tem por base o *Report Server* cuja função consiste em suportar toda e qualquer interacção dos utilizadores. Para além deste componente, existe ainda um conjunto de características que conferem à plataforma um carácter extensível. Esta arquitectura vai ser sumariamente apresentada nas secções seguintes.

2.1 Extensibilidade da Plataforma

Para além da plataforma suportar um conjunto diversificado de fontes de dados, tais como, SQL Server, Oracle, OLE DB ou ODBC, a arquitectura dos "*Reporting Services*" está desenhada de forma a suportar outros tipos de fontes de dados.

O mesmo se passa relativamente aos formatos de *output*: para além dos formatos disponibilizados por omissão (i.e., HTML, PDF, Microsoft Excel), os programadores podem criar as suas próprias extensões de *rendering* de modo a tirar vantagem das capacidades de outros tipos de dispositivos.

Quanto à autenticação suportada pela arquitectura baseia-se na autenticação *Windows*, embora se possam definir outros tipos de segurança.

No que toca à distribuição de relatórios, esta arquitectura encontra-se preparada para efectuar uma distribuição via *e-mail*, embora outros tipos de distribuição possam ser programados.

2.2 Componentes Aplicacionais

O ***Report Server***, como componente principal da arquitectura, funciona como um serviço normal em ambiente *Windows*.

A **base de dados** do *Report Server* encontra-se no *SQL Server* e armazena toda a informação referente aos *Reporting Services*. Esta informação passa pelo registo das definições dos relatórios, dos metadados associados aos relatórios, relatórios em *cache*, *snapshots*, configurações de segurança, dados encriptados, configurações de *scheduling* e outras informações referentes a extensões aos "*Reporting Services*".

Esta base de dados apenas é acessível através do *Report Server*, ou seja, mesmo utilizando o *Report Manager*, o *Report Designer* ou outros "utilitários *command-line*" (ver abaixo), estamos a utilizar interfaces programáticas de acesso à base de dados que comunicam com o *Report Server* de uma forma transparente.

O ***Report Manager*** é uma aplicação *Web* que acompanha a plataforma com o intuito de gerir e aceder a relatórios através de uma interface gráfica. A partir do *Report Manager* é possível realizar as seguintes operações: (1) Visualizar, procurar e subscrever relatórios; (2) Criar e gerir pastas, *linked reports*, ligações a fontes de dados e subscrições de relatórios; (3) Criar e modificar propriedades e parâmetros dos relatórios; e (4) Gerir definições de papéis que controlam o acesso a relatórios e pastas por diferentes utilizadores.

O ***Report Designer*** é uma ferramenta que permite a criação e a publicação de relatórios de uma forma simples e cómoda. Esta ferramenta acompanha o *Visual Studio .Net 2003* e utiliza uma interface gráfica bastante prática.

3 A linguagem RDL (*Report Definition Language*)

A linguagem RDL (*Report Definition Language*) é a linguagem utilizada pelos "*Reporting Services*" para definição de relatórios. Todos os relatórios, segundo a plataforma de *reporting* da Microsoft, são descritos por um ficheiro em XML. Estes

ficheiros estão de acordo com um esquema XML (*XML schema*) e podem ser visualizados pelo servidor de *reporting* a partir do momento que estejam publicados no mesmo. A definição de um relatório à custa da linguagem XML, permite a criação de objectos específicos de acordo com o interesse do utilizador (ou programador) sem estar restringido a um conjunto limitado de objectos. Este aspecto prova a extensibilidade da plataforma, tanto a nível dos objectos do relatório (*design*) como dos tipos de fontes de dados que se possam aceder.

De acordo com o esquema XML para definição de relatórios actualmente publicado pela Microsoft [6] um relatório contém informações de acesso a dados e informações de *layout* do relatório.

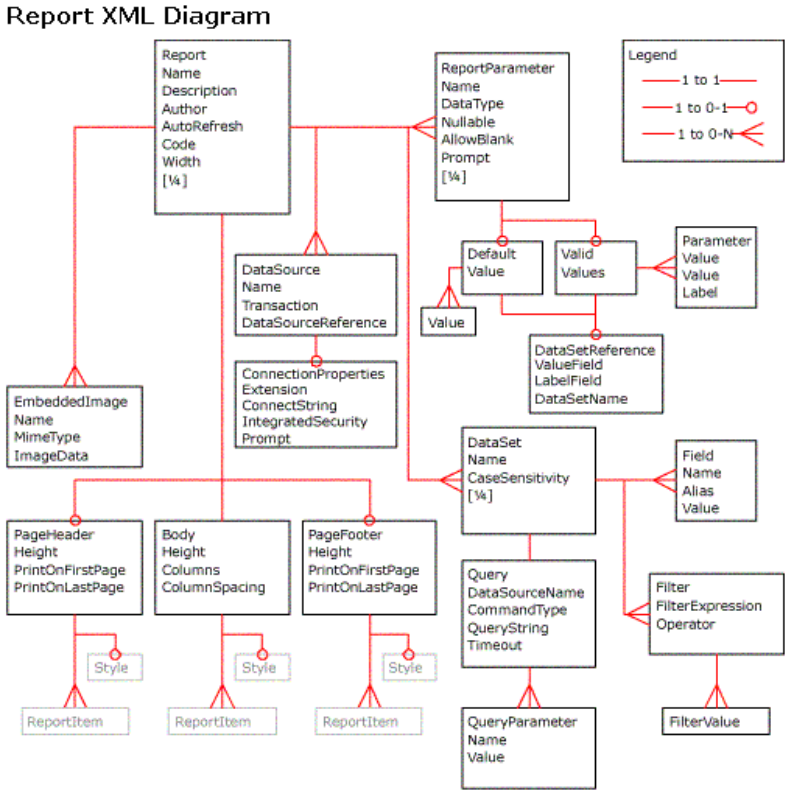


Figura 2 – Visão geral do modelo de objectos e de relatórios segundo a linguagem RDL

A Figura 2 apresenta o diagrama que descreve um relatório XML. Um relatório contém três elementos: *datasources*, *datasets* e *body*.

O elemento ***datasources*** descreve o tipo de fontes de dados e respectivas propriedades necessárias para acesso aos dados (e.g., nome da base de dados, nome de utilizador, palavra-passe). Embora só possa existir um elemento *datasources*, este pode conter vários elementos *datasource* que descrevem a informação referida anteriormente. Os "*Reporting Services*", ao contrário de outras plataformas de *reporting*, permitem a definição de diversas fontes de dados para o mesmo relatório.

O elemento ***datasets*** contém uma lista de elementos *dataset*. Cada elemento *dataset* contém uma *query* de acesso a dados e um conjunto de campos disponíveis para o relatório. Cada elemento *dataset* contém ainda referência a um *datasource* definido anteriormente.

O elemento ***body*** contém a descrição do *layout* do relatório. Este elemento contém um sub-elemento (*ReportItems*) que contém por sua vez, os objectos a constar no relatório. Segundo o esquema XML [7], um relatório pode conter tabelas, matrizes, caixas de texto, imagens, linhas, rectângulos e ainda sub-relatórios. A Figura 3 apresenta o excerto de um ficheiro XML de definição de um relatório visualizado através do *Internet Explorer*.

```
<?xml version="1.0" encoding="utf-8" ?>
- <Report xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://schemas.microsoft.com/sqlserver/reporting/2003/10/reportdefinition">
  + <DataSources>
  - <DataSets>
    - <DataSet Name="DataSet1">
      + <Fields>
      - <Query>
          <DataSourceName>dmport</DataSourceName>
          <CommandText>SELECT {[Measures].[Nnavios]} on columns, NON EMPTY Descendants([Tempo].[All Tempo],
            [Tempo].[Mes], LEAVES) on rows, NON EMPTY {[ClasseDWT].[Classedwt Nome].Members} on pages FROM
            [EuroF1]</CommandText>
        </Query>
      </DataSet>
    </DataSets>
  - <Body>
    - <ReportItems>
      + <Textbox Name="Textbox_Title">
      + <Matrix Name="Matrix1">
      + <Chart Name="Chart1">
      </ReportItems>
      <Height>26.25cm</Height>
      <ColumnSpacing>1cm</ColumnSpacing>
      <Style />
    </Body>
    <Width>21cm</Width>
  </Report>
```

Figura 3 - Exemplo de um ficheiro XML (RDL) para definição de um relatório

No exemplo da Figura 3 é possível verificar a definição dos elementos *datasources* e *datasets*. O elemento *datasets* contém apenas um *dataset*. O elemento *ReportItems* contém três objectos: uma caixa de texto, uma matriz e um gráfico.

4 Arquitectura do sistema proposto

Esta investigação tem como requisito básico a produção de relatórios *on the fly*, isto é, de forma dinâmica. Em vez dos relatórios serem desenhados previamente e definidos de modo estático, pretendemos que seja o próprio utilizador a definir o seu relatório, interactivamente (em *runtime*), consoante as suas necessidades de informação.

A produção de relatórios nesta solução foca-se essencialmente na geração de *queries* MDX [8][9], ou seja, *queries* de acesso a dados multidimensionais. A utilização

de dados multidimensionais neste protótipo traduz-se num caso particular, sendo por isso, a solução extensível a outros tipos de dados. Embora a solução apenas se foque na produção dinâmica de *queries* MDX, também se poderia ter dado maior importância na geração dinâmica do *design* do relatório – o sistema desenvolvido apenas permite a construção dinâmica do título do relatório e escolher entre a visualização do relatório sob a forma de tabela ou a forma gráfica. Contudo, este aspecto, não menos importante na produção de relatórios, cai fora do âmbito da nossa investigação.

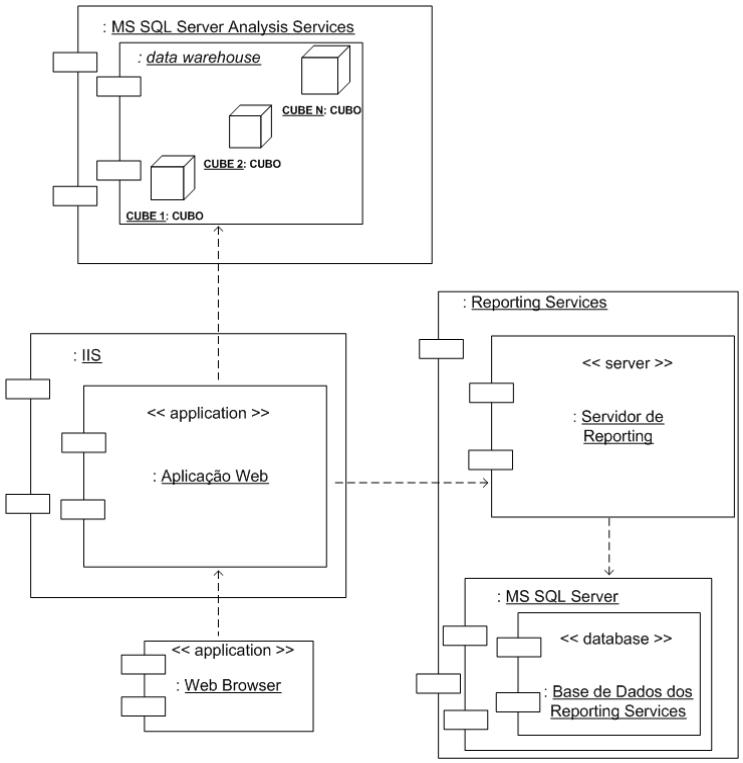


Figura 4 - Diagrama de componentes da arquitectura do sistema proposto

Este sistema é composto por três elementos fundamentais: (1) base de dados multidimensional; (2) aplicação *Web*; e (3) plataforma de *reporting*.

A Figura 4 ilustra a arquitectura do sistema proposto.

A **base de dados multidimensional** (*data warehouse*) contém todos os cubos com a informação necessária à produção de relatórios e está armazenada no servidor *Analysis Services* [10].

A **aplicação *Web*** reside num servidor aplicacional IIS (*Internet Information Services*).

A **plataforma de *reporting*** utilizada foi os "*Reporting Services*" (descritos na Secção 2). Esta plataforma contempla um servidor de relatórios (*Report Server*) e uma base de dados.

A aplicação pode ser acedida por um utilizador através de um *browser*, por exemplo, o *Internet Explorer*.

Embora actualmente todos os elementos do sistema residam na mesma máquina, tanto a aplicação *Web*, a base de dados multidimensional, como a plataforma de *reporting* podem residir em servidores distintos. Se tal situação acontecer, basta alterar o ficheiro de configurações da aplicação (*Web.config*) de modo a contemplar os endereços dos respectivos servidores.

5 Método para produção dinâmica e interactiva de relatórios

Embora os "*Reporting Services*" disponibilizem um componente específico para realizar o *design* de relatórios (o *Report Designer* como foi já referido na sub-secção 2.2), a produção dinâmica de relatórios obriga-nos a recorrer a API's específicas dos *Reporting Services* e à utilização de uma linguagem de programação (utilizou-se nesta investigação o *C#* e o ambiente *Visual Studio .Net 2003*).

A produção dinâmica e interactiva de relatórios subdivide-se em dois problemas: (1) a selecção da informação a constar no relatório (carácter interactivo); e (2) a construção do relatório propriamente dito (carácter dinâmico).

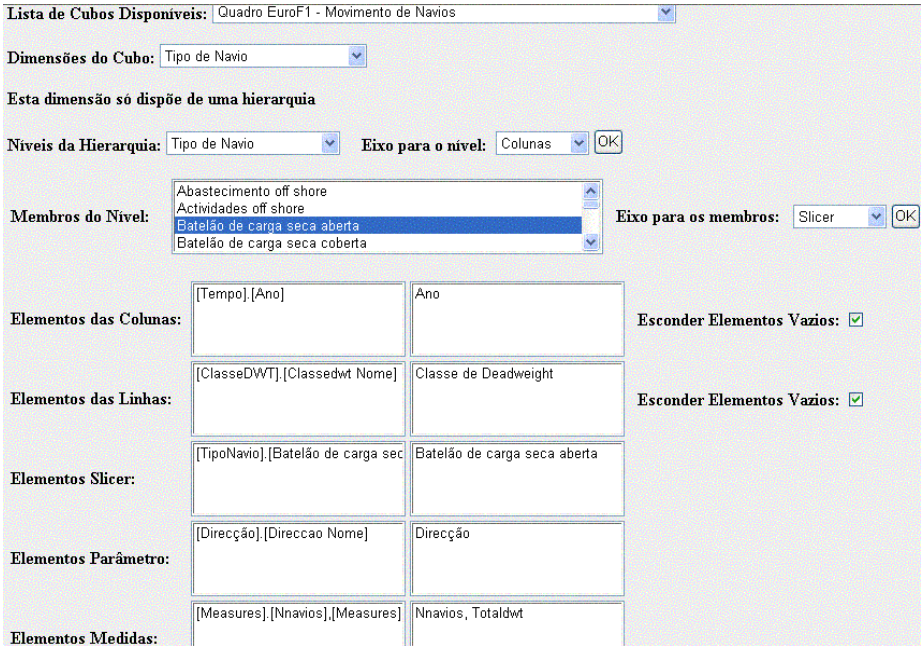


Figura 5 - Protótipo desenvolvido

5.1 Interactividade

Relativamente ao carácter interactivo, a solução adoptada passou pela construção de um protótipo que disponibilizasse a informação existente na base de dados, de modo a que o utilizador pudesse seleccionar os dados pretendidos para o relatório. Embora neste sistema tenham sido utilizados dados multidimensionais, extraídos a partir dos *Analysis Services*, a solução estende-se com maior facilidade também a bases de dados relacionais, e por isso, é generalizável. A Figura 5 apresenta um *snapshot* do protótipo desenvolvido, em que um utilizador constrói interactivamente o relatório.

5.2 Dinâmica

Em relação ao carácter dinâmico dos relatórios, a solução consiste no mapeamento das entidades seleccionadas pelo utilizador no modelo de objectos dos "*Reporting Services*" (descritos na Secção 3).

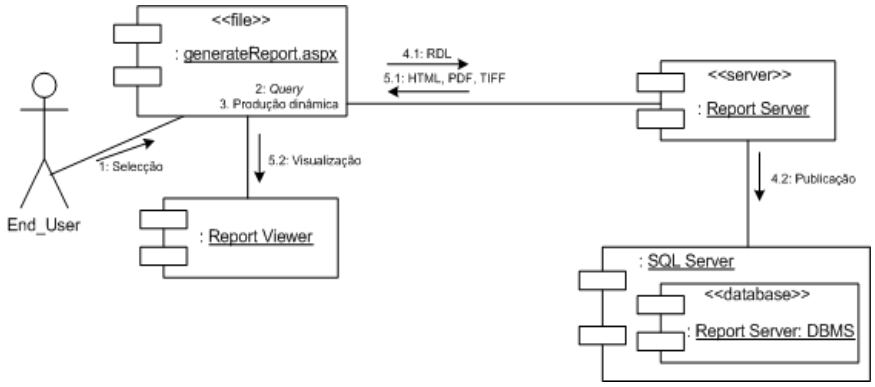


Figura 6 - Diagrama de componentes relativo à produção de um relatório dinâmico

A produção dinâmica e interactiva de relatórios desenrola-se segundo a seguinte sequência de passos (conforme sugerido na Figura 6):

Selecção do cubo e respectivas dimensões a constar no relatório por parte do utizador utilizando a interface gráfica disponibilizada (Passo 1)

Neste passo o utilizador tem a possibilidade de escolher o cubo que deve constar no relatório e quais as respectivas dimensões do cubo. As medidas do cubo (*Measures*) são tratadas de uma forma particular. Na construção de um relatório utilizando os "*Reporting Services*" é obrigatório colocar as medidas de um cubo na cláusula *Columns* da *query* MDX. Por isso, houve o cuidado de separar as medidas das restantes dimensões como pode ser observado na Figura 5, por forma a facilitar a construção da *query* MDX.

Criação da *query* MDX mediante as dimensões seleccionadas (Passo 2)

Este passo consiste na geração da *query* MDX de acordo com as dimensões indicadas pelo utilizador. Como exemplo do MDX gerado apresentamos o seguinte excerto (relacionado com os dados do caso de estudo), fazendo referência à cláusula *Columns* que apenas contém Medidas (*Measures*).

```
SELECT  NON EMPTY {[Measures].[Pesobruto]} on columns,
NON EMPTY Descendants([Tempo].[All Tempo], [Tempo].[Mes], LEAVES) on
rows,
NON EMPTY CROSSJOIN ({[Porto].[Porto
Nome].Members},{[TipoCarga1].[TipoCarga1 Nome].Members}) on pages
FROM [EuroA1]
```

Produção do relatório (Passo 3)

A produção de um relatório segundo os "*Reporting Services*" consiste na criação de um ficheiro XML com a extensão RDL (*Report Definition Language*). Uma vez que se pretende a produção dinâmica de relatórios o ficheiro XML teve que ser criado de raíz, programaticamente. A produção do relatório implica a criação das fontes de dados, das *queries* MDX e dos objectos de visualização (no caso, tabelas e/ou gráficos), de acordo com as opções do utilizador.

Na produção do relatório segundo a linguagem RDL, recorreu-se à ferramenta *xsd.exe* (*XML Schema Definition Tool*) que acompanha a *framework .Net*. Esta ferramenta permite, a partir de um esquema XSD (*XML Schema Definition Language*) a geração de classes *C#* conformes ao esquema. Foi utilizado o esquema de dados *XSD* que acompanha os "*Reporting Services*" e que descreve a especificação RDL [6].

Uma vez tendo as classes geradas, é possível a criação dos objectos que constarão no relatório, de acordo com as opções do utilizador. A geração das classe permite programar num nível mais elevado, ao invés de programar directamente sobre XML. Após a criação de todos os objectos é efectuada a sua serialização para disco, ou seja, é criado o ficheiro de extensão RDL (a Figura 7 apresenta um excerto de código XML produzido).

```
- <Matrix Name="Matrix1">
    <Top>2cm</Top>
    <Left>0.5cm</Left>
    <DataSetName>DataSet1</DataSetName>
  + <Corner>
  - <ColumnGroupings>
    - <ColumnGrouping>
        <Height>0.63cm</Height>
      - <DynamicColumns>
        - <Grouping Name="Matrix_Grouping_Tempo_Ano">
          - <GroupExpressions>
              <GroupExpression>=Fields!Tempo_Ano.Value</GroupExpression>
            </GroupExpressions>
          </Grouping>
        + <ReportItems>
          <Visibility />
        </DynamicColumns>
      </ColumnGrouping>
    + <ColumnGrouping>
    + <ColumnGrouping>
    + <ColumnGrouping>
    </ColumnGroupings>
  + <RowGroupings>
  + <MatrixRows>
  + <MatrixColumns>
  </Matrix>
```

Figura 7 - Excerto do código XML produzido para definição de uma matriz

Publicação do relatório (Passo 4)

Uma vez tendo sido produzido o relatório no passo 3 é necessário publicá-lo para o servidor para que este possa ser visualizado. A publicação do relatório para o *Report Server* passa pelo registo de toda a informação associada ao relatório na base de dados dos *Reporting Services*). Na publicação do relatório foi invocado o método *CreateReport* do *Web Service* disponível pelos *Reporting Services*:

```
ReportingService rs = new ReportingService();
rs.Credentials = System.Net.CredentialCache.DefaultCredentials;

Byte[] definition = null;
Warning[] warnings = null;

…

FileStream stream = File.OpenRead(ut.getXMLValue(“reportOutput”));
definition = new Byte[stream.Length];
stream.Read(definition, 0, (int) stream.Length);
stream.Close();

…

warnings = rs.CreateReport(name, “/” + ut.getXMLValue(“reportsPath”),
true, definition, null);
```

A função auxiliar *getXMLValue* permite obter o valor de um parâmetro definido no ficheiro de configurações Web.config.

Visualização do relatório publicado através do *Report Viewer* (Passo 5)

Finalmente, o passo 5 envolve a visualização do relatório num ambiente *Web*. Para tal, foi necessário utilizar o *viewer* desenvolvido pela Microsoft para o efeito, o ***Report Viewer***.

```
protected Microsoft.Samples.ReportingServices.ReportViewer ReportViewer1;

…

ReportViewer1.ReportPath = "/" + ut.getXMLValue("reportsPath") + "/" +
                           ut.getXMLValue("reportName");
ReportViewer1.ServerUrl = "http://" + ut.getXMLValue("datasource") +
                           ut.getXMLValue("reportServer");
```

A visualização do relatório no âmbito da nossa investigação é possível segundo duas formas: a forma em tabela (matriz) e a forma gráfica. A Figura 8 apresenta a visualização de um relatório através do Report Viewer sob a forma em tabela.

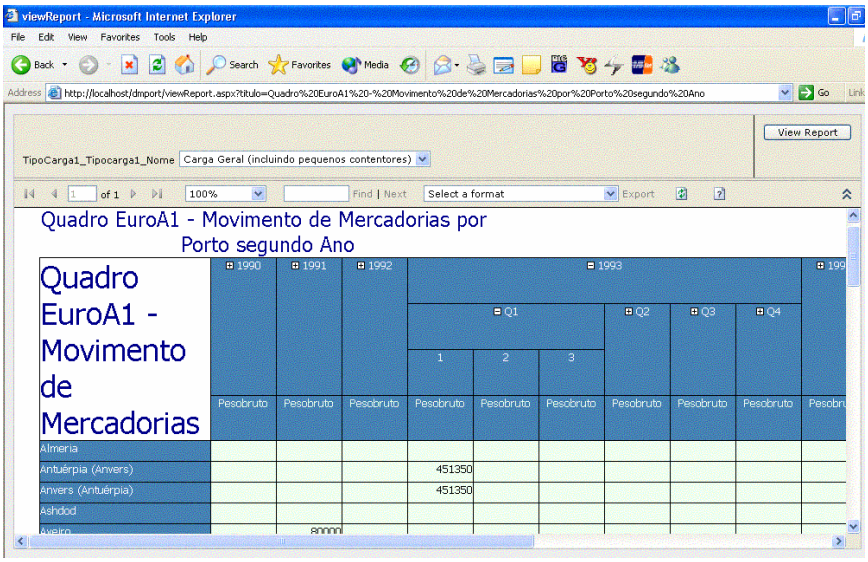


Figura 8 - Visualização do relatório através do *Report Viewer*

6 Conclusões

A plataforma de *reporting* da Microsoft, "*Reporting Services*", permite disfrutar de uma plataforma extensível para construção e distribuição de relatórios. Ao contrário da maioria dos produtos existentes no mercado na área de *reporting*, cujos relatórios são definidos segundo um formato proprietário, os Reporting Services definem os relatórios segundo um formato aberto. O facto dos relatórios serem definidos à custa de uma linguagem aberta, o XML, permite aos utilizadores definirem novos elementos de acordo com as necessidades do seu negócio. A esta vantagem acresce o facto de podermos implementar dinamicamente relatórios (em *runtime*), sem estarmos restringidos a um conjunto dos mesmos definidos previamente (em *design-time*).

Referências

1. Acuma, World Class Information Management, http://www.acuma.co.uk/802568AC005A7933/pages/BusinessIntelligenceOverviewBusinessIntelligence.html
2. Laudon, Kenneth C.; Laudon, Jane P., "Management Information Systems – Managing the Digital Firm", 7th Edition, Prentice Hall, 2002
3. Jessup, Leonard; Valacich, Joseph, "Information Systems Today", "Chapter 6 – Organizational Information Systems", Prentice Hall, 2003
4. "The 5 Styles of Business Intelligence: Industrial-Strength Business Intelligence", A White Paper prepared by MicroStrategy, Inc., Copyright 2002, http://www.microstrategy.com/Solutions/5StylesBook.asp
5. Microsoft SQL Server: Reporting Services Product Overview, http://www.microsoft.com/sql/reporting/productinfo/overview.asp
6. Microsoft SQL Server 2005, Report Definition Language Schema, May 2004, http://schemas.microsoft.com/sqlserver/reporting/2003/10/reportdefinition/reportdefinition.xsd
7. Report Definition Language Specification, Third Draft, December 2003, Microsoft Corporation, http://download.microsoft.com/download/4/7/d/47d7d117-9f91-49ad-98d5-46aa6f3251a8/RDLDec03.pdf
8. Spofford George, "MDX Solutions with Microsoft SQL Server Analysis Services", Wiley, 2001
9. Whitehorn, Mark; Pasumansky, Mosha; Zare, Robert, "Fast Track to MDX", Springer
10. Jacobson, Reed, "Microsoft SQL Server 2000 Analysis Services Step by Step", Wiley, 2001, Microsoft Press, 2000

(**Nota:** As hiperligações foram acedidas no período compreendido entre Outubro e Novembro de 2004)

Document Composer: uma aplicação XML para extracção de informação de repositórios XML

José Carlos L. Ramalho

Departamento de Informática, Universidade do Minho
{jcr}@di.uminho.pt

Resumo *Document Composer* é uma ferramenta desenvolvida como um exercício de reflexão em XSLT e que nasce da tentativa de criar um nível de abstracção para alunos dum curso de XML. Com este nível de abstracção pretendia-se que o aluno conseguisse realizar tarefas para as quais já havia adquirido os conceitos mas não a capacidade técnica.
A ferramenta permite que um utilizador interrogue um repositório de documentos XML (os documentos poderão pertencer a várias classes, i.e., estar de acordo com DTDs ou Schemas diferentes ou, simplesmente serem bem formados). O resultado de cada operação de interrogação é um novo documento XML. A ferramenta foi concebida de maneira a que os resultados individuais de cada interrogação possam ser combinados da forma que o utilizador entender para produzir o documento resultado (a composição é feita declarativamente na definição da estrutura do novo documento).
O processo de interrogação/extracção de informação é especificado numa linguagem XML desenvolvida para o efeito. Com esta linguagem o utilizador especifica o esqueleto de um novo documento e indica, através de interrogações (*queries*) escritas em XPath, como é que vai povoar cada componente do novo documento com a informação extraída dos vários documentos no repositório.
Neste artigo, serão descritos os passos seguidos para a criação desta ferramenta e alguns casos de estudo onde ela já foi utilizada.

1 Introdução

A ideia original para a construção desta ferramenta surgiu quando, numa aula de um curso sobre tecnologia XML, foi necessário colocar os alunos a trabalhar com XPath antes destes saberem construir *stylesheets* XSL ou manipular sistemas baseados em XQuery. O requisito principal era dar aos alunos a capacidade de seleccionar partes de documentos XML tendo apenas conhecimento do XPath.

Existem algumas ferramentas que permitem visualizar o resultado de uma expressão de selecção especificada em XPath. Por exemplo, o XMLSpy [Alt] incorpora um utilitário que permite o cálculo de expressões XPath fornecendo ao utilizador uma lista dos nodos da árvore documental abstracta que podem ser visualizados e manipulados graficamente por este. No nosso caso, pretendia-se um pouco mais. Desejava-se que o resultado da selecção fosse retornado como um novo documento XML para depois se poder processá-lo e transformá-lo noutro documento ou simplesmente poder navegar sobre ele.

Para resolver este problema resolveu-se utilizar o XSLT como motor de processamento. O seu modelo de programação segue o paradigma de programação por regras, ou seja, padrão-acção. E em XSLT, o XPath é usado para os padrões de selecção, i.e., as partes do documento às quais irão ser aplicadas acções são seleccionadas por expressões XPath. A ideia de usar o XSLT como motor de cálculo baseado em regras surgiu em 1999 com o XCSL [Ram01], e o Schematron [Jel04]. Estas aplicações adicionam um nível semântico aos DTDs e Schemas que pode ser especificado em regras escritas em XPath. Nos anos que se seguiram, surgiram mais algumas aplicações deste tipo em áreas diferentes mas onde era necessário criar um nível de abstracção, por exemplo: a arquitectura MTRANS [PBG01] desenvolvida para a conversão de modelos MOF e o GRAPHOTRON [Nic00] para a definição de vistas sobre grafos.

Logo à partida, surgiu um problema. Os alunos desconheciam o XSL e portanto não poderiam especificar as *stylesheets*. Consequentemente, estas teriam de ser geradas automaticamente.

O processo de geração automática de *stylesheets* é um processo de geração a dois níveis, envolvendo aquilo que normalmente se designa por *stylesheet genérica de segunda geração* (nalguns artigos são também referidas como *stylesheets* com reflexão [KK03,Kos03]). Uma transformação XSL de segunda geração é uma transformação (um pouco abstracta) que vai transformar um documento XML, que contem a especificação de um determinado processo, e que vai gerar como resultado uma nova transformação XSL (mais específica) que implementa o processo especificado. Esta nova *stylesheet* quando aplicado a um determinado documento XML irá então produzir o resultado final esperado. Há vários exemplos de aplicações que utilizam esta metodologia para criar níveis de abstracção encapsulando, desta forma, alguma da complexidade dos processos que implementam.

Ao longo deste artigo, começa-se por descrever a evolução da ferramenta até ao seu estado actual. Continua-se com a discussão da estrutura actual da ferramenta e de cada um dos seus componentes. A seguir apresenta-se um caso de estudo que exemplifica a utilização da ferramenta. Por fim, apresentam-se algumas conclusões e apontam-se algumas linhas para trabalho futuro.

2 Do *XPath Wrapper* ao *Document Composer*

Para, colocar alunos, que não tinham conhecimentos de XSL, a trabalhar com XPath como se esta fosse uma linguagem de query houve que criar um nível de abstracção que tornasse a complexidade de criação duma *stylesheet XSL* invisível para o utilizador. O nível de abstracção consistiu numa especificação XML composta por uma sequência de expressões XPath devidamente colocadas no seio de alguns elementos especificamente criados para o efeito, ou seja, as expressões XPath são colocadas como conteúdo de elementos XML e cada uma corresponde a uma query sobre o documento alvo da pesquisa.

2.1 Especificação de queries

Para a linguagem XML de especificação foi desenvolvido um pequeno DTD que se apresenta a seguir:

```
<!ELEMENT query-set (query+)>
<!ELEMENT query (#PCDATA)>
```

O conteúdo do elemento *query* deverá ser uma das expressões XPath a serem calculadas.

Considere o seguinte exemplo.

Exemplo 1: Queries sobre o Arquivo Sonoro de Ernesto Veiga de Oliveira

---

Suponha que possuía uma versão do Arquivo Sonoro de Ernesto Veiga de Oliveira em XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <arq>
  <doc>
    <prov>Alentejo</prov>
    <local>Santa Vitória, Beja</local>
    <tit>Disse a laranja ao limão</tit>
    <musico>Jorge Montes Caranova</musico>
    <file t="MP3">d1/evo001.mp3</file>
    <duracao>1:02</duracao>
  </doc>
- <doc>
    ...
  </doc>
  ...
</arq>
```

e que sobre ele queria calcular as seguintes queries:

1. Seleccionar o título das músicas que contêm a palavra "Jesus"no título.
2. Seleccionar o nome de todos os músicos referidos.
3. Seleccionar o título de todas as músicas de Castelo Branco.

Estas queries seriam então especificadas na linguagem descrita acima:

```
<?xml version="1.0"?> <query-set>
  <query>/arq/doc/tit[contains(.,'Jesus')]</query>
  <query>//musico</query>
  <query>//doc/tit[contains(../local),'Castelo Branco')]</query>
</query-set>
```

Como se pode ver do exemplo, o utilizador desta ferramenta apenas precisa de conhecer o XPath. Na secção seguinte, apresenta-se o método seguido para o cálculo das queries.

2.2 Cálculo das queries

A maneira mais fácil de processar uma expressão XPath é imbuti-la numa stylesheet XSL e processar o documento XML sobre o qual se querem calcular as queries com esta stylesheet.

Para isso, desenvolveu-se uma stylesheet XSL especial (vamos designá-la por XPP - "XPath Processor") que implementa este nível de abstracção. Esta stylesheet quando aplicada a um documento XML com a especificação de queries gera uma stylesheet XSL mais específica que quando aplicada ao documento produz o resultado do cálculo das queries.

A seguir mostra-se o resultado da aplicação de XPP ao documento com as queries apresentado no exemplo 1.

Exemplo 2: Stylesheet XSL que implementa o cálculo das queries

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<axsl:stylesheet
  xmlns:axsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <axsl:output method="xml" omit-xml-declaration="no"
    encoding="iso-8859-1" standalone="yes" indent="yes"/>

  <axsl:template match="/">
    <doc-result>
      <result-set id="q0" qexp="/arq/doc/tit[contains(.,'Jesus')]">
        <axsl:apply-templates mode="Q0"/>
      </result-set>
      <result-set id="q1" qexp="//musico">
        <axsl:apply-templates mode="Q1"/>
      </result-set>
      <result-set id="q2" qexp="//doc/tit[../prov='Castelo Branco']">
        <axsl:apply-templates mode="Q2"/>
      </result-set>
    </doc-result>
  </axsl:template>

   <axsl:template mode="Q0" match="/arq/doc/tit[contains(.,'Jesus')]">
     <result>
       <axsl:copy-of select="."/>
     </result>
   </axsl:template>

   <axsl:template match="text()" priority="-1" mode="Q0"/>

   <axsl:template match="text()" priority="-1"/>
</axsl:stylesheet>
```

Esta stylesheet é um pouco diferente das stylesheets que especificam no dia-a-dia:

linha 1-33 – o prefixo utilizado não é o `xsl` e sim `axsl`; isto deve-se ao facto da stylesheet ter sido gerada e por isso estes elementos tiveram de coexistir na stylesheet XPP com elementos do XSL que estariam activos; a maneira de se conseguir distinguir na mesma stylesheet elementos XSL activos de elementos XSL que se estão a gerar como resultado é associando estes ao mesmo `Namespace` mas com um prefixo diferente.

linha 10-22 – esta é a template principal que coordena todo o processo de cálculo; como se pode ver o documento resultado segue também uma estrutura que se pode especificar no seguinte pseudo DTD:

```
<!ELEMENT doc-result (result-set+)>
<!ELEMENT result-set (result+)>
<!ATTLIST result-set
              id ID #REQUIRED
              qexp CDATA #REQUIRED>
<!ELEMENT result (subárvore do documento original)>
```

Referimo-nos a este DTD como pseudo DTD pois o conteúdo de `result` é uma subárvore de elementos que depende da instância que está a ser processada e da expressão de query e por isso sempre variável, o que inviabiliza a sua formalização.

linha 13,16,19 – estas linhas três linhas contêm a invocação de três travessias à instância documental (atributo `mode`), uma por cada query especificada; não é possível calcular as queries em simultâneo pois nada nos garante que estas sejam disjuntas e todos os processadores de XSL exigem que as templates sejam disjuntas na sua selecção de nodos.

linha 24-28 – esta `template` corresponde à implementação da primeira query; vai copiar cada uma das subárvores, cujo nodo raiz obedeça à expressão XPath especificada na query, para dentro dum elemento `result` no documento resultante.

linha 30 – como as travessias iniciadas na `template` principal são globais, vão visitar todos os nodos, é necessário filtrar aqueles que não interessam; é o que faz esta `template` para a travessia `Q0`.

linha 32 – esta `template` à semelhança da anterior filtra nodos residuais na travessia principal, que nesta aplicação apenas é utilizada para desencadear as outras travessias.

No exemplo seguinte, apresenta-se o resultado da aplicação desta stylesheet mais específica ao arquivo sonoro de Ernesto Veiga de Oliveira.

Exemplo 3: Resultados finais

Depois da aplicação da stylesheet específica o documento XML resultante tem a seguinte forma:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-result>
   <result-set id="q0" qexp="/arq/doc/tit[contains(.,'Jesus')]">
      <result>
         <tit>Versos ao Menino Jesus</tit>
      </result>
      ...
   </result-set>
   <result-set id="q1" qexp="//musico">
      <result>
         <musico>Jorge Montes Caranova</musico>
      </result>
      <result>
         <musico>Catarina Chitas</musico>
      </result>
      ...
   </result-set>
   <result-set id="q2"
        qexp="//doc/tit[contains(../local,'Castelo Branco')]">
      <result>
         <tit>Dança dos Homens;
            dedicada a Nª Senhora dos Altos Céus</tit>
      </result>
      ...
   </result-set>
</doc-result>
```

Como se pode ver, cada `result-set` tem a ele associados um identificador e a expressão XPath que o originou. Por acaso, neste exemplo, todos os nodos seleccionados eram nodos folha na árvore documental da instância, se fossem nodos intermédios cada `result` conteria a respectiva subárvore completa.

2.3 *Pretty-print* dos resultados

Uma vez que os resultados são gerados estruturadamente num documento XML, ficam à distância duma stylesheet XSL de qualquer outra representação. Para já, desenvolveu-se uma stylesheet XSL que gera uma página HTML que permite uma fácil navegação nos resultados das queries.

2.4 Pipeline de execução

Nos exemplos apresentados, da especificação das queries até aos resultados finais passamos por três processos de transformação encadeados em série. Actualmente esta cadeia de processos está implementada num ficheiro `bat` mas está prevista uma solução mais genérica baseada em XPipe [DuC02,PS03]. Com efeito, já se realizou um projecto

[RTFR04] cujo objectivo era encapsular estas pipelines de execução por detrás de um Web Service.

A figura 1 ilustra as duas primeiras etapas desta cadeia de transformações.

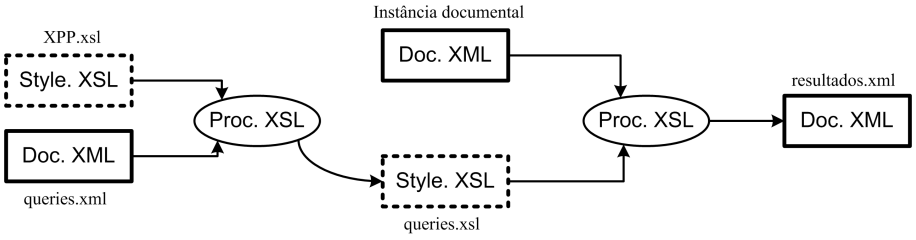


Figura 1. Pipeline de execução

Na próxima secção apresenta-se o `Document Composer` que resultou da generalização do `XPath Wrapper` permitindo ao utilizador combinar resultados de queries colocadas a vários documentos compondo deste forma novos documentos.

3 *Document Composer*

Depois de alguma experiência com o `XPath Wrapper` em contextos lectivos e aplicacinais, surgiu a ideia de tentar adaptá-lo para permitir colocar queries de uma só vez a todo um repositório documental em lugar de a um documento apenas.

Para isso, os vários componentes do sistema tiveram que sofrer alterações e alguns compromissos tiveram que ser estabelecidos.

Nas secções seguintes, descrevem-se as alterações, mais profundas nuns casos do que noutros, introduzidas nos vários componentes da ferramenta.

3.1 Especificação de queries

As diferenças tiveram que começar a ser introduzidas logo no módulo inicial. Assim, a linguagem para especificação de queries sofreu algumas alterações no sentido de generalizar o sistema de interrogação e de permitir uma maior facilidade na composição dos resultados finais.

Na figura 2, apresenta-se a estrutura do XML Schema desenvolvido para a nova linguagem.

Em que:

– `new-doc` é o elemento raiz da especificação e é composto por um subelemento `target`, que identifica o elemento raiz do documento resultado, e por um mais elementos `part`, cada um correspondendo a um conjunto de queries.

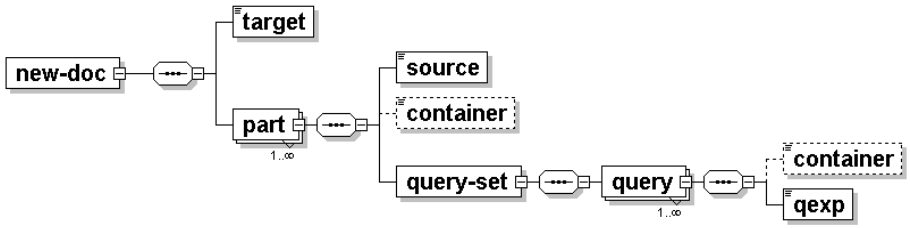


Figura 2. Estrutura da *Query Language* do `Document Composer`

– Cada elemento `part` é constituído por um triplo: um elemento `source` que indica o documento XML sobre o qual irão ser calculadas as queries, um elemento opcional `container` que identifica o elemento no documento resultado que irá conter os resultados deste conjunto de queries e, um elemento `query-set` que contem um conjunto de queries.
– O elemento `query-set` manteve a sua definição anterior e é composto por um ou mais elementos `query`.
– O elemento `query` é composto por um par: um elemento `container` que é opcional e que indica o elemento no documento resultado que irá conter o resultado da query em causa e, um elemento `qexp` que irá conter a expressão XPath que especifica a query.

O exemplo seguinte mostra uma aplicação concreta do sistema.

Exemplo 4: Lista de publicações e autores

Imagine que guardava num documento XML o registo das suas publicações. Para optimizar, um autor só é preenchido na primeira vez que aparece, nas vezes subsequentes é utilizado por referência. Acabou a de receber uma solicitação para o relatório anual: a sua lista de publicações no ano de 2003. Aparentemente bastaria uma query mas devido às referências aos autores é preciso criar um novo documento com duas partes: a lista de publicações e a lista de autores. A especificação em `Document Composer` seria a seguinte:

```
1  <?xml version="1.0" encoding="UTF-8"?> <new-doc>
2    <target>minhaspubs2003</target>
3    <part>
4      <source>minhaspubs.xml</source>
5      <container>autores</container>
6      <query-set>
7        <query>
8          <qexp>//author</qexp>
9        </query>
10     </query-set>
11   </part>
```

```
12   <part>
13     <source>minhaspubs.xml</source>
14     <container>pubs</container>
15     <query-set>
16       <query>
17         <qexp>//*[year='2003']</qexp>
18       </query>
19     </query-set>
20   </part>
21 </new-doc>
```

Está-se a criar um documento com a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?> <minhaspubs2003>
  <autores>
    Todos os nodos referentes a autor
  </autores>
  <pubs>
    Todos os nodos com um filho year
    com conteúdo igual a 2003
  </pubs>
</minhaspubs2003>
```

Como se viu a principal modificação para além da especificação do documento fonte sobre o qual se querem calcular as queries, foi a introdução opcional da identificação de um elemento que irá conter os resultados de um grupo de queries.

3.2 Cálculo das queries

Para calcular as queries vai-se, de novo, gerar uma stylesheet XSL específica e proceder da mesma maneira. Como podem existir vários documentos de entrada, a pipeline de execução irá usar um documento XML produzido para este efeito e que nem sequer é analisado: `comp-driver.xml`.

Como na discussão do `Xpath Wrapper` não se apresentou a meta-stylesheet vai-se agora apresentá-la em pequenos excertos.

A template principal cria o elemento raiz (linha 15) e gera as várias travessias (linhas 16) para processar cada conjunto de queries.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:axsl="http://www.w3.org/1999/XSL/TransformAlias">
  <xsl:namespace-alias stylesheet-prefix="axsl" result-prefix="xsl"/>

  <xsl:output method="xml" omit-xml-declaration="no"
              standalone="yes" indent="yes"/>

```

```
9   <xsl:template match="/">
10    <axsl:stylesheet version="1.0">
11      <axsl:output method="xml" omit-xml-declaration="no"
12        encoding="iso-8859-1" standalone="yes" indent="yes"/>
13      <axsl:template match="/">
14        <xsl:element name="{new-doc/target}">
15          <xsl:apply-templates select="new-doc/part"  mode="do-all-parts"/>
16        </xsl:element>
17      </axsl:template>
18      <xsl:apply-templates/>
19      <axsl:template match="text()" priority="-1">
20        <!-- strip characters -->
21      </axsl:template>
22    </axsl:stylesheet>
23  </xsl:template>
```

A template que trata os conjuntos de queries, elemento `part`, apenas coloca os resultados daquelas como conteúdo do elemento identificado pelo utilizador em `container`.

```
24 <xsl:template match="part" mode="do-all-parts">
25     <xsl:choose>
26       <xsl:when test="container">
27           <axsl:element name="{container}">
28               <xsl:apply-templates mode="do-all-parts" select="query-set"/>
29           </axsl:element>
30       </xsl:when>
31       <xsl:otherwise>
32            <xsl:apply-templates mode="do-all-parts" select="query-set"/>
33       </xsl:otherwise>
34     </xsl:choose>
35 </xsl:template>
```

Cada query é calculada sobre o documento identificado em `source` (linha 40) e numa travessia distinta (linha 39).

```
36 <xsl:template match="query" mode="do-all-parts">
37   <axsl:apply-templates
38     mode="P{count(../../preceding-sibling::*)}Q{count(preceding-sibling::*)}"
39     select="document('{../../source}')"/>
40 </xsl:template>
```

Por fim, cada query é calculada na sua travessia específica, e o respectivo resultado é colocado no elemento `container` se este foi definido pelo utilizador (linhas 49-51), caso contrário, o resultado é simplesmente concatenado ao documento resultado (linha 54).

```
41 <xsl:template match="query">
42   <axsl:template
43     mode="P{count(../../preceding-sibling::*)}Q{count(preceding-sibling::*)}"
44     match="{qexp}">
```

```
45
46      <xsl:choose>
47        <xsl:when test="container">
48          <xsl:element name="{container}">
49            <axsl:copy-of select="."/>
50          </xsl:element>
51        </xsl:when>
52        <xsl:otherwise>
53          <axsl:copy-of select="."/>
54        </xsl:otherwise>
55      </xsl:choose>
56    </axsl:template>
57  </xsl:template>
```

Nesta breve apresentação, foram omitidos alguns pormenores como a geração de comentários que documentam o processo e a filtragem dos nodos textuais que estão fora do contexto das expressões de query.

3.3 Produção de resultados

Uma vez que o utilizador pode "interferir"na estrutura do documento final de resultados não há possibilidade de definir uma stylesheet normalizada para os resultados. No entanto, pode sempre utilizar-se a stylesheet desenvolvida para o `XPath Wrapper` ou desenvolver outra. Para se poder utilizar a stylesheet do `XPath Wrapper` a especificação `Document Composer` teria de ser:

```
1  <?xml version="1.0" encoding="UTF-8"?> <new-doc>
2    <target>doc-result</target>
3    <part>
4      <source>fonte.xml</source>
5      <container>result-set</container>
6      <query-set>
7        <query>
8          <container>result</container>
9          <qexp>XPath exp</qexp>
10       </query>
11     </query-set>
12   </part>
13 </new-doc>
```

3.4 Pipeline de execução

O Pipeline de execução é muito semelhante ao do `XPath Wrapper` e está, neste momento, implementado através dum ficheiro `bat`.

4 Conclusão e Trabalho Futuro

Quer o `XPath Wrapper` quer o `Document Composer` têm sido bastante úteis em contextos lectivos no entanto, acreditamos que também podem sê-lo em contextos

aplicacionais. Ainda não houve oportunidade de fazer-lhes testes sérios neste último domínio. Até agora, foram usados em exemplos de pequena dimensão mas está prevista a sua aplicação e mesmo inclusão em sistemas de maior porte como por exemplo um sistema para interrogação de um reposiório de documentos XML.

Quanto a trabalho futuro, existem muitas linhas de trabalho que se poderão desenvolver:

- Desenvolver duma interface gráfica integrada com o `Document Composer` que transforme este numa ferramenta WYSIWYG de composição documental.
- Desenvolver o suporte a Namespaces no `Document Composer`. Tornaria possível a optimização do número de travessias que se fazem para cálculo das queries.
- Permitir a associação de novos atributos aos elementos `container`.
- Normalizar os nomes dos elementos na linguagem de especificação, criar um Namespace próprio para a aplicação e criar uma lista de casos de uso.

Brevemente, estas ferramentas ficaão disponíveis na Web.

Referências

[Alt] Xmlspy website. http://www.altova.com.

[DuC02] Bob DuCharme. Maintaining schemas for pipelined stages. In *Proceedings of the XML Conference 2002*, Baltimore, USA, 2002.

[Jel04] Rick Jelliffe. Document schema definition language (dsdl) - part 3: Rule-base validation (schematron). `http://www.jtc1sc34.org/repository/0524c.htm`, Jun 2004.

[KK03] Oleg Kiselyov and Shriram Krishnamurthi. Sxslt: Manipulation language for xml. *Lecture Notes in Computer Science*, 2562:256 – 272, Jan 2003.

[Kos03] Jirka Kosek. Xslt reflection. `http://www.xml.com/pub/a/2003/11/05/xslt.html`, Nov 2003.

[Nic00] Miloslav Nic. Graphotron. `http://www.zvon.org/ZvonSW/ZvonGraphotron/`, 2000.

[PBG01] Mikael Peltier, Jean Bézivin, and Gabriel Guillaume. Mtrans: A general framework, based on xslt,for model transformations. In *WTUML01, Proceedings of the Workshop on Transformations in UML*, Genova, Italy, 2001.

[PS03] Michael Pediaditakis and David Shrimpton. Device neutral pipelined processing of xml documents. In *Poster Proceedings of the Twelfth International World Wide Web Conference, WWW2003*, 2003.

[Ram01] José Carlos Ramalho. Constraining content: Specification and processing. In *Proceedings of XML Europe 2001*, Berlin, Germany, May 2001.

[RTFR04] José Carlos Ramalho, Pedro Taveira, Ricardo Ferreira, and Vasco Rocha. Gerador de web services para cadeias de transformações de documentos xml. In *XATA2004, XML Aplicações e Tecnologias Associadas, 2ª Conferência Nacional*, Faculdade de Engenharia da Universidade do Porto, Fevereiro 2004.

Construção e utilização de um protótipo para o processamento da linguagem de interrogação IXDIRQL

Alda Lopes Gançarski[1] and Pedro Rangel Henriques[2]

[1] Laboratoire d'Informatique de Paris 6, Paris, France
email : Alda.Lopes@lip6.fr
[2] Universidade do Minho, Braga, Portugal
email: prh@di.uminho.pt

Abstract. O XPath é uma linguagem de interrogação para acesso a componentes de documentos XML usando restrições estruturais e sobre o conteúdo. A linguagem de interrogação IXDIRQL estende o XPath com *operações de similaridade textual* típicas da Recuperação de Informação e com *operações de selecção* que permitem uma construção interactiva das perguntas manipulando os resultados intermédios. O protótipo, criado para o processamento da linguagem IXDIRQL, foi posto à disposição de utilizadores reais para realizar inúmeros testes. Por um lado, quis-se verificar o seu correcto funcionamento; por outro lado, pretendeu-se experimentar a facilidade de compreensão da própria linguagem IXDIRQL, analisando a capacidade dos utilizadores para formularem as perguntas correctas face a determinados pedidos pré-definidos. Neste artigo vamos apresentar alguns aspectos que julgamos particularmente interessantes sobre a construção do protótipo e sua utilização.

1 Introdução

A **recuperação de informação (RI)** [1] (*Information Retrieval* em inglês) consiste na pesquisa e entrega ao utilizador dos documentos que se julgam relevantes para satisfazer o seu pedido formulado através de uma pergunta (ou interrogação, do inglês *query*). Uma pergunta consiste numa expressão em linguagem natural que descreve o assunto procurado. Usualmente, a relevância dum documento é obtida através duma função de similaridade entre o documento (uma sua síntese, ou caracterização) e a pergunta. Os documentos são entregues ao utilizador ordenados por ordem decrescente da sua relevância.

Para tirar partido da informação estrutural dos documentos XML, o formato das perguntas foi enriquecido de modo a aceder-se a partes específicas dos documentos. Uma pergunta consiste, agora, numa sequência de restrições estruturais (caminhos, filtros, grupos) sobre o conteúdo (comparações com valores textuais ou numéricos). O XQuery é proposto pelo *W3C Consortium* como a norma de interrogação para XML [3]. O XQuery baseia-se em diferentes linguagens existentes, entre as quais o XPath [2] e o XML-QL [5]. Estas linguagens são

consideradas como de *recuperação de dados* (em inglês, *data retrieval*) porque a resposta à pergunta é um conjunto com todos os elementos que *satisfazem* literalmente a respectiva pergunta, indistintamente. Não há, portanto, nestes sistemas a noção de relevância, ou seja não há qualquer indicação sobre as melhores ou piores respostas. Alguns trabalhos [7] [4] propuseram extensões a estas linguagens para incluir restrições de similaridade textual cujo resultado é uma lista de elementos ordenados pela sua relevância. Recentemente, o sistema de RI associado à linguagem IXDIRQL [8, 9], baseada no XPath, propõe, além da similaridade textual, um paradigma interactivo na construção das perguntas. Esta abordagem permite a manipulação dos resultados intermédios, i.e. os resultados obtidos numa fase podem ser tomados em consideração (é possível seleccionar o sub-conjunto de elementos interessantes) para refinar a pergunta seguinte.

Este artigo apresenta alguns aspectos que nos pareceram particularmente relevantes sobre a construção e utilização de um protótipo para o processamento da linguagem IXDIRQL. Na secção 2, a linguagem é introduzida. Depois, a secção 3 mostra como são calculadas as relevâncias dos elementos perante uma operação de similaridade textual e como são propagadas ao longo das perguntas. A secção 4 introduz o conceito de índice e a sua relevância para optimizar a operação de procura. O editor IXDIRQL associado a um processador incremental é o tema da secção 5. A secção 6 aborda uma questão fundamental para o sucesso desta abordagem: a apresentação dos resultados. O protótipo construído foi posto à disposição de utilizadores para testar a sua funcionalidade e a facilidade com que as operações de selecção são usadas em determinados pedidos de informação. Esta experiência é descrita na secção 7. O artigo termina com uma breve conclusão, dando directrizes para trabalho futuro.

2 A linguagem IXDIRQL

Como se explica ao pormenor em [9], a linguagem IXDIRQL estende o XPath com os seguintes operadores de similaridade textual sobre elementos:

ROSearch Procura elementos de qualquer tipo onde os termos indicados ocorrem. Por exemplo, $ROSearch\ 'XML'$ procura elementos de qualquer tipo sobre *'XML'*.

ROSearchTE Semelhante ao anterior, procura elementos de qualquer tipo onde os termos indicados ocorrem, mas agora apresenta-os ordenados em listas diferentes, uma para cada tipo de elemento. Por exemplo, $ROSearchTE\ 'XML'$ procura elementos de qualquer tipo onde o termo *'XML'* ocorre, sendo o resultado uma lista ordenada de artigos, outra de secções, outra de referências, etc.

Sim Procura elementos de um tipo especificado onde o termo indicado ocorre. Este operador é integrado nas perguntas estruturadas do XPath. Por exemplo, $/artigo/titulo\ Sim\ 'XML'$ procura elementos do tipo *titulo* que contenham no seu interior o termo *'XML'*. O resultado é uma lista de títulos ordenados por ordem decrescente de relevância.

Este artigo inside apenas na operação *Sim*.

Os elementos resultantes destes três operadores são associados a uma relevância cujo valor está no intervalo [0, 1]: 0 para elementos não relevantes, 1 para elementos totalmente relevantes e valores intermédios para diferentes níveis de similaridade textual entre o conteúdo do elemento específico e a expressão em linguagem natural pedida. As operações importadas do XPath conduzem sempre a uma relevância 1 porque os elementos encontrados satisfazem completamente as restrições estruturais e sobre o conteúdo. As operações lógicas importadas do XPath retornam agora, não valores booleanos (verdadeiro e falso), mas *valores de verdade* representados pela relevância R associada, o que significa *'verdadeiro com probabilidade R'*. Assim, verdadeiro é o valor de verdade de relevância 1 e falso é o valor de verdade de relevância 0. Valores de verdade com relevâncias entre 0 e 1 podem ocorrer em expressões lógicas como

$$titulo\ Sim\ 'XML'\ \$and\$\ autor =' Silva'$$

Neste caso, o resultado é uma série de valores de verdade calculados tendo em conta a relevância dos títulos.

Face aos resultados intermédios das perguntas, o sistema IXDIRQL permite, graças ao seu carácter interactivo e incremental, três operadores de selecção:

Select Selecciona entre os elementos apresentados aqueles que o utilizador achou interessantes; a selecção é feita indicando a respectiva chave (o identificador único de cada elemento). Por exemplo

$$artigo/titulo\ Select\ tit\_4, tit\_8$$

selecciona os títulos tit_4, tit_8 do conjunto de títulos encontrados.

SelectN n Selecciona o conjunto dos primeiros n elementos. Por exemplo, a pergunta

$$/artigo[autor =' Silva']\ SelectN\ 10$$

conduz aos 10 primeiros artigos do autor *'Silva'* encontrados na colecção.

JudgeRel Restrinje a lista de documentos resultante de uma operação de similaridade textual (*Sim*) previamente realizada, por selecção explícita dos elementos que pertencem aos documentos que aparentam ser mais interessantes para as necessidades do utilizador. Por exemplo,

$$artigo[titulo\ Sim\ 'XML'\ JudgeRel\ tit\_4, tit\_8]/referencia$$

é uma pergunta que permite obter a lista de de referências bibliográficas citadas nos dois artigos cujos títulos (contendo *'XML'*) parecem ser os mais relevantes. Primeiro, obtem-se, com o operador *Sim*, uma lista de títulos (sobre *'XML'*) ordenada pela relevância. Depois, através do operador *JudgeRel*, o utilizador escolhe o conjunto de títulos dessa lista que ele considera importantes. Por fim, obtem-se a lista de referências bibliográficas citadas nos artigos correspondentes aos títulos relevantes.

3 O cálculo das relevâncias

Para incluir a similaridade textual numa linguagem de interrogação para XML houve necessidade de adaptar as fórmulas de cálculo da RI tradicional para o formato estruturado dos documentos.

3.1 Na recuperação de informação tradicional

Em RI tradicional [1], um termo com uma frequência grande num documento é seguramente um bom representante desse documento. Sejam: $\{d_1, ..., d_N\}$ um conjunto de documentos; $\{t_1, ..., t_T\}$ um conjunto de termos; n_i o número de documentos onde aparece o termo t_i (i=1,...,T); $freq_{ij}$ a frequência do termo t_i no documento d_j (j=1,...,N). A frequência tf_{ij} **normalizada** do termo t_i no documento d_j é calculada por:

$$tf_{ij} = \frac{freq_{ij}}{\max_{l=1,\ldots,T} freq_{lj}}$$

Um termo que aparece numa pequena fracção da colecção de documentos tem um bom **poder discriminante** dessa fracção em relação à colecção. O poder discriminante do termo t_i é calculado por:

$$idf_i = \log N/n_i$$

A multiplicação das duas medidas (*tf*idf*) é usada frequentemente para a representação dos termos e é referida como *tf.idf*. Os documentos e as perguntas em linguagem natural são muitas vezes representados por vectores cujos componentes são as medidas tf.idf dos termos relativamente ao documento (ou pergunta) em causa. Neste modelo vectorial, a relevância de um documento em relação a uma pergunta é o resultado de uma função de correlação entre os vectores correspondentes. Uma função simples, eficiente e, portanto, normalmente utilizada é o cosseno. A similaridade textual entre o documento d_j (j=1..N) e e pergunta q é, então, estimada pela expressão seguinte:

$$sim(d_j, q) = \frac{d_j \times q}{|d_j| \times |q|}$$

Seja w_{ij} a medida tf.idf do termo t_i (i=1..T) no vector d_j (j=1..N). A similaridade calcula-se, então, por:

$$sim(d_j, q) = \frac{\sum_{i=1}^{T} w_{ij} \times w_{iq}}{\sqrt[2]{\sum_{i=1}^{T} w_{ij}^2} \times \sqrt[2]{\sum_{i=1}^{T} w_{iq}^2}}$$

3.2 Na recuperação de informação em XML

Tendo em conta que a relevância no IXDIRQL é calculada para os elementos, a fórmula tf.idf foi adaptada para que as frequências sejam calculadas no conjunto dos elementos (e não dos documentos) da colecção. Assim, sejam: Ne o número de elementos da colecção; n_i o número de elementos onde o termo t_i aparece (i=1..T); $freq_{ij}$ a frequência do termo t_i no elemento e_j (j=1..Ne). A representação tf.idf passa a chamar-se *tf.ief* onde:

$$tf_{ij} = \frac{freq_{ij}}{\max_{l=1..T} freq_{lj}} \quad \text{e} \quad ief_i = \log Ne/n_i.$$

Para combinar as relevâncias dos resultados de diferentes operações, são usadas seguintes fórmulas de cálculo probabilístico para a disjunção e a conjunção de eventos [7]:

$$P(p_1 \vee ... \vee p_n) = \sum_{i=1}^{n} (-1)^{i-1} (\sum_{1 \leq j_1 < ... < j_i \leq n} P(p_{j1} \wedge ... \wedge p_{ji}))$$

$$P(p_1 \wedge ... \wedge p_n) = P(p_1) \times ... \times P(p_n), \textit{ assumindo } p_1, ..., p_n \textit{ independentes.}$$

Um evento é o facto de um elemento ser relevante. A relevância é a probabilidade associada a um evento, i.e. a probabilidade de que um elemento seja relevante. Por exemplo, seja a pergunta

$$/artigo\ Sim\ 'XML'/referencia\ Sim\ 'XSL'$$

Há aqui dois predicados de similaridade textual, um associado aos artigos, outro às referências. Suponhamos que o artigo a_1 tem relevância $P(a_1)$ interpretada como a probabilidade de que a_1 seja relevante em relação ao assunto *'XML'*. Suponhamos, também, que a referência bibliográfica r_1 é citada no artigo a_1 e tem relevância $P(r_1)$. $P(r_1)$ é a probabilidade de que r_1 seja relevante em relação ao assunto *'XSL'*. No resultado final, há que combinar as duas relevâncias de forma a calcular a relevância total de cada referência. Isso é feito usando $P(a_1 \wedge r_1) = P(a_1) \times P(r_1)$ que traduz a probabilidade de que o artigo a_1 e a referência r_1 sejam ambos relevantes.

Quando o operador *Sim* é seguido do operador *JudgeRel*, primeiro é calculada a relevância associada ao operador *Sim*; depois, a relevância de cada elemento seleccionado por *JudgeRel* é substituída por 1, sendo 0 a dos restantes.

Quanto aos valores de verdade, eles resultam das operações lógicas do IXDIRQL. Se não houver nenhum predicado de similaridade na pergunta, a relevância associada aos valores de verdade é 1. Caso contrário, a relevância é calculada pelas fórmulas de cálculo probabilístico apresentadas para os elementos, assumindo agora um evento como o facto do valor de verdade ser o valor lógico verdadeiro.

4 Os índices de texto e de estrutura

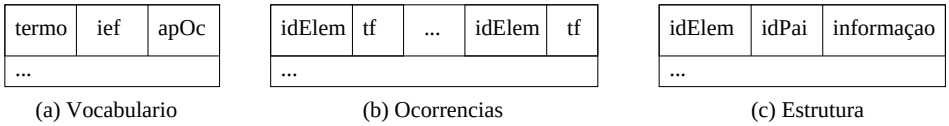
Em RI, a informação sobre os termos da colecção é guardada em *índices* para um rápido processamento das perguntas. Os termos são extraídos a partir de

operações efectuadas sobre certas palavras encontradas nos documentos. As palavras seleccionadas excluem as proposições, os determinantes, os pronomes e outras consideradas de *significado 'vazio'* (*sem interesse*) para a RI. As palavras seleccionadas são, depois, reduzidas à sua raíz etimológica. Estas operações reduzem o tamanho dos índices a cerca de 40%.

Em relação aos documentos textuais, os índices mais usados são os *inverted files*. Uma *inverted file* é composta por dois ficheiros: o *vocabulário*, que guarda a informação sobre os termos; e o das *ocorrências* (*postings* em inglês), que assinala as posições de cada termo nos documentos.

No protótipo do IXDIRQL, a representação vectorial *tf.ief* é guardada, conforme manda a tradição, nas tabelas `vocabulário` e respectivas `ocorrências`. Como se pode ver na figura 1, o vocabulário (a) associa a cada termo o seu valor de *ief* e um apontador para a lista de ocorrências respectivas (*apOc*). Esta lista, representada na figura 1 (b) contém, agora, nas suas células os identificadores dos elementos onde o termo aparece (*idElem*) e o respectivo valor de *tf*.

Neste caso criou-se, ainda, um índice de estrutura (figura 1, c) que associa a cada elemento (*idElem*) o seu pai (*idPai*) e informação (*informação*) que inclui o seu tipo, um apontador para o próximo elemento do mesmo tipo e os atributos XML respectivos. Este índice permite o acesso aos elementos existentes na colecção para se escolherem aqueles que satisfazem cada operação, de acordo com o seu tipo e as suas relações hierárquicas. A relevância associada aos elements obtidos numa operação de *Sim* é calculada usando em paralelo os índices de estrutura e de texto (o vocabulário e as ocorrências, onde se faz a correspondência entre os termos e os elementos).



(a) Vocabulario (b) Ocorrencias (c) Estrutura

Fig. 1. Os índices de texto (vocabulário (a) e ocorrências (b)) e de estrutura (c).

5 O editor/processador do IXDIRQL

Para a escrita e o processamento de perguntas IXDIRQL, foi criado automaticamente um editor associado a um processador incremental. Para isso, foi usado o LRC [10], um gerador de ambientes incrementais baseados na definição formal da linguagem em causa. A definição da linguagem é feita através duma gramática de atributos onde a semântica dinâmica corresponde ao cálculo do resultado das perguntas. A meta-linguagem usada para definir essa gramática é a *Synthesizer Specification Language* (SSL) [11]. Para dar uma ideia do estilo da especificação SSL da gramática abstracta IXDIRQL, considere-se o símbolo não-terminal *Operation* que permite a derivação de uma operação de conjuntos, de

uma operação lógica ou de um caminho. A produção (designada por *OPath*) relativa à derivação de *Operation*, a um caminho (*Path*) escrita em SSL é a seguinte :

$$
\begin{aligned}
&Operation : OPath\ (Path)\\
&\qquad \{\ Operation.aRes = Path.aRes;\\
&\qquad Path.aContext = Operation.aContext;\\
&\qquad Path.aPathOp = Operation.aPathOp;\ \};
\end{aligned}
$$

As regras semânticas associadas a esta produção (entre '{' e '}') calculam os valores do atributo sintetizado *aRes* (guarda o resultado do caminho) e dos atributos herdados *aContext* (guarda o conjunto de elementos a partir dos quais a procura dos novos elementos é feita) e *aPathOp* (corresponde ao operador de caminho filho ou descendente).

O editor construído é estruturado, sendo a edição dirigida pela sintaxe do IXDIRQL. Os símbolos não-terminais e pseudo-terminais da gramática são associados a indicações de introdução de informação. Estas consistem numa expressão que sugere o tipo de informação a introduzir, entre < e >. Quando um símbolo não-terminal pode ser derivado por diferentes produções alternativas, o utilizador escolhe a que lhe interessa num menu, como o que é mostrado na figura 2. Nesta figura, uma operação estruturada (derivada pelo símbolo *Operation* já referido) é indicada por <*Operation?*> e pode ser uma das seis alternativas do menu. Estas alternativas correspondem às diferentes produções em que *Operation* deriva, dentre as quais a produção que deriva o caminho dada atrás como exemplo. Se a operação escolhida no menu for uma intersecção, o editor passa a ter o aspecto da figura 3, onde um dos operandos da intersecção foi já introduzido (*article*/*ref*).

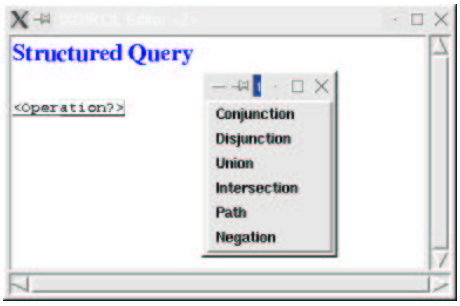


Fig. 2. O editor da linguagem IXDIRQL mostrando uma indicação de introdução de informação.

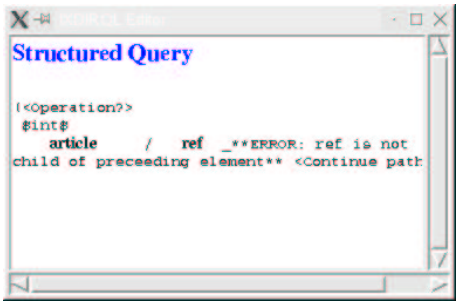


Fig. 3. O editor da linguagem IXDIRQL mostrando uma intersecção.

Todos os símbolos terminais da gramática são automaticamente colocados no sítio correcto, dispensando-se o utilizador dessa tarefa. Por exemplo, na figura 3, o operador *int* é introduzido automaticamente.

A semântica estática da linguagem IXDIRQL consiste na verificação da consistência entre os tipos de elementos introduzidos e as respectivas relações hierárquicas. Por exemplo, na figura 3, existe um erro semântico no caminho $/article/ref$ pois o tipo de elementos ref não é filho do tipo de elementos $article$. Durante a escrita de uma pergunta, estes erros são anunciados ao utilizador logo após a operação onde o erro é cometido. Assim, o utilizador pode corrigir imediatamente o seu texto.

6 Apresentação dos resultados

No protótipo construído, os resultados das perguntas são mostrados numa janela de visualização de documentos estruturados (foi escolhido o *Mozilla*). Após cada operação, a lista de elementos ou valores de verdade do resultado é incluída numa série de documentos HTML, dependendo do tamanho do resultado. A figura 4 é um exemplo de um resultado. Os resultados são ordenados por ordem decrescente da relevância dos elementos ou dos valores de verdade. Se as relevâncias forem iguais, é usada a ordem da colecção.

Na janela de visualização, cada elemento é apresentado com o seu identificador (que pode ser usado em operações de selecção), a sua relevância, um apontador para o documento respectivo e o seu conteúdo textual. Por sua vez, os valores de verdade são apresentados com a sua relevância, o identificador do elemento respectivo (por exemplo, o elemento associado ao filtro onde uma condição lógica é efectuada) e um apontador para o documento onde está inserido esse elemento. A figura 5 apresenta um exemplo de resultado composto por valores de verdade.

De momento, o acesso aos resultados intermédios é feito usando a janela de controle apresentada na figura 6. O utilizador acede ou não a cada resultado, respondendo às questões da janela. Quando o utilizador decide ver um resultado intermédio, este é-lhe mostrado, à parte, na janela de visualização (do tipo indicado nas figuras 4 e 5).

7 Utilização do protótipo por utilizadores reais

O protótipo foi posto à disposição de 5 utilizadores para efectuarem 3 procuras de informação pré-definidas. Por um lado, pretendeu-se testar a funcionalidade do sistema e, por outro, verificar se as perguntas introduzidas pelos utilizadores incluem operações de selecção. Isto significa que os utilizadores compreenderam as operações de selecção e souberam utilizá-las correctamente.

A interface do protótipo foi apresentada oralmente aos utilizadores. Os utilizadores foram escolhidos entre os membros do laboratório de informática onde

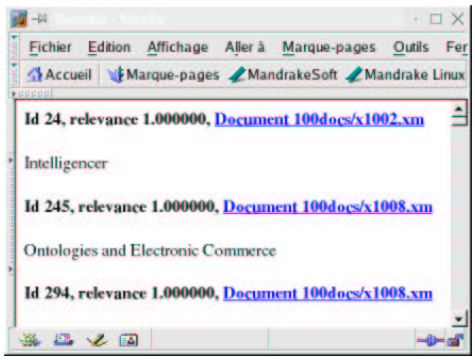


Fig. 4. A janela de visualização de um resultado composto por elementos.

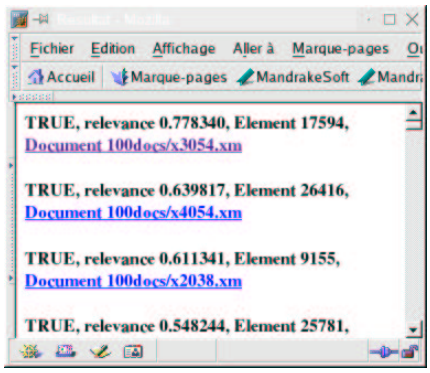


Fig. 5. A janela de visualização de um resultado composto por valores de verdade.

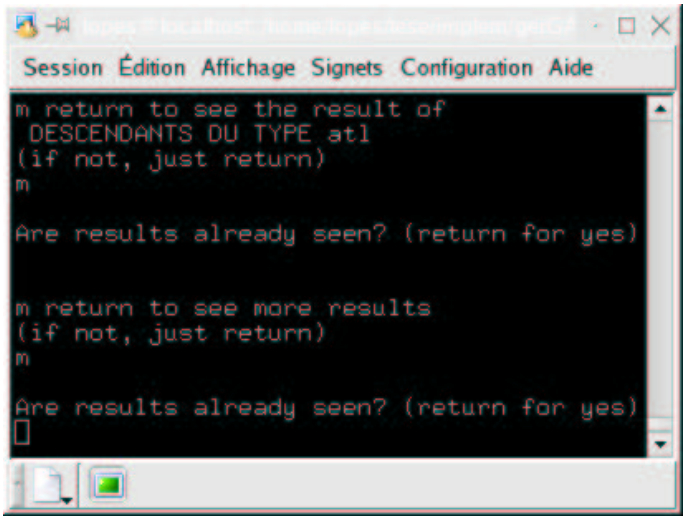


Fig. 6. A janela de controle para o acesso aos resultados intermédios.

o trabalho decorreu, com a condição de conhecerem XML e XPath. Assim, a linguagem IXDIRQL foi-lhes apresentada através de uma explicação sobre as operações de similaridade textual e de selecção.

A colecção de documentos utilizada consiste num conjunto de 100 artigos sobre *Inteligent Systems* da colecção INEX[3] [6], dos anos 2000 e 2001. Os artigos têm entre 110 e 2568 elementos. O DTD associado define, entre outros, os tipos de elementos e respectivas relações hierárquicas da figura 7.

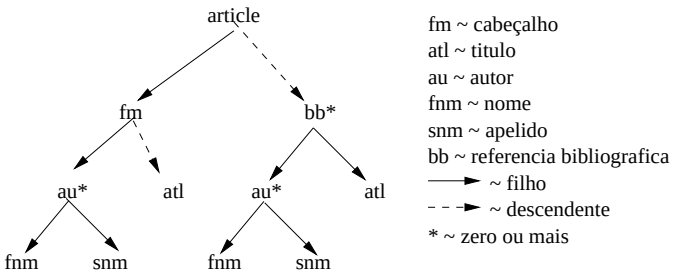


Fig. 7. Alguns tipos de elementos e relações hierárquicas do DTD INEX.

Na figura 8 pode ver-se um exemplo de documento da colecção onde existem elementos do tipo dos da figura 7. Este exemplo foi mostrado aos utilizadores, juntamente com a figura 7 e respectivas explicações.

```
<article>...
    <fm>...
        <atl>What's Next for the E-Book?</atl>
        <au><fnm>Giovanni</fnm><snm>Flammia</snm>...</au>...
    </fm>...
    <bb id="bibx10181">
        <au><fnm>B.N.</fnm><snm>Schilit</snm></au>
        <atl>"As We May Read: The Reading Appliance..."</atl>
    </bb>
    <bb id="bibx10182">
        <au><fnm>B-W.</fnm><snm>Chang</snm></au>...
        <au><fnm>J.</fnm><snm>Mackinlay</snm></au>...
        <atl>"Fluid Links for Informed and Incremental..."</atl>...
    </bb>...
</article>
```

Fig. 8. Um exemplo de artigo da colecção.

[3] Iniciativa para a avaliação da recuperação de informação em XML (*INEX: initiative for the evaluation of XML retrieval*), http://qmir.dcs.qmw.ac.uk/INEX/.

7.1 Os pedidos de informação

Foram propostos aos utilizadores três pedidos de informação baseados apenas nos tipos de elementos da figura 7. Esses pedidos foram definidos de forma a que o seu resultado implique a utilização de operações de selecção na construção das perguntas, a menos que os resultados sejam alterados manualmente. Assim, foram entregues aos utilizadores os pedidos seguintes:

1. Procurar as referências bibliográficas citadas em dois artigos do autor cujo nome é *"Giovanni"*.
2. Procurar as referências bibliográficas citadas em artigos cujo título é, na sua opinião, sobre *"XML"*.
3. A partir das referências bibliográficas obtidas em 2., procurar os autores daquelas cujo título é interessante para si (escolher pelo menos um título).

As perguntas que satisfazem os três pedidos anteriores são, respectivamente:

1. *article[fm//fnm='Giovanni'] SelectN 2//bb*
   article[fm//fnm='Giovanni'] Select $\{id_1, id_2\}$*//bb*
2. *article[fm//atl Sim 'XML' JudgeRel {...}]//bb*
3. *article[fm//atl Sim 'XML' JudgeRel {...}]//bb[atl Select {...}]//au*
   article[fm//atl Sim 'XML' JudgeRel {...}]//bb[atl Sim '...' JudgeRel {...}]//au

Na primeira pergunta, o operador *SelectN* permite reduzir o número de artigos do autor *"Giovanni"* a 2. Assim, as referências bibliográficas encontradas são citadas em 2 artigos. Outra possibilidade é fazer a selecção dos dois artigos através do operador *Select* aplicado a dois identificadores $\{id_1, id_2\}$.

A segunda pergunta procura, primeiro, os títulos de artigos sobre *"XML"* usando o operador *Sim*. De seguida, o operador *JudgeRel* reduz os artigos encontrados aos que o utilizador julga relevantes. Assim, as referências bibliográficas encontradas são as citadas nos artigos cujo título é, na opinião do utilizador, sobre *"XML"*.

Por fim, a terceira pergunta inclui o operador *Select* para que o utilizador escolha os títulos das referências bibliográficas que lhe interessam. Alternativamente, o utilizador pode fazer a procura de títulos usando *Sim* sobre um assunto que lhe interesse e seleccionar com *JudgeRel* aqueles que ele considera relevantes.

7.2 As perguntas feitas pelos utilizadores

As perguntas feitas pelos utilizadores podem pertencer a um dos quatro casos seguintes:

A: a pergunta inclui uma operação de selecção e satisfaz o pedido de informação;
B: a pergunta não inclui uma operação de selecção e não satisfaz o pedido de informação;

C: a pergunta inclui uma operação de selecção e não satisfaz o pedido de informação;
D: a pergunta não inclui uma operação de selecção mas satisfaz o pedido de informação porque o utilizador alterou manualmente os resultados.

A tabela 1 relaciona os 3 pedidos de informação e os 5 utilizadores através da classificação A a D da pergunta efectuada, indicando também os operadores de selecção que foram utilizados.

Pedido/Utilizador	**1**	**2**	**3**	**4**	**5**
1	A *SelectN*	A *Select*	A *SelectN*	A *SelectN*	A *Select*
2	A *JudgeRel*	A *JudgeRel*	B	B	B
3	A *JudgeRel*	A *Select*	B	A *Select*	A *Select*

Table 1. A classificação das perguntas feitas pelos utilizadores.

O comportamento dos utilizadores inclui aspectos que não podem ser controlados, como a concentração na execução da experiência, o nível de conhecimento do XPath, a compreensão do documento entregue sobre a experiência (que inclui explicações sobre IXDIRQL, a colecção de documentos e os pedidos de informação). Apesar desta dificuldade, pela tabela 1 constata-se o seguinte:

- Não há perguntas do tipo C. Todas as perguntas onde uma operação de selecção foi feita satisfazem o pedido, i.e. 100% das utilizações da selecção são correctas. Isto mostra que os utilizadores souberam sempre porque é que a selecção foi usada.
- O caso D nunca se verifica pois nenhum utilizador modificou os resultados manualmente. Em princípio, espera-se que um sistema de RI possa entregar o resultado desejado, sem ser necessário fazer alterações posteriormente.
- Nas 15 perguntas feitas pelos utilizadores, há 5 possibilidades de utilizar *SelectN* (no pedido 1), 1 de utilizar *Select* (nos pedidos 1 e 3) e 10 de utilizar *JudgeRel* (nos pedidos 2 e 3). As frequências verificadas são 2 em 5 para *SelectN* (40%), 6 em 10 para *Select* (60%) e 3 em 10 (30%) para *JudgeRel*. *Select* é o operador mais utilizado, o que pode mostrar que os utilizadores perceberam melhor a sua funcionalidade ou o julgam mais interessante ou fácil de aplicar.
- Por fim, o facto mais interessante é que, dentre as 15 perguntas efectuadas, 11 são do tipo A, i.e. em 73% das perguntas, os utilizadores recorrem à selecção correctamente.

8 Conclusão e trabalho futuro

Neste artigo pretendeu-se dar especial destaque a 2 assuntos relacionados com a recuperação incremental de informação em documentos estruturados, anotados

em XML.

Por um lado, pretendeu-se discutir algumas estratégias e decisões tomadas para implementar o sistema IXDIRQL de RI, montado sobre uma extensão ao XPath, que tem a capacidade de tratar perguntas baseadas na similaridade textual e oferece um modo de procura incremental. Nesta abordagem, que requereu uma adaptação dos cálculos de relevância tradicionais, uma pergunta pode ir sendo refinada através da selecção parcial dos resultados que se vão obtendo. Para o seu sucesso, enfatizámos a importância do recurso a 3 índices e a forma como os resultados parciais vão sendo mostrados no écran ao utilizador.

Por outro lado, falou-se do modo pragmático seguido numa primeira fase para validar o IXDIRQL, avaliando o comportamento de vários utilizadores a quem foi disponibilizado o sistema e fornecido um conjunto de perguntas.

Os testes realizados ao protótipo construído para o processamento da linguagem IXDIRQL mostraram que o mesmo satisfaz as características necessárias, pois permite o acesso aos resultados intermédios das perguntas e faz correctamente o cálculo de relevância de cada operação. Esse cálculo é incremental para que, após cada alteração da pergunta, apenas os cálculos dependentes do que foi alterado sejam efectuados. A utilização do protótipo por utilizadores mostrou, não só o seu correcto funcionamento, mas também que as operações de selecção são, em geral, correctamente utilizadas para obter o resultado de certas procuras de informação. Embora tenhamos consciência das limitações das experiências realizadas —as possíveis até à data— com um número reduzido de utilizadores, de questões e de documentos, o trabalho realizado mostrou o caminho que deve ser seguido nesta fase de validação da proposta e avaliação do sistema. Estes resultados, apesar de tudo, foram fundamentais para consolidar a proposta, evidenciando a sua exequabilidade.

Como trabalho futuro, apontamos alguns projectos que temos em mente, uns complementares, outros alternativos:

- Permitir a especificação de caminhos mostrando ao utilizador um documento exemplo da colecção onde ele selecciona os elementos desejados. Este mecanismo é conhecido em inglês por *"query by example"*.
- Estender o XQuery com operações de similaridade textual e com o paradigma interactivo de construção das perguntas (à semelhança do IXDIRQL). Construir um protótipo adequado para o processamento e avaliar a sua funcionalidade face a utilizadores reais.
- Desenvolver uma metodologia para utilizar o XQuery assim estendido para interrogação de fontes de informação organizadas com ontologias, como sucede na Semantic Web, criando assim um novo sistema a que chamámos OntIXQuery. Para tal, pretendemos explorar duas hipóteses: (1) fazer as perguntas aos documentos XML da Web e refinar a resposta obtida usando a meta-informação ontológica associada ao recurso; (2) fazer as perguntas à ontologia em uso para integrar os recursos e recuperar os documentos associados a cada componente da ontologia obtido como resposta.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.
2. A. Berglund, S. Boag, D. Chamberlin, M. Fernandez, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0 W3C Working Draft. http//www.w3c.org/xpath20/, October 2004.
3. S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C Working Draft. http://www.w3.org/TR/xquery/, October 2004.
4. T. T. Chinenyanga and N. Kushmerick. Expressive Retrieval from XML Documents. In *Proceedings of International ACM SIGIR Conference on Research and Development in Information retrieval (SIGIR'01)*, New Orleans, Louisiana, USA, September 2001.
5. A. Deutsch, M. Fernandez, A. Levy, and D. Suciu. XML-QL: A query language for XML. W3C Note, August 1998.
6. N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas. INEX: Initiative for the Evaluation of XML Retrieval. In R. Baeza-Yates, N. Fuhr, and Y. Maarek, editors, *Proceedings of the Workshop on XML and Information Retrieval, International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02),* , Tampere, Finland, August 2002.
7. N. Fuhr and K. Grobjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of International ACM SIGIR Conference on Research and Development in Information retrieval (SIGIR'01)*, New Orleans, Louisiana, USA, September 2001.
8. A. Gançarski and P. Henriques. Interactive Information Retrieval from XML Documents Represented by Attribute Grammars. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, Grenoble, France, November 2003.
9. A. Gançarski and P. Henriques. IXDIRQL: an Interactive XML Data and Information Retrieval Query Language. In *Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing*, Guimarães, Portugal, June 2003.
10. M. Kuiper and J. Saraiva. Lrc: A generator for incremental language-oriented tools. In K. Koskimies, editor, *7th International Conference on Compiler Construction*, volume 1383, pages 298—301. Lecture Notes in Computer Science, April 1998.
11. T. Reps and T. Teitelbaum. *The Synthesizer Generator Reference Manual.* Texts and Monographs in Computer Science. GrammaTech, Inc., 1993.

Inferência de tipos em documentos XML

José João Almeida and Alberto Manuel Simões

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

Resumo A estrutura dos documentos XML é descrita, habitualmente, em DTDs e/ou Schemas, o que permite ao programador estudar a forma de processamento estrutural mais correcta para o tipo de documento em causa.
No entanto, outros documentos há em que o tipo de documento não está definido e que obriga a analisar o documento para inferir a estrutura em causa. Paralelamente algumas especificações baseadas em Schemas tendem a ser de tal modo grandes que se tornam impossíveis de ler.
Neste documento pretendemos apresentar a ferramenta PFS[1], capaz de inferir tipos a partir de documentos XML, de mostrar de forma compacta essa informação e pretendemos ainda mostrar como esses tipos podem ajudar no processamento desses documentos (usando XML::DT com tipos).

1 Introdução

Como o XML é uma linguagem fortemente estruturada, a forma mais elegante de processar documentos XML é baseada na estrutura do documento. A estrutura de um documento não é mais do que um mapeamento de cada elemento a um tipo ou estrutura de dados.

Considere como exemplo o elemento `ul` do `XHTML`[5]. A estrutura de dados a si associada é uma lista, já que os seus filhos são etiquetas de um mesmo tipo (`li`).

Da mesma forma, ao elemento `item` do `RSS/RDF`[3] o tipo de dados associado é uma função finita, já que todos os seus filhos têm nomes diferentes e não são mais do que campos do elemento `item`.

Estes dois exemplos fazem parte de duas especificações da W3C que, como tal, estão bem documentadas e incluem DTDs e Schemas associados. No entanto, documentos há em que não existe um DTD associado. Para estes, é importante a existência de uma ferramenta que permita inferir os tipos e estruturas associadas a cada elemento.

Sem olhar para um DTD ou Schema, o tipo de um documento XML pode ser descrito formalmente por:

$$\begin{aligned} XML &\equiv elemento \\ elemento &\equiv nome \times atributos \times filho^{\star} \\ atributos &\equiv nomeAtributo \mapsto valorAtributo \\ filho &\equiv texto + elemento \end{aligned}$$

[1] *Pig from sausages.*

Esta estrutura arbórea permite descrever sintacticamente a informação, mas não lhe associa semântica.

Imagine-se a representação de uma lista de elementos em XML. A representação típica é:

```
<!ELEMENT lista (item*)>
```

onde aparece uma etiqueta externa (lista) e uma para cada elemento (item). Outros exemplos incluem listas directamente como filhos de elementos (sem a etiqueta externa):

```
<!ELEMENT data (a, b, c, item*, d, e)>
```

Da mesma forma, há uma série de propriedades de estrutura que não têm fácil expressão: a simples estrutura não nos diz se esta lista poderá ter elementos repetidos (se será um conjunto).

A definição de uma função finita[2] também pode tomar várias formas. Quando o domínio é fechado e de baixa cardinalidade, o mais simples será usar um conjunto de etiquetas (sem repetições) que servem de chaves:

```
<!ELEMENT hash (a, b, c, d, e, f, g)>
```

No entanto, nada obsta a que se defina funções finitas com outras sintaxes:

1.
```
<!ELEMENT hash (item+)>
<!ELEMENT item (#PCDATA)>
<!ATTLIST item key CDATA #REQUIRED>
```

2.
```
<!ELEMENT hash (key, data)+>
```

3.
```
<!ELEMENT hash (item+)>
<!ELEMENT item EMPTY>
<!ATTLIST item key  CDATA #REQUIRED
               data CDATA #REQUIRED>
```

Com toda esta panóplia de possíveis combinações de etiquetas, torna-se possível apresentar de formas muito diversas várias estruturas de dados simples. Por outro lado, torna-se impraticável detectar (adivinhar) automaticamente todos estes tipos de dados.

No entanto há várias situações que podem ser tratadas de modo homogéneo e automático, levantando duas questões diferentes:

- a sua descrição e processamento,
- a sua detecção automática

[2] Designaremos por **Função Finita de A para B**, as correspondências unívocas de A para B. Esta designação inclui as HASH do Perl, os Arrays associativos, os Mappings, etc.

1.1 Tipos de elementos

Esta secção discute alguns tipos de dados que conseguimos encontrar em documentos XML cuja detecção automática é viável.

Cada tipo corresponde a uma combinação de etiquetas XML que pretendemos ver tratadas como um todo, e corresponde ainda a uma estratégia de processamento dos filhos (tipicamente a um agrupar numa estrutura única os resultados dos processamentos dos filhos).

- **String** (STR)
  A maior parte dos elementos mistos irão, com certeza, cair neste tipo. Por exemplo, o elemento `p` do `XHTML` é um elemento misto cujos filhos são, por exemplo, `b`, `i` e outros, que deverão ser concatenados numa string depois de processados.

- **Sequência** (SEQ)
  Alguns elementos servem de delimitadores de uma lista de filhos, todos com o mesmo nome. Este é um caso bastante típico, e ao qual associamos o tipo *sequência*.
  Por exemplo, os elementos `ul` ou `ol` são do tipo *sequência*, já que são constituídos por uma lista de items (`li`).
  Um processador típico para este tipo de elemento terá de lidar com uma lista ou sequência de elementos.

- **Mapeamento** (MAP)
  Existem elementos que são delimitadores de um conjunto de vários elementos, todos com nomes diferentes. Por exemplo, o elemento `item` do `RSS/RDF` é um destes casos:

```
<item rdf:about="http://localhost/archives/2004/11/syndication.html">
  <title>Syndication</title>
  <link>http://localhost/archives/2004/11/syndication.html</link>
  <description>
     <![CDATA[<p>I forgot to told you, my dear readers, that this
      blog is syndicated on <a href="http://planet.botfu.org">Planet
      BotFu</a>, where all the folks from <tt>#botfu</tt> blog their
      lifes.</p>]]>
  </description>
  <dc:subject>Misc</dc:subject>
  <dc:creator>null</dc:creator>
  <dc:date>2004-11-12T11:05:52+00:00</dc:date>
</item>
```

- **Mapeamento Múltiplo** (MULTIMAP)
  Há casos em que um elemento contém várias etiquetas, mas cujos nomes se vão repetindo. Este modelo cria um mapeamento do nome da etiqueta para uma lista dos conteúdos de todas as etiquetas com esse nome.
  Ou seja, para um XML como

```
<exemplo>
  <bar> textoA </bar>
  <foo> textoB </foo>
  <bar> textoC </bar>
  <bar> textoD </bar>
  <foo> textoE </foo>
</exemplo>
```

construiríamos uma árvore como a seguinte:

```
{
  bar => ["textoA", "textoC", "textoD"],
  foo => ["textoB", "textoE"],
}
```

– **Mapeamento Múltiplo Selectivo** (MMAPON)
Este modelo de dados é o meio termo entre o mapeamento simples e o mapeamento múltiplo, permitindo ao utilizador especificar quais os elementos que vão ter repetições.
Ou seja, para um XML como

```
<exemplo>
  <bar> textoA </bar>
  <foo> textoB </foo>
  <bar> textoC </bar>
  <bar> textoD </bar>
</exemplo>
```

e usando o modelo `MMAPON(bar)`, construiríamos uma árvore como a seguinte:

```
{
  bar => ["textoA", "textoC", "textoD"],
  foo => "textoB",
}
```

– **Elemento Único** (THECHILD)
Embora pouco provável, especialmente em documentos bem estruturados, é possível em alguns formatos a existência de um elemento com apenas um filho...

2 XML::DT baseado em tipos

Embora o conhecimento dos tipos associados a cada elemento possa ajudar o programador na sua tarefa, este trabalho está ainda mais simplificado ao usar uma ferramenta que consiga tirar partido da associação de tipos aos elementos XML, como é o caso do XML::DT[1,4].

Antes da apresentação de como se pode inferir os tipos de documentos XML, torna-se imperativa a apresentação de alguns exemplos de como o XML::DT lida com eles.

2.1 Exemplo introdutório

Considere-se, por exemplo, o seguinte documento XML:

```
<?xml version="1.0"?>
<institution>
  <id>U.M.</id>
  <name>University of Minho</name>
  <tels>
    <item>1111</item>
    <item>1112</item>
    <item>1113</item>
  </tels>
  <where>Portugal</where>
  <contacts>J.Joao; J.Rocha; J.Ramalho</contacts>
</institution>
```

Neste exemplo, `institution` pode ser vista como uma função finita de elemento para o seu conteúdo, já que cada etiqueta está a definir propriedades (sem haver propriedades repetidas) de um mesmo objecto (uma instituição). Por outro lado, a etiqueta `tels` agrega uma lista de telefones.

Assim, um processador Perl usando o XML::DT para processar este documento pode ser escrito como:

```
#!/usr/bin/perl -w
use XML::DT;

%handler = ( -default => sub{$c},
             -type    => { institution => 'MAP',
                           tels        => 'SEQ' },
             contacts => sub{ [ split(";",$c)] },
           );

$a = dt("ficheiro.xml", %handler);
```

Este exemplo define que, por omissão (regra `-default`), o processamento de uma etiqueta é, simplesmente, devolver o seu conteúdo (`$c` - guarda o *contents*).

Por sua vez, que os tipos (regra `-type`) das etiquetas `institution` e `tels` são, respectivamente, uma função finita e uma sequência.

A regra seguinte, para a etiqueta `contacts` divide o seu conteúdo pelo caracter ponto-e-vírgula (split) retornando a lista de contactos.

O processamento do documento apresentado com este pequeno código, gera a seguinte estrutura de dados Perl:

```
{
  tels     => [ 1111, 1112, 1113 ],
  name     => 'University of Minho',
  where    => 'Portugal',
  id       => 'U.M.',
  contacts => [ 'J.Joao', ' J.Rocha', ' J.Ramalho' ],
}
```

2.2 Exemplo RSS

Para demonstrar a versatilidade do uso de tipos para o processamento de documentos XML, consideremos a transformação de um documento RSS para HTML.

```
#!/usr/bin/perl
use CGI qw/:standard/;
use XML::DT;

%hdl=(
    -default => sub {$c},

    -type => { channel   => 'MAP',
               item      => 'MAP', },

    'channel' => sub{
      h1(a({-href => $c->{link}},
           img({-src => $c->{image}}), $c->{title}))
    },

    'item' => sub {
      h3(a({-href => $c->{link}},$c->{title})).
        blockquote($c->{description})
    },

);

print dt(shift(), %hdl);
```

A estrutura do processador é semelhante à do exemplo anterior, com a definição de que os elementos `channel` e `item` são tratados como funções finitas. O primeiro elemento, define as propriedades do canal RSS que estamos a transformar, e cada `item` é uma das notícias/entradas do RSS.

As respectivas entradas para os elementos `channel` e `item` são funções de processamento que recebem como *contents* (`$c`) uma função finita (hash) e constroem o HTML respectivo.

3 Inferência de tipos com PFS

A inferência de tipos XML::DT partindo de um DTD (ou Schema) é quase directa mas, como vimos, nem sempre temos o DTD para analisar. O nosso objectivo é inferir tipos directamente do documento XML, e criar automaticamente um conjunto de vistas que permitam ao utilizador interpretar a estrutura do documento XML.

O Trang[2] é uma ferramenta desenvolvida por James Clark que converte DTD em Schemas e vice-versa, bem como noutros formatos. Uma das suas potencialidades é a adivinhação de um DTD ou Schema a partir de um documento XML. No entanto, o PFS pretende ser mais do que isso, já que deverá ser capaz de obter informação mais detalhada do que a definida num DTD ou Schema.

Figura 1. HTML Gerado de um documento RSS

3.1 Uso do PFS

Como primeiro exemplo, consideremos e exemplo do XML que define uma lista de números de telefone, que vimos anteriormente. O resultado do PFS sobre este documento devolve o seguinte resumo:

```
# institution ...Tue Nov 30 15:30:03 2004
institution  =>  tup(contacts, name, tels, id, where)
contacts     =>  text
name         =>  text
tels         =>  seq(item)
id           =>  text
where        =>  text
item         =>  text
```

Neste resumo da estrutura do documento são explicitados os elementos de texto (`#PCDATA`) bem como que o elemento `tels` contém uma sequência de items, e o `institution` é um tuplo de elementos (identificados pelo nome da respectiva etiqueta). Como vimos, a forma natural de representar este tuplo em Perl será o uso de uma função finita de nome de elemento para o seu conteúdo.

O resultado de analisar um documento RSS é um pouco mais complicado:

```
# rdf:RDF ...Tue Nov 30 15:46:19 2004
rdf:RDF     =>  mtup(channel, item+)
item        =>  tup(link, dc:creator, dc:subject, title, description,
                    dc:date) * rdf:about
channel     =>  tup(link, dc:creator, title, admin:generatorAgent,
                    description, dc:date, items) * rdf:about
dc:creator  =>  text
link        =>  text
dc:subject  =>  text
title       =>  text
dc:date     =>  text
description =>  text
admin:generatorAgent =>  empty * rdf:resource
items       =>  rdf:Seq
rdf:Seq     =>  seq(rdf:li)
rdf:li      =>  empty * rdf:resource
```

Neste exemplo já é possível ver alguns elementos com atributos (separados da definição do tipo de elemento por um asterisco) e alguns tipos mais complicados:

- o elemento `rdf:RDF` é um mapeamento múltiplo: é constituído por um `channel` e uma lista de `item`;
- o elemento `item` e o elemento `channel` são tuplos;
- os elementos `admin:generatorAgent` e `rdf:li` são elementos vazios;

O PFS pode ser invocado em modo *shell*, o que permite analisar ramos de uma árvore XML bem como atributos de determinados elementos.

```
[ambs@eremita XML-DT]$ pfs -shell index.rss
? item

item        =>  tup(link, dc:creator, dc:subject, title, description,
                    dc:date) * rdf:about
dc:creator  =>  text
link        =>  text
dc:subject  =>  text
title       =>  text
dc:date     =>  text
description =>  text

? li[@rdf:resource]

URL
```

```
http://null.perl-hackers.net/archives/2004/11/rsser.html
http://null.perl-hackers.net/archives/2004/11/palm_zire_31.html
http://null.perl-hackers.net/archives/2004/11/the_perl_review.html
```

Neste exemplo, o primeiro *query* permite analisar apenas a estrutura do elemento `item` (assim como os elementos necessários para a sua definição). O segundo exemplo, com uma sintaxe semelhante a XPath para referir um atributo de determinado elemento, permite inferir o tipo de dados (neste caso, URL) e mostrar o domínio activo deste atributo.

Da mesma forma que é possível obter um resumo de um documento, o PFS pode ser usado para a criação de um esqueleto de um processador em XML::DT ou para escrever um esqueleto de um DTD:

```
[ambs@eremita XML-DT]$ pfs -dt index.rss > rss.pl

[ambs@eremita XML-DT]$ pfs -dtd index.rss > rss.dtd
```

3.2 Exemplo Real

Uma outra aplicação do `pfs` é a análise reversa de documentos `HTML`. O processamento de páginas como a do Jornal Público difere muito do processamento de documentos bem formados `HTML 4.1` e muito mais do de documentos `XHTML`.

Para facilitar esta tarefa, o `pfs` pode ser usado para extrair a estrutura destes documentos (mesmo `HTML`). O exemplo seguinte foi obtido processando a página do Jornal Público online:

```
# html ...Mon Jan 24 17:33:10 2005
html    =>  tup(body, head)
body    =>  mtup(br, img, script, table+) * link * alink * vlink * topmargin
head    =>  mtup(link, meta+, script, title)
script  =>  text * language * src
br      =>  empty
table   =>  mtup(form?, tr*) * width * align * valign * height * border *
            bgcolor * cellspacing * cellpadding * id * class
img     =>  empty * width * vspace * alt * align * src * height * border *
            hspace
link    =>  empty * rel * href * type
title   =>  text
meta    =>  empty * http-equiv * content
tr      =>  mixed(td) * bgcolor * align * valign * class * height
form    =>  mtup(#PCDATA?, input?, tbody?, tr*) * action * name * id * method
td      =>  mixed(script, div, a, input, br, table, img, marquee, iframe, p,
            b, span, select) * width * onmouseover * align * valign * style *
            onmouseout * background * height * bgcolor * rowspan * colspan *
            class
tbody   =>  seq(tr)
input   =>  empty * value * name * onclick * class * type * id
div     =>  mtup(#PCDATA?, img?, script?) * align * valign
```

```
a       =>  mixed(br, strong, img, span) * target * href * onclick * title *
            class
marquee       =>  tup(p, #PCDATA) * direction * width * scrollamount * behavior *
            loop
iframe  =>  mixed(a, table, img) * width * scrolling * src * name * style *
            height * marginwidth * frameborder * id * marginheight
p       =>  mixed(br, a, strong, b, span) * align
b       =>  mixed(strong, b)
span    =>  mixed(span) * class
select  =>  seq(option) * onchange * name
strong  =>  text
option  =>  text * value
```

Da mesma forma, o `pfs` permite-nos consultar a estrutura do documento interactivamente,

```
? name

input(name):_id  = {Submit32, Submit22, respostas, pwd, rurl, Submit}
form(name):_id   = {seccao, frmVotacao, frmAction}
select(name):_id = {seccao, dia}
iframe(name):_id = {framePlus, barraplus}
```

3.3 Estrutura interna

A ferramente PFS internamente é um programa escrito em Perl e XML::DT que processa documentos XML, e HTML e constrói uma representação interna da meta-informação dos documentos.

Durante a análise dos documentos exemplo é calculado:

- estatísticas diversas
- o conjunto de possíveis raízes (quando são vários documentos)
- para cada elemento:
  - a lista de todos os seus atributos e o respectivo tipo (exemplo: int, url, email, id, str...) e domínio activo
  - o padrão de ocorrência dos filhos: o conjunto de listas de filhos que foram encontrados
- umas ordenações breath-first (para escolher uma ordem mais didáctica de mostrar resultados

Com o cruzamento das várias tabelas são em seguida criados vários resumos que são usados para geração do DTDs, da visão resumida mostrada nos exemplos acima e para os diálogos do modo *shell*

4 Conclusões

A análise da estrutura de um documento XML é complicada. Mesmo quando se tem um DTD ou Schema, se não se possuir uma ferramenta que esquematize

gráficamente a estrutura do documento, a análise manual é morosa e sujeita a erros.

A sintaxe usada pelo PFS é clara, minimalista e os resultados apresentados ordenados topologicamente, o que permite facilidade na sua interpretação.

Por sua vez, o PFS pode ser usado como ferramenta de *bootstrap* para a definição de um DTD (no caso de o ter definido de forma *bottom-up*, criando primeiro o documento XML) ou para a criação de um esqueleto de um processador desse tipo de documentos XML usando o XML::DT.

Referências

1. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, May 1999.
2. James Clark. Trang – multi-format schema converter based on RELAX NG. Technical report, Thai Open Source Software Center Ltd, 2003. `http://www.thaiopensource.com/relaxng/trang.html`.
3. RSS-DEV Working Group. RDF Site Summary (RSS) 1.0. Technical report, W3C - World Wide Web Consortium, 2001. `http://web.resource.org/rss/1.0/spec`.
4. Alberto Simões. XML::DT - down translating xml. *The Perl Review*, 1(1), Winter 2004.
5. W3C HTML Working Group. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). Technical report, W3C - World Wide Web Consortium, 2002. `http://www.w3.org/TR/xhtml1/`.

CBPEL - Linguagem para definição de processos de negócio interorganizacionais

António Teófilo[1] e Alberto Rodrigues da Silva[2]

[1] ISEL/DEETC, ateofilo@deetc.isel.ipl.pt
[2] IST/INESC-ID, alberto.silva@inesc-id.pt

Resumo: Têm emergido recentemente várias linguagens XML para descrever processos de negócio, nomeadamente: BPEL, WSCI, XLANG, ou BPML. Contudo, estas linguagens são inadequadas para descrever processos de negócio interorganizacionais, que são processos com vários participantes em que o controle de execução é partilhado por todos. Para processos com estes requisitos poderia utilizar-se as linguagens WS-CDL ou BPSS/ebXML, mas que apresentam algumas limitações na sua utilização. Este trabalho visa propor uma linguagem para modelar processos de negócio interorganizacionais, baseada nos conceitos da linguagem BPEL, à qual designamos por CBPEL (*Common Business Process Execution Language*).

1 Introdução

O desenvolvimento dos Serviços Web ("*Web Services*") [13] como uma tecnologia modular de interoperabilidade global, coloca-os como uma peça fundamental para a área dos processos de negócio (PN). A sua adopção como base para os processos descritos em notações *XML* [8], define a tecnologia de PN baseados em *XML*. Esta tecnologia visa a normalização da descrição, execução, e supervisão dos PN, proporcionando aos utilizadores um elevado grau de liberdade face às soluções proprietárias que caracterizaram a era antes-do-*XML*/*WS*, da qual se destaca somente as iniciativas de normalização do grupo *Workflow Management Coalition* [17].

A tecnologia de PN baseadas em *XML*, utiliza a pilha de protocolos básicos dos Serviços Web para os requisitos mais elementares, como: o *SOAP* [7], para a troca de mensagens; o *WSDL* [11], para a descrição dos serviços; e o *UDDI* [16] para publicação e descoberta de serviços. Pode, também, utilizar outros protocolos para providenciar características adicionais nas interacções, como por exemplo: o *WS-Sec*urity [5], para segurança, o *WS-ReliableMessaging* [6], para entrega fiável de mensagens, e o *WS-Transaction*[1] [9] para comportamento transaccional.

Actualmente existem várias linguagens *XML* para descrever PN que utilizam como tecnologia de suporte as várias especificações relacionadas com os Serviços Web, como por exemplo: *BPEL* [2], *XLANG* [15], *BPML* [3], *BPSS/ebXML* [10], *WS-CDL* [12], ou o *WSCI* [4].

[1] Esta designação abrange as especificações: WS-Coordination, WS-AtomicTransation e WS-BusinessTransaction

Contudo o foco deste artigo é sobre os PN que envolvem várias organizações na sua execução, e que são designados de PN interorganizacionais. Estes PN são caracterizados por possuírem uma execução repartida pelos vários participantes, existindo portanto uma distribuição da sua execução.

Verifica-se que as linguagens anteriormente mencionadas, com excepção das linguagens *WS-CDL* e *BPSS/ebXML*, estão vocacionadas para a composição de Serviços Web centrada num processo, pelo que são inadequadas para a descrição de PN interorganizacionais. Este trabalho visa, preencher esse espaço, propondo uma linguagem baseada na linguagem *BPEL* capaz de descrever processos interorganizacionais.

Este artigo tem cinco secções, em que a primeira introduz o contexto do artigo, a segunda discute a descrição dos processos interorganizacionais, a terceira apresenta a linguagem interorganizacional baseada na linguagem *BPEL*, a quarta contém um exemplo descrito nessa linguagem, e por último são apresentadas as considerações finais do trabalho.

2 Descrição de processos interorganizacionais

Em relação à descrição de processos interorganizacionais será primeiro apresentada a distinção para com os processos centralizados, e depois será apresentada a forma global unificada, que é uma forma de descrever os processos interorganizacionais de forma mais concisa e menos propensa a erros.

Como já referido algumas linguagens adoptam um ponto de vista centrado no processo em questão, e assumem que a interacção com os outros processo é coordenada por ele, e portanto, que os outros processos se comportam de forma passiva. Este modelo é designado de *composição* ou *orquestração* de Serviços Web. É um modelo em que os parceiros, do processo central, são entidades com uma lógica simplista tipo pedido-resposta. As linguagens BPEL, WSCI, XLANG e BPML são exemplos deste tipo de linguagem.

O contexto de um processo que envolva várias organizações, em que todas possuem parte do controle do processo global, implica que o seu controle encontra-se distribuído pelos vários participantes. Nestes contextos o modelo centralizado não pode ser aplicado directamente, pois cada participante não tem o controle absoluto do processo. Este contextos são designados de *coreografia* de processos, pois todos os parceiros são, ou implementam, processos de negócio.

A descrição dos processos interorganizacionais poderá assumir duas formas. Uma delas consiste no somatório das descrições individuais dos processos de todos os participantes envolvidos. Essa forma implica que os processos sejam concebidos em paralelo e que haja perfeita correlação em todos os aspectos, como por exemplo o fluxo de controle, e o fluxo de dados. A outra forma tem em conta que: as interacções têm de ser complementares nos parceiros envolvidos; e que o fluxo de controle pode ser partilhado entre os vários participantes. Pelo que o processo interorganizacional pode ser descrito por um único fluxo global em que as actividades de interacção indicam quem é o participante emissor e o receptor, e que posteriormente será decomposto nos

fluxos correspondentes a cada participante. A esta forma de descrever o processo interorganizacional designamos de forma global unificada.

2.1 A forma global unificada

A forma global unificada é uma forma de descrever os processos interorganizacionais, de um modo mais sintético. Um processo interorganizacional pode ser descrito na sua totalidade pelo somatório dos processos das várias entidades participantes. Contudo, à medida que os processos vão crescendo em dimensão, a complexidade da sua edição vai aumentando, tornando-a propensa a erros, e de difícil observação.

Assim torna-se premente uma representação mais sintética, e se possível que proporcione uma redução da complexidade total. Esse é o objectivo da representação global unificada, que se baseia nos seguintes pontos:

- As interacções envolvem uma emissão e a respectiva recepção, que podem ser representadas numa única actividade de emissão-recepção.
- Apesar dos fluxos de controle existentes nos vários participantes poderem ser diferentes, eles devem partilhar padrões comuns para que o processo global seja isento de erros. Por exemplo, um envio de três mensagens diferentes em sequência, requer a recepção de três mensagens, que pode ser em paralelo, em sequência, ou por uma ordem qualquer. Contudo a utilização da sequência como ordem de recepção será o modo mais natural.

Juntando as interacções emissão-recepção num fluxo de controle único e global, que envolva todos os participantes, teremos a descrição do processo interorganizacional descrito de forma global unificada. Nas Figura 2 servem como exemplos de processos interorganizacionais, envolvendo três e quatro participantes, que do lado esquerdo são descritos de forma discreta, ou seja, cada participante tem o seu processo, e do lado direito são descritos pelo processo total na forma global unificada.

No exemplos das figuras mencionadas, utiliza-se a notação gráfica das Redes de Petri [14] para descrever os processos. Onde cada círculo representa um lugar, que é um contentor de unidades, e cada quadrado/rectângulo representa uma transição que ao disparar: retira uma unidade de cada lugar de entrada; deposita uma unidade em cada lugar de saída; e executa uma acção associada (se houver).

A decomposição do processo descrito de forma global unificada nos processos dos vários participantes tem os seguintes passos: 1) replicar o processo global para cada participante; 2) no processo de cada participante fazer: a) substituir todas as actividades em que o próprio não participa por actividades inertes, ou seja que não têm quaisquer efeitos; b) substituir todas as actividades de interacção por actividades de emissão ou recepção, consoante o seu papel; c) substituir as primitivas de decisão, no fluxo de controle, em que o próprio não seja o participante activo na tomada de decisão, por decisões tardias em função das mensagens que são recebidas; d) eliminar as actividades inertes e simplificar os fluxos de controle.

2.2 Suporte à forma global unificada

A linguagem *BPSS/ebXML* permite a descrição de processos que envolvem vários parceiros, por intermédio de colaborações com múltiplas partes ("*multiparty collaborations*") mas que resultam da junção de várias colaborações binárias ("*binary collaborations*") por intermédio de transições. Cada colaboração binária possui a descrição da interacção entre dois participantes. Utiliza, portanto, parcialmente a forma global unificada. Esta linguagem também apresenta as desvantagens de ser mais complexa e requerer uma base de implementação mais complexa que as linguagens anteriormente apresentadas. A linguagem *WS-CDL* utiliza directamente a forma global unificada, contudo encontra-se em fase de desenvolvimento ("*draft*") e apresenta uma série de conceitos pouco claros, tais como o alinhamento de estado, e a utilização de canais ("*channels*").

A linguagem BPEL (*Business Process Execution Language*) foi escolhida para servir de base à nova, agora proposta, linguagem de descrição de processos interorganizacionais especificados de forma global unificada, porque se encontra estável, possui uma sustentação teórica clara, que é baseada nas redes de Petri, e porque já existiam, à altura do início deste trabalho, implementações de executores de processos BPEL.

3 A linguagem Common BPEL (CBPEL)

Nesta secção é apresentada a linguagem de descrição de processos interorganizacionais, baseada na BPEL, que designamos de *Common Business Process Execution Language*, abreviadamente CBPEL. A descrição da CBPEL encontra-se organizada em dois grupos: dados e parceiros; e interacções e controle de fluxo.

3.1 Primeiro grupo: dados e parceiros

Neste grupo são abordados os elementos: *partnerLinkType*, *partnerLink*, *partners*, *variables*, *correlation sets*, *process* e *assign*.

3.1.1 PartnerLinkType e PartnerLinks

Na BPEL os *partnerLinkTypes* (PLT) são declarações ao nível do WSDL, que identificam uma ligação entre eventualmente dois intervenientes, em que cada interveniente assume um papel ("*Role*") que é desempenhado por um *portType*. Os *partnerLinks* definem, no processo, para cada PLT, quem implementa cada uma das suas *roles*, indicando com *myRole* o papel do próprio processo, e com *partnerRole* o papel do outro parceiro.

Na CBPEL: o elemento PLT permanece igual, devendo contudo definir sempre dois papéis; nos *partnerLinks* dado que a vista do processo é global e única para todos os participantes, a noção de *myRole* será substituída por outro elemento *partnerRole*.

Estrutura do elemento *partnerLinks* na CBPEL:

```
<partnerLinks>?
  <partnerLink name="ncname" partnerLinkType="qname" >+
      <role name="ncname" partnerLinkTypeRoleName=”ncname”/>
      <role name="ncname" partnerLinkTypeRoleName=”ncname”/>
  </partnerLink>  </partnerLinks>
```

3.1.2 Partners

A definição de *partners*, num processo BPEL, identifica quem são os outros parceiros do próprio processo, e quais os *partnerlinks* em que eles participam.

A definição de parceiros (“*partners*”) é fundamental na CBPEL, pois vai definir quais os papeis globais existentes e para cada um deles qual a sua participação em cada *partnerLink*. Cada *partner*, como será descrito de forma imparcial, e terá que possuir uma associação a cada *partnerlink-role* em que participa.

Estrutura do elemento “*partners*” na CBPEL fica então:

```
<partners>
  <partner name="ncname">+
    <partnerLink name="qname" role=”ncname”/>+
  </partner>  </partners>
```

3.1.3 Variable(s), Assign, copy e CorrelationSets

Na BPEL: o elemento *variable*, permite definir variáveis; os elementos *assign* e *copy* permitem realizar afectações e transformações de dados; e os conjuntos de correlação (“*correlation sets”*) permitem definir um conjunto de propriedades (“*properties*”) como um identificador de correlação. As *properties* são referências para campos de dados existentes nas mensagens que têm que ser conhecidos pelo processo. Este tipo de informação é normalmente designado de dados do protocolo (“*protocol data*”). Todos estes elementos permanecem inalterados na CBPEL.

3.1.4 Process

O elemento *process* é o elemento que contém as definições de um processo na linguagem BPEL. Na CBPEL o nome deste elemento será alterado para commonProcess, para elucidar a diferença de domínio.

Estrutura deste elemento na CBPEL:

```
<commonProcess name="ncname" . . .
  xmlns="http://wwww.temp.org/cbpel/2004/09/common-business-
process/">
      <partnerLinks/>?       <partners/>?       <variables/>?
      <correlationSets/>?    <faultHandlers/>?
      <compensationHandler/>? <eventHandlers/>?
   activity
</commonprocess>
```

3.2 Segundo grupo: interacções e controle de fluxo

Neste grupo são abordados os seguintes elementos: interacções (*invoke*, *receive*, *reply*), *wait*, *empty*, *sequence*, *switch*, *while*, *pick*, e *flow*.

3.2.1 Interacções

Para modelar interacções, na BPEL, existem os elementos: *receive*, *reply* e *invoke*. O elemento *receive* permite receber uma mensagem de um parceiro. O *reply* permite responder a uma chamada síncrona de um parceiro, do qual já se fez um *receive*. O *invoke* permite fazer o envio de uma mensagem e opcionalmente de forma síncrona, esperar por uma resposta.

A CBPEL, à semelhança com a *WS-CDL*, apresenta uma vista global comum sobre o processo, pelo que cada interacção é modelada por um único elemento, designado de "*send*", e que representa um i*nvoke* num parceiro e um *receive noutro* parceiro.

Estrutura do elemento *send* na CBPEL:

```
<send partnerLink="ncname"
      fromPartner="ncname"   toPartner="ncname"
      fromPortType="qname"   toPortType="qname"
      fromOperation="ncname" toOperation="ncname"
      inputVariable="ncname" outputVariable="ncname"
      createInstance="yes|no"? <!-- receiver -->
      syncSendName="ncname"?   <!-- receiver -->
      standard-attributes> standard-elements
      <correlations>?
</send>
```

3.2.2 Wait e Empty

Na BPEL, o elemento *wait* permite realizar uma pausa na execução, e o elemento *empty* permite realizar a acção vazia. Estes elementos deverão ter na CBPEL a indicação de quem cumpre essa acção.

Estrutura dos elementos *wait* e e*mpty* na CBPEL:

```
<wait (for="duration-expr"|until="deadline-expr")
      partner="ncname" standard-attributes>
      standard-elements
</wait>
<empty partner="ncname" standard-attributes>
      standard-elements
</empty>
```

3.2.3 Sequence

O elemento *sequence*, na BPEL, define que a execução das actividades englobadas são realizadas em sequência. Este elemento permanece igual na CBPEL.

3.2.4 Flow

O elemento *flow*, na BPEL, permite definir actividades que serão executadas concorrentemente. Podem ser modeladas dependências, entre essas actividades concorrentes, pelo elemento "*link*". Este elemento permanece igual na CBPEL. Na Figura 1 encontra-se um exemplo de um fluxo paralelo envolvendo vários participantes, e sua modelação na CBPEL. Na Tabela 1 encontra-se as descrições dos processos BPEL e CBPEL do exemplo da Figura 1.

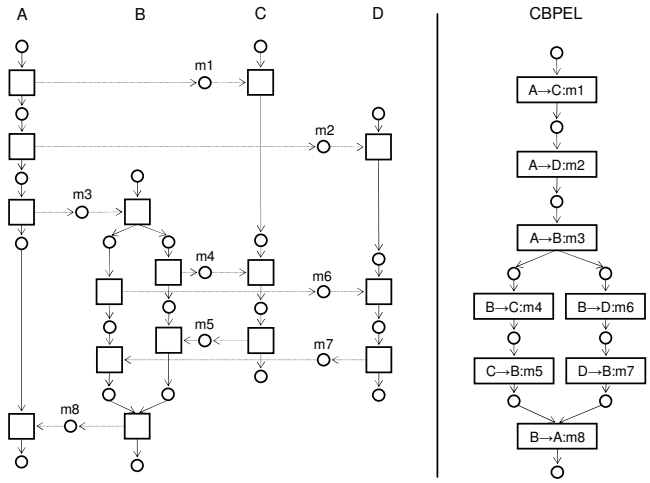


Figura 1 – Exemplo de um fluxo paralelo (*flow*) na CBPEL

Tabela 1 – Listagens BPEL e CBPEL do exemplo da Figura 1

Participante A

```
SEQ
  Send m1/c
  Send m2/d
  Send m3/b
  Rec m8/b
```

Participante B

```
SEQ
  Rec m3/a
  Flow
   Seq
     Send m4/c
     Rec m5/c
   Seq
     Send m6/d
     Rec m7/d
     Send m8/d
```

Participante C

```
Seq
  Rec m1/a
  Rec m4/b
  Send m5/b
```

Participante D

```
Seq
  Rec m2/a
  Rec m6/b
  Send m7/b
```

CBPEL

```
Seq
  a → c: m1
  a → d: m2
  a → c: m3
  Flow
   Seq
     b → c: m4
     c → b: m5
   Seq
     b → d: m6
     d → b: m7
     b → a: m8
```

3.2.5 Switch

Este elemento, na BPEL, permite realizar uma decisão de escolha múltipla. Numa definição global unificada de um processo, uma escolha múltipla, terá que ser executada por um participante, o qual deverá iniciar a primeira interacção de cada um dos ramos alternativos. Os outros participantes que figuram dentro da escolha múltipla terão que efectuar uma decisão tardia (*pick*) para saberem que ramo alternativo foi escolhido. Na Figura 2 encontra-se um exemplo de um fluxo contendo uma decisão, modelado em processos BPEL, e o respectivo processo CBPEL. As listagens deste exemplo foram omitidas por limitações de espaço.

3.2.6 While

Este elemento, na BPEL, permite a execução cíclica de uma actividade ou de um grupo de actividades. Este elemento tem um comportamento semelhante ao "*switch*", pois

inclui uma decisão. Na Figura 3 encontra-se um exemplo de um fluxo interorganizacional descrito graficamente na BPEL e na CBPEL, e que contém um ciclo. Na Tabela 2 encontra-se as listagens correspondentes a esse exemplo.

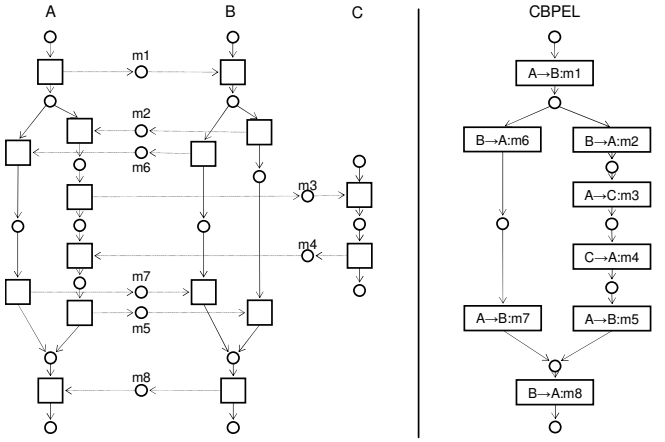


Figura 2 – Exemplo de uma decisão (*switch*) na CBPEL

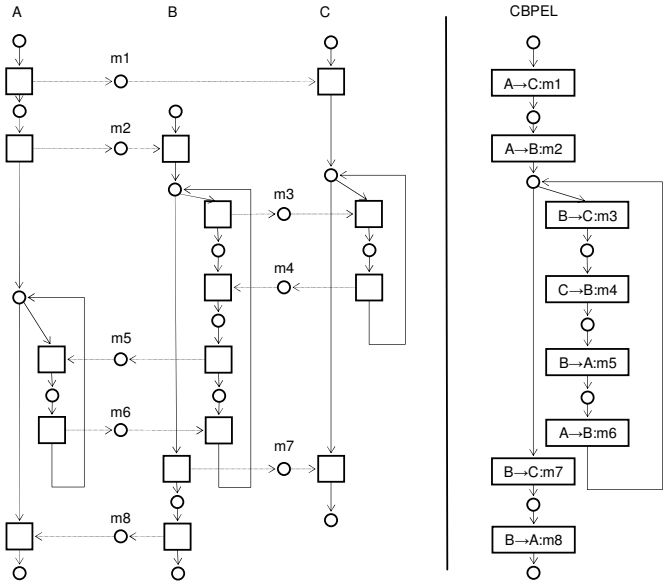


Figura 3 – Exemplo de um ciclo (*while*) na CBPEL

O exemplo da Figura 3 descreve um ciclo envolvendo três participantes. Da sua observação verifica-se que qualquer participante que intervenha dentro de um ciclo terá que também ele implementar esse ciclo, pois eventualmente não saberá quantas iterações irão existir. Contudo se o fluxo do participante é iniciado dentro do ciclo, então cada iteração do ciclo gera uma nova instância do fluxo desse participante.

Tabela 2 – Listagens BPEL e CBPEL do exemplo da Figura 3

Participante A	Participante B	Participante C	CBPEL
SEQ Send m1/c Send m2/b Sair = 0 while (!sair) Pick Rec m5/b Send m6/b Rec m8/b Sair = 1	SEQ Rec m2/a While Seq Send m3/c Rec m4/c Send m5/a Rec m6/a Send m7/c Send m8/a	Seq Rec m1/a While (!sair) Pick Rec m3/b Send m4/b Rec m7/b Sair = 1	Seq a → c: m1 a → b: m2 While Seq b → c: m3 c → b: m4 b → a: m5 a → b: m6 b → c: m7 b → a: m8

3.2.7 Pick

Este elemento, na BPEL, permite seleccionar a execução de actividades em função do primeiro acontecimento a ocorrer de entre uma série de mensagens (*onMessage*) e de uma série de alarmes temporais (*onAlarm*) que são vulgarmente designados de *timeouts*. O elemento *pick* em si não faz sentido existir na CBPEL pois é um elemento passivo. No entanto a capacidade de modelar *timeouts* deve existir na CBPEL. Assim o elemento *onAlarm* é adicionado ao elemento *send,* permitindo a indicação de *timeouts* na recepção de mensagens.

Estrutura do elemento *onAlarm* na CBPEL fica então:

```
<onAlarm (for="duration-expr" | until="deadline-expr")>*
       activity
</onAlarm>
```

4 Exemplo

Como exemplo, de um processo interorganizacional, é apresentado um cenário referente à realização de uma compra numa livraria electrónica, o qual é um cenário adaptado de [1]. O cenário contém quatro entidades participantes: o cliente (c), a livraria (b), o editor (p), e o serviço de entregas (s). Na Figura 4 apresenta-se o cenário descrito na sua forma global unificada, cujo descrição sumária em CBPEL é a seguinte:

```
<commonprocess name="Livraria Electrónica"
  xmlns="http://tempuri.org/cbpel/2004/09/common-business-process/">
 <partners>
   <partner name="Customer"/>       <partner name="Bookstore"/>
   <partner name="Publisher"/>      <partner name="Shipper"/>
 </partners>
 <sequence>
   <send fromPartner="Customer" toPartner="Bookstore"
         Operation="c_order" />
   <send fromPartner="Bookstore" toPartner="Publisher"
         Operation="b_order" />
   <switch>
     <case v1>
       <sequence>
```

```
        <send fromPartner="Publisher" toPartner="Bookstore"
              Operation="pb_rejected" />
        <send fromPartner="Bookstore" toPartner="Customer"
              Operation="bb_rejected" />
      </sequence>   </case>
    <case v2>
      <sequence> // notação ainda mais resumida
        <send "Publisher" to "Bookstore" Op "pb_accepted" />
        <send "Bookstore" to "Customer"  Op "bb_accepted" />
        <send "Bookstore" to "Shipper"   Op "req_shipment" />
        <switch>
          <case v1>
            <sequence>
              <send "Shipper" to "Bookstore"   Op "sb_rejected" />
              <send "Bookstore" to "publisher" Op "bp_rejected" />
              <send "Bookstore" to "Customer"  Op "bb2_rejected" />
            </sequence>   </case>
          <case v2>
            <sequence>
              <send "Shipper"   to "Bookstore" Op "sb_accepted" />
              <send "Bookstore" to "Publisher" Op "inform_pub" />
              <send "Publisher" to "Shipper"   Op "send_book" />
              <send "Shipper"   to "Customer"  Op "ship_book" />
              <send "Shipper"   to "Bookstore" Op "notify" />
              <send "Bookstore" to "Customer"  Op "send_bill" />
              <send "Customer"  to "Bookstore" Op "payment" />
            </sequence>  </case>  </switch>  </sequence>  </case>
                              </switch>  </sequence>  </commonprocess>
```

Seguidamente apresenta-se o conteúdo sumário do processo da livraria, descrito em BPEL, e extraído do processo global CBPEL anterior:

```
<process name="Bookstore-process" . . .
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
 <sequence>
   <receive partner="Customer" operation="c_order" />
   <invoke partner="Publisher" operation="b_order" />
   <pick>
     <onMessage frompartner="Publisher" operation="pb_rejected" />
         <invoke partner="Customer" operation="bb_rejected" />
     </onMessage>
     <onMessage frompartner="Publisher" operation="pb_accepted" />
       <sequence>
         <invoke partner="Customer" operation="bb_accepted" />
         <invoke partner="Shipper" operation="req_shipment" />
         <pick>
           <onMessage frompartner="Shipper" operation="sb_rejected" />
             <sequence>
                <invoke partner="publisher" operation="bp_rejected" />
                <invoke partner="Customer" operation="bb2_rejected" />
             </sequence>
           </onMessage>
           <onMessage frompartner="Shipper" operation="sb_accepted" />
             <sequence>
               <invoke partner="Publisher" operation="inform_pub" />
               <receive partner="Shipper" operation="notify" />
               <invoke partner="Customer" operation="send_bill" />
               <receive partner="Customer" operation="payment" />
             </sequence>  </onMessage>  </pick>  </sequence>
                         </onMessage>  </pick> </sequence> </process>
```

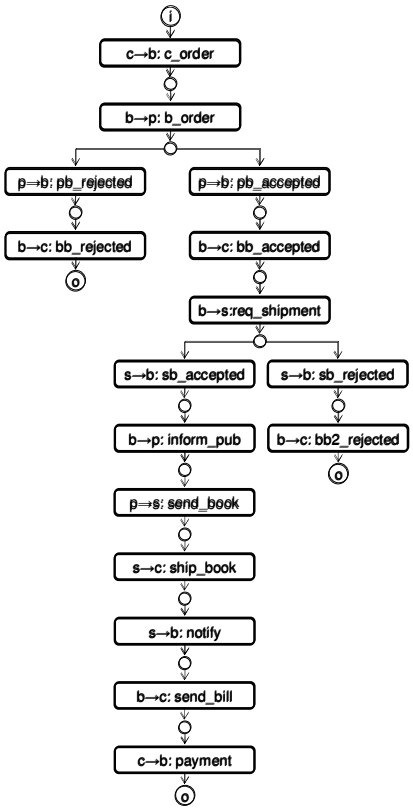


Figura 4 – Cenário interorganizacional da livraria electrónica

5 Considerações finais

Este trabalho propõe uma linguagem designada de *Common Business Process Execution Language*, ou abreviadamente CBPEL, que visa a descrição de processos de negócio interorganizacionais, e que é baseada na linguagem BPEL. A CBPEL descreve os processos interorganizacionais de uma forma global, que é unificadora das perspectivas dos seus vários participantes. O objectivo desta forma de descrição dos processos é diminuir a sua complexidade, que de outro modo seriam descritos pela conjunção dos processos individuais de todos os participantes.

No entanto, esta forma de descrever os processos interorganizacionais apresenta dois problemas: fraca noção da responsabilidade dos processos individuais; e limitação na definição dos fluxos. A fraca noção da responsabilidade dos processos individuais deve-se a que os processos agora estão todos juntos num único fluxo, contudo a ferramenta de edição pode apresentar de forma paralela com o processo global, o processo individual dos participantes de modo a se poder visualizar as responsabilidades individuais. A limitação dos fluxos dos participantes é o resultado da

descrição por fluxo único, contudo os participantes são livres de implementar o fluxo acordado do modo que entendam desde que preservem as suas características [1].

Como trabalho futuro são identificados os seguintes aspectos: (1) demonstrar que características a forma global unificada deve possuir para que os processos gerados, a partir de um processo global bem construído, sejam processos bem construídos; (2) incorporar o tratamento de eventos, falhas e compensações; e (3) incorporar um protocolo de coordenação distribuída como o protocolo *WS-BusinessActivity* [9] para garantir a coerente execução entre os vários participantes.

Referências

1. W. van der Aalst, M. Weske, The P2P Approach to Interorganizational Workflows, In K.R. Dittrich, A. Geppert, M. C. Norrie (eds), Proc. Of the 13th Int. Conf. on Advanced Information Systems(CAiSE'01), Berlin, 2001
2. T. Andrews, et al, Business Process Execution Language for Web Services, Version 1.0, May 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
3. A. Arkin, Business Process Modeling Language (BPML), Working Draft 0.4, BPMI, March 2001, http://www.bpmi.org/
4. A. Arkin, et al, Web Services Choreography Interface (WSCI), version 1.0, W3C Note, August, 2002, http://www.w3.org/TR/wsci/
5. B. Atkinson, et al, Web Services Security (WS-Security), Microsoft IBM VeriSign, April, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
6. R. Bilorusets, et al, Web Services Reliable Messaging Protocol (WS-ReliableMessaging), March, 2004,
7. D. Box, et al, Simple Object Access Protocol, version 1.1, May, 2000, http://www.w3.org/TR/SOAP
8. T. Bray, et al, Extensible Markup Language (XML) 1.0, W3C, February, 1998, http://www.w3.org/TR/REC-xml
9. L. Cabrera, et al, Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity, Microsoft Corporation, Jan, 2004, http://msdn.microsoft.com/library/en-us/dnwebsrv/html/ wsacoord.asp
10. D.Chappel, et al, Professional ebXML Foundations, Wrox Press Ltd, UK, 2001 (http://www.ebxml.org)
11. E. Christensen, et al, Web Services Description Language (WSDL), version 1.1, W3C Note, March, 2001, http://www.w3.org/TR/wsdl.html
12. N. Kavantzas, D. Burdett, G, Ritzinger, Web Services Choreography Description Language (WSCDL), version 1.0, April, 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/
13. H. Kreger, Web Services Conceptual Architecture (WSCA 1.0), IBM Software Group
14. J. Peterson, Petri net theory and the modeling of systems, Prentice Hall, 1981
15. S. Thatte: XLANG - Web Services for Business Process Design, Draft Specification. Microsoft, May 2001,http://www.gotdotnet.com/team/xml_wsspecs/xlangc/default.htm
16. UDDI Group, Universal Description, Discovery and Integration, UDDI Technical White Paper, Ariba Microsoft IBM, Sep, 2000
17. WfMC, Workflow Management Coalition: Terminology & Glossary, UK, Feb, 1999

A gestão de obras digitalizadas na BND

José Luís Borbinha, Gilberto Pedrosa, João Penas, João Gil

National Library of Portugal
Jose.Borbinha@bn.pt; gfsp@ext.bn.pt; penas@ext.bn.pt; jgil@ext.bn.pt

Palavras chave: *XML, METS, XHTML, Digitalização, Bibliotecas Digitais, Preservação Digital.*

Resumo. Este artigo aborda o problema da gestão de obras digitalizadas na iniciativa Biblioteca Nacional Digital. O desenvolvimento de colecções de obras digitais e digitalizadas levanta problemas especiais ao nível da gestão de cópias digitalizadas de originais manuscritos ou impressos, que se pretendem abordar de forma a permitir flexibilidade, reutilização e longevidade. Para responder a este problema a BN decidiu adoptar um esquema XML designado de METS, tendo desenvolvido para o efeito um conjunto de ferramentas adequadas aos principais casos de uso associados, sendo aqui descritas duas delas, denominadas de PAPAIA e ContentE, assim como a interoperabilidade entre ambas.

Introdução

Este artigo aborda o problema da gestão de obras digitalizadas na BN – Biblioteca Nacional, tal como está sendo abordado na perspectiva da iniciativa BND – Biblioteca Nacional Digital [1]. No contexto da BND tem vindo a ser desenvolvido um trabalho de digitalização de manuscritos, obras impressas e de outras obras e materiais em suporte físico. Além disso tem-se procurado desenvolver uma política de depósito de cópias digitais de obras relevantes para a missão da BN de biblioteca patrimonial, compreendendo tanto as obras nascidas e criadas para publicação digital como as cópias digitais de obras destinadas a ser impressas.

O desenvolvimento de colecções de obras digitais e digitalizadas levanta problemas especiais ao nível dos processos da descrição e catalogação, os quais não são neste momento abordados. Na BND seguem-se para este fim as regras e procedimentos normais definidos para a BN e recomendados pela PORBASE [6], os quais se têm mostrado suficientes no geral. Os novos desafios aqui abordados localizam-se a montante desses problemas, nos casos em que estamos perante a criação de cópias digitalizadas de originais manuscritos ou impressos.

De um modo simples o problema será o de como organizar as imagens e os respectivos metadados que são produzidas num projecto de digitalização, isso de forma estruturada e que permita a reutilização fácil e flexível de toda esses conteúdos em qualquer contexto (tal como a criação de cópias em XHTML).

METS

A manutenção de uma biblioteca de objectos digitais exige a manutenção dos metadados sobre esses objectos. Os metadados necessários para utilizar e gerir com sucesso objectos digitais são diferentes e mais vastos que os metadados utilizados para gerir colecções de obras impressas e outros materiais físicos. Embora uma biblioteca possa manter metadados descritivos sobre um livro da sua colecção, o livro não se dissolverá numa série de páginas soltas caso a biblioteca não registar metadados estruturais sobre a organização do livro, nem os investigadores serão incapazes de avaliar o valor do livro se a biblioteca não anotar que o livro foi produzido numa dada imprensa.

O mesmo não pode ser dito para uma versão digitalizada do mesmo livro. Sem metadados estruturais, os ficheiros com imagens ou texto serão de pouca utilidade, e sem metadados técnicos sobre o processo de digitalização, os investigadores poderão ter dúvidas sobre a exactidão da reflexão do original que a versão digital oferece. Por questões de gestão interna, a biblioteca deve ter ainda acesso a metadados técnicos apropriados para lhe permitir refrescar e migrar os dados, garantindo a durabilidade dos recursos.

Fig. 1. Exemplo da estrutura de uma obra em METS (criado pela aplicação ContentE).

Para responder a este problema a BN decidiu adoptar um esquema XML designado de METS – Metadata Encoding and Transmission Standard [4], de que se apresenta um exemplo de aplicação na Figura 1. Para geração e processamento desse formato de metadados estruturais foi desenvolvido um conjunto de ferramentas adequadas aos principais casos de uso associados, conforme adiante se descreve.

O formato METS foi definido pela DLF [2], tendo sete secções principais:

- **Cabeçalho**: O cabeçalho contém metadados descrevendo o documento METS em si, incluindo informação como o criador, editor, etc.
- **Metadados Descritivos**: Esta secção pode referenciar metadados externos ao documento METS (como um registo UNIMARC ou EAD acessíveis num servidor na Internet), conter metadados embebidos, ou ambos. Múltiplas instâncias de metadados, internas ou externas, podem ser incluídos nesta secção.
- **Metadados Administrativos**: Esta secção oferece informação sobre como os ficheiros foram criados e armazenados, direitos de propriedade intelectual, metadados sobre o objecto original a partir do qual o objecto digital foi derivado, etc. Tal como os metadados descritivos, os metadados administrativos podem ser tanto externos ao documento METS como codificados internamente.
- **Secção de Ficheiros**: Esta secção lista todos os ficheiros que compõem o objecto. Elementos <file> podem ser agrupados em elementos de grupos de ficheiros <fileGrp>, para permitir a subdivisão de ficheiros por versão do objecto.
- **Mapa Estrutural**: Este mapa é o coração do documento METS. Ele esboça uma estrutura hierárquica para o objecto, e liga os elementos dessa estrutura a ficheiros com conteúdos e metadados referentes a cada elemento.
- **Ligações Estruturais**: Esta secção permite registar referências entre nós na hierarquia esboçada no Mapa Estrutural. Esta secção tem um valor particular na utilização do METS para arquivar sítios da Internet.
- **Comportamento**: Estas secções podem ser usadas para associar comportamentos para os conteúdos dos objectos METS. Cada comportamento é composto de uma definição abstracta e ainda de uma referência para o módulo de código executável que o concretiza.

O esquema METS oferece uma norma útil para a troca de objectos digitais entre repositórios. Adicionalmente, o METS oferece a possibilidade de associar um objecto digital com comportamentos ou serviços. A discussão anterior descreve o esquema, mas uma examinação mais detalhada da sua documentação é necessária para compreender todo o alcance das suas capacidades.

No contexto do modelo de referência do OAIS – Open Archival Information System [5], e dependendo da sua utilização, um documento METS pode ser usado como Pacote de Informação de Submissão (SIP), um Pacote de Informação de Arquivo (AIP) ou um Pacote de Informação de Disseminação (DIP).

Todas as obras depositadas na BND são assim estruturadas segundo o esquema METS. As obras digitais criadas no exterior são sujeitas processos de transformação, mas já as obras digitalizadas criadas na BN (ou noutros locais, mas eventualmente utilizando a tecnologia disponibilizada para o efeito pela BN, acessível livremente), são estruturadas tendo logo de raiz o METS como formato nativo. Uma das ferramentas que a BN desenvolveu para esse efeito (estruturar obras digitalizadas em METS) é a aplicação ContentE, a qual pode ser utilizada de raiz, ou em complemento

a uma outra denominada de PAPAIA. Na Figura 1 apresenta-se um exemplo de um ficheiro criado dessa forma.

PAPAIA

A aplicação PAPAIA tem por objectivo o tratamento inicial das obras logo após a sua digitalização. As suas funcionalidades são essencialmente as seguintes:

- **Renomeação:** Renomeação automática de ficheiros de imagem de acordo com normas e parâmetros configuráveis, tais como cota, número de sequência, etc.
- **Edição de metadados TIFF:** Edição dos cabeçalhos TIFF [7] das imagens.
- **Estruturação:** Associação das imagens à descrição estrutural de uma obra.

O PAPAIA tem uma interface gráfica análoga à da navegação de directórios de um sistema de ficheiros. Além da estruturação, é possível associar a cada imagem uma ou mais palavras-chave, o que pode ser usado para criar índices de navegação. Toda esta informação é registada num ficheiro XML, como apresentado na Figura 2 (ficheiro este que pode ser reutilizado pelo ContentE).

São registados três tipos de elementos nesta descrição:

- **level** - nível ou nó na hierarquia estrutural de uma imagem ou agrupamento;
- **image** - associação de uma imagem a um nível da árvore;
- **keyword** - palavra-chave atribuída a uma determinada imagem.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <work xmlns="http://www.bn.pt/PapaiaSchema/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
  xsi:schemaLocation="http://www.bn.pt/PapaiaSchema/ http://teresa.bn.pt/schemas/Papaia">
  - <application>
      <builder app="PAPAIA" date="2004:11:26:02:37:24" version="1.3" />
    </application>
  - <root title="Root">
    - <level title="Introducao">
      - <level title="L-64003-P_0007_1_t0.tif">
          <image src="L-64003-P_0007_1_t0.tif" alt="" />
        </level>
      - <level title="L-64003-P_0008_2_t0.tif">
          <image src="L-64003-P_0008_2_t0.tif" alt="" />
        </level>
      - <level title="L-64003-P_0009_3_t0.tif">
        - <image src="L-64003-P_0009_3_t0.tif" alt="">
            <keyword>D. Dinis</keyword>
          </image>
        </level>
      - <level title="L-64003-P_0010_4_t0.tif">
          <image src="L-64003-P_0010_4_t0.tif" alt="" />
        </level>
      - <level title="L-64003-P_0011_5_t0.tif">
          <image src="L-64003-P_0011_5_t0.tif" alt="" />
        </level>
      </level>
    - <level title="Capitulo 1">
      - <level title="L-64003-P_0012_6_t0.tif">
          <image src="L-64003-P_0012_6_t0.tif" alt="" />
        </level>
      - <level title="L-64003-P_0013_7_t0.tif">
          <image src="L-64003-P_0013_7_t0.tif" alt="" />
        </level>
```

Fig. 2. Exemplo da estrutura descritiva de uma obra exportada pela aplicação PAPAIA.

O elemento "keyword" pode incluir as coordenadas do rectângulo envolvente que localiza a palavra na imagem respectiva, o que pode expandir as opções de pesquisa e apresentação da obra (essa informação pode ser gerada por outra aplicação também desenvolvida na BND, denominada de KIWI).

ContentE

A aplicação ContentE tem como objectivo suportar o processo de construção de estruturas formais de obras digitalizadas. Um projecto é constituído por um ficheiro XML de abertura e uma directoria "master", na qual são inseridos os diferentes ficheiros que compõem a obra, possivelmente em múltiplos tipos MIME (TIFF, GIF, JPEG, ASCII, PDF, etc.). Podem-se importar descrições de estruturas das obras do PAPAIA ou criar novas descrições de raiz, e ainda importar outros metadados.

É possível assim a partir da cópia "master" gerar várias cópias da obra em formato XHTML, podendo estas apresentar diferentes estruturas e formas de visualizações. É também possível criar múltiplos índices de visualização de uma obra, segundo o tipo de exploração e acesso que se pretender oferecer. Pode-se exportar a obra em METS, sendo no entanto possível utilizar outros formatos equivalentes desde que devidamente definidos. Na Figura 3 apresenta-se a arquitectura da aplicação, com um exemplo da sua interface na Figura 4.

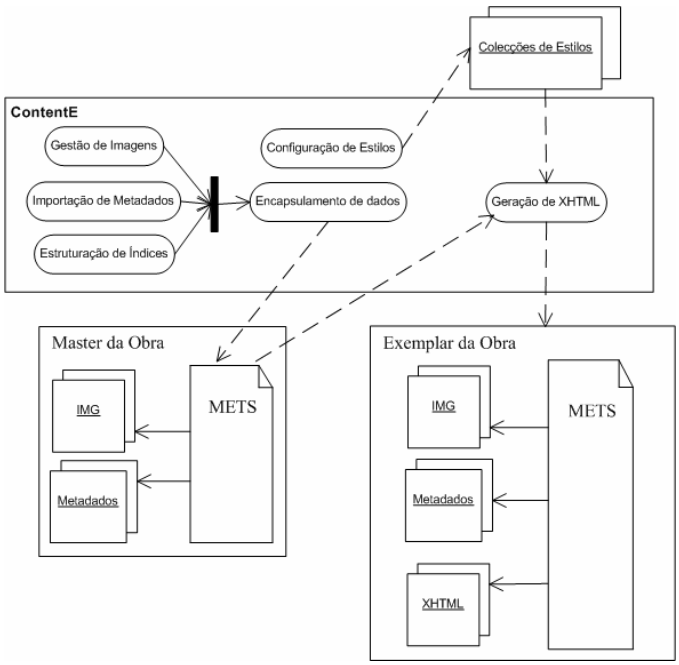


Fig. 3. Arquitectura da aplicação ContentE.

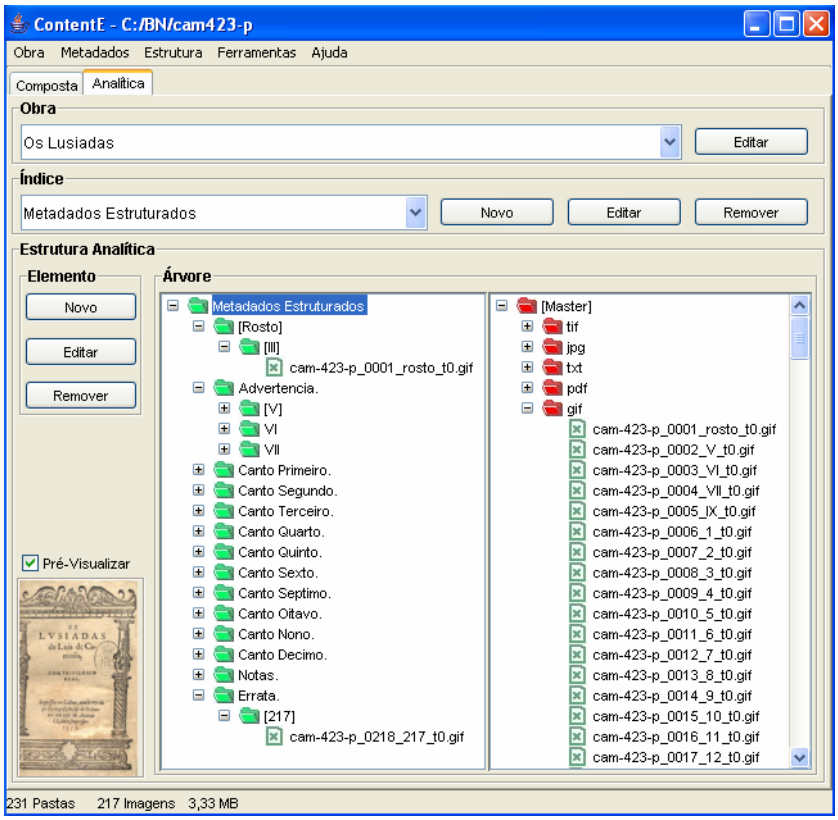


Fig. 4. Interface da aplicação ContentE.

Metadados

A cada obra digitalizada poderá ser associado um conjunto de diversos tipos de metadados. É possível associar diferentes metadados descritivos a diferentes nós. Os metadados importados poderão estar definidos em diferentes esquemas XML, sendo apenas necessário ter para cada caso o respectivo estilo XSL para a sua visualização. Exemplos desses metadados são metadados de direitos (a aplicar à obra no seu todo ou a partes específicas), técnicos (informação relativa ao processo de digitalização) ou descritivos (UNIMARC, Dublin Core, etc.).

No caso da BN, os metadados descritivos são importados em linha da PORBASE, através de um serviço web próprio, e associados à obra. Neste caso estes metadados são registados em MARCXML [3], como ilustrado na Figura 5.

Finalmente é possível registar ainda estruturas de metadados de termos e condições de uso das obras, os quais poderão ser aplicados a qualquer nó da estrutura (toda a obra, apenas um capítulo, apenas uma imagem, etc.). Neste momento a informação interpretada para este fim pela aplicação ContentE é registada no próprio METS, nos elementos definidos para o efeito, sendo no entanto possível registar

independentemente qualquer outra estrutura também (estas estruturas não são no entanto interpretadas, sendo utilizadas apenas para registo e visualização).

```
- <collection xsi:schemaLocation="http://www.bn.pt/standards/metadata/marcxml/1.0/
  http://xml.bn.pt/schemas/Unimarc-1.0.xsd">
  - <record>
      <leader>01463cam 2200397 450 </leader>
      <controlfield tag="001">323613</controlfield>
      <controlfield tag="005">20030117160300.0</controlfield>
    - <datafield ind1=" " ind2=" " tag="095">
        <subfield code="a">PTBN00339700</subfield>
      </datafield>
    - <datafield ind1=" " ind2=" " tag="100">
        <subfield code="a">19880426d1572 k y0pora0103 ba</subfield>
      </datafield>
    - <datafield ind1="0" ind2=" " tag="101">
        <subfield code="a">por</subfield>
      </datafield>
    - <datafield ind1=" " ind2=" " tag="102">
        <subfield code="a">PT</subfield>
        <subfield code="b">Lisboa</subfield>
      </datafield>
    - <datafield ind1="1" ind2=" " tag="200">
        <subfield code="a"><Os >Lusiadas</subfield>
        <subfield code="f">de Luis de Camões</subfield>
      </datafield>
    - <datafield ind1=" " ind2=" " tag="210">
        <subfield code="a">Lisboa</subfield>
        <subfield code="c">em casa de Antonio Gõçaluez</subfield>
        <subfield code="d">1572</subfield>
      </datafield>
    - <datafield ind1=" " ind2=" " tag="215">
```

Fig. 5. Exemplo de um registo bibliográfico em MARCXML

Criação de cópias de visualização

No ContentE é possível definir os parâmetros de apresentação da obra, que são guardados em ficheiros XML seguindo um esquema criado para o efeito. A partir destas predefinições é possível criar várias configurações de apresentação (colecções de estilos) que podem ser modificadas e reutilizadas posteriormente, e que se adaptam a cada tipo de obra a gerar. Assim, torna-se necessário escolher o estilo de apresentação da obra antes da sua geração.

Antes da geração do novo exemplar é também produzido um ficheiro METS respectivo. Este ficheiro irá incluir apenas os conteúdos escolhidos especificamente para o exemplar a ser gerado e será inserido no directório do exemplar respectivo.

O passo seguinte consiste na aplicação de um estilo definido em XSL a este ficheiro METS. Este processo é repetido para cada página da obra, o que resulta na geração de um ficheiro XHTML por página. Para além destas, é ainda produzido uma página XHTML com a capa da obra e com os metadados descritivos. Também aqui, poderão surgir diferentes visualizações da informação recorrendo ao uso diferentes folhas de estilo, como ilustrado na Figura 6 e na Figura 7.

Os exemplares assim gerados podem ser manuseados de forma independentes, podendo ser transportados e consultados em qualquer lado.

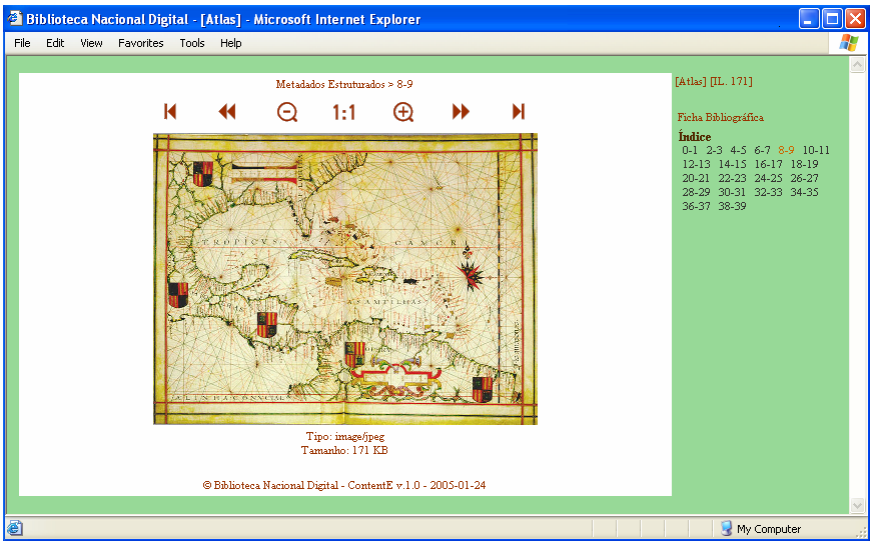


Fig. 6. Exemplo de uma obra publicada em formato XHTML

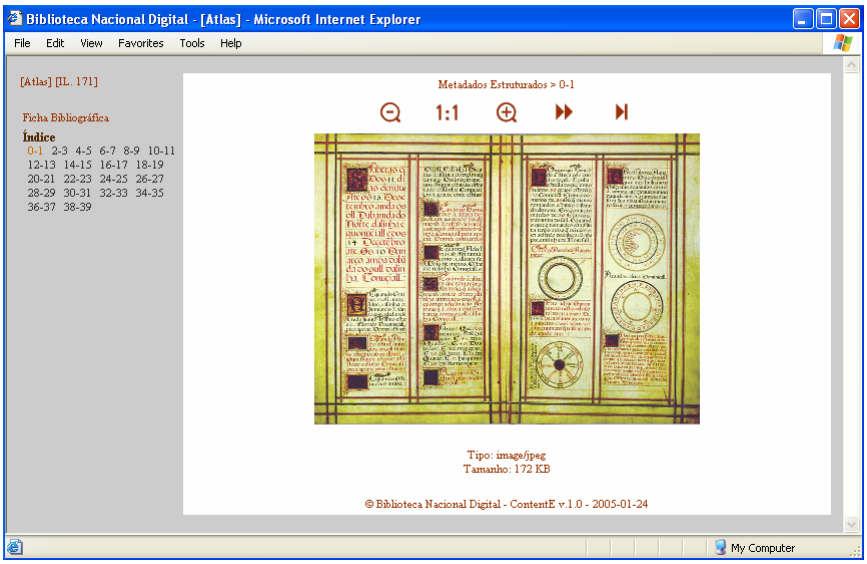


Fig. 7. Exemplo da mesma obra apresentada na Figura 6, mas agora com um outro estilo.

Conclusões

Os resultados aqui descritos correspondem a uma parte dos desenvolvimentos da segunda fase da iniciativa da Biblioteca Nacional Digital, que decorreu de meados de 2003 até ao final de 2004. Para além das provas de conceito terem demonstrado a validade dos modelos, foram ao mesmo tempo nesta fase desenvolvidas ferramentas e definidos processos igualmente válidos, suportando realmente os processos em curso.

Como trabalho futuro, fruto da aprendizagem entretanto adquirida e da reanálise dos processos de produção, foi já decidido integrar na aplicação Contente as funções da aplicação PAPAIA, visando obter um ambiente mais poderoso (as duas aplicações nasceram em contextos diferentes, tendo a sua convergência ocorrido naturalmente mas numa fase posterior às primeiras fases de análise).

Está planeado vir ainda a acrescentar ao ambiente ContentE a capacidade de importação e processamento de mais esquemas de metadados descritivos e estruturais, especialmente de descrições EAD [9] e TEI [10], assim como a capacidade de produção de local de cópias para acesso em PDF ou DAISY [11] (neste último caso associando ainda a capacidade de indexação de conteúdos sonoros).

Referências

1. BND – Biblioteca Nacional Digital [Em linha]. URL: <http://bnd.bn.pt>
2. DLF – Digital Library Federation [Em linha]. URL: http://www.diglib.org/dlfhomepage.htm
3. MARCXML – MARC21 XML Schema [Em linha] URL: http://www.loc.gov/standards/marcxml/
4. METS – Metadata Encoding and Transmission Standard [Em linha]. URL: http://www.loc.gov/standards/mets/
6. OAIS – Reference Model for an Open Archival Information System. [Em linha]. URL: http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html
7. PORBASE – Base Nacional de Dados Bibliográficos [Em linha]. URL: http://www.porbase.org
8. TIFF – Tag Image File Format reference v.6 [Em linha]. URL: http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf
9. EAD - Encoded Archival Description [Em linha] URL: http://www.loc.gov/ead/
10. TEI - Text Encoding Initiative [Em linha] URL: http://www.tei-c.org/
11. DAISY Consortium [Em linha] URL: http://www.daisy.org/

O protocolo MOS e o SIGDiC

Jorge Bertocchini[1],João Paulo Rodrigues[1]

INESC Porto, R. Dr. Roberto Frias, 4200-465 Porto
http://www.inescporto.pt/
jbertocchini@netcabo.pt,jpr@inescporto.pt

Resumo O MOS (Media Object Server Protocol) é um protocolo criado para comunicação entre aplicações de redacção - Newsroom Computer Systems (NCS) - e equipamentos de Broadcast - Media Object Servers (MOS)-, como sejam, servidores de vídeo ou áudio, arquivos multimédia, geradores de caracteres, entre outros equipamentos.
Pela sua importância actual em ambientes profissionais de broadcast, o protocolo MOS foi escolhido, não só para a integração do Sistema Integrado de Gestão e Difusão de Conteúdos (SIGDiC) com os restantes equipamentos profissionais, mas também para implementação de mecanismos de comunicação interna entre os vários módulos do SIGDiC.
A natureza distribuída do SIGDiC sobre uma rede MOS, implicava a necessidade de implementar elementos que terminariam o protocolo MOS em cada módulo, o que levou ao desenvolvimento de uma framework em Java.

1 Introdução

O trabalho aqui apresentado foi realizado no âmbito de um projecto financiado pelo PRIME - Sistema Integrado de Gestão e Difusão de Conteúdos (SIGDiC)[5] - cujo objectivo é desenvolver um sistema capaz de integrar diferentes fontes de conteúdos e disponibilizá-los, automaticamente ou semi-automaticamente, em diferentes canais de comunicação (SMS, Teletexto, WAP, Web, etc.).

Tendo em vista a definição de requisitos, o projecto tem como parceiro uma entidade líder em Portugal na área da produção, edição, gestão e difusão de conteúdos - a RTP. Contudo, o SIGDiC foi concebido tendo em mente as necessidades da generalidade das organizações que desenvolvem o seu "core-business"nesta área e, deste modo, são potenciais utilizadores dos resultados do projecto, todas as empresas nacionais e internacionais que produzem, editam e geram conteúdos para os sectores do audiovisual, Internet e telecomunicações.

O SIGDiC irá ter uma arquitectura modular, baseada em sistemas abertos, e está a ser desenvolvido recorrendo, tanto quanto possível, a ferramentas de domínio público. É neste contexto que surge a opção pelo MOS (Media Object Server Communications Protocol)[4], como tecnologia de middleware, pelo facto de ser um protocolo open-source e por, actualmente, constituir um standard de facto como protocolo de interligação entre equipamentos heterogéneos de múltiplos fabricantes.

O MOS é um protocolo emergente, construído para possibilitar uma comunicação fácil e simples dentro de sistemas de produção de notícias servindo, especialmente, para a interligação de sistemas de redacção - Newsroom Computer Systems (NCS) - e Media Object Servers, tais como servidores de vídeo e geradores de caracteres, entre outros.

Exemplos do que o MOS permite fazer:

1) Usando Media Servers e comunicação através de MOS, produtores em centros noticiosos podem gerir alinhamentos de programas com conteúdos multimédia em estações distantes.

2) Usando software MOS, é possível incluir conteúdos multimédia, simplesmente arrastando o apontador MOS para o corpo da notícia.

2 O Protocolo MOS

O protocolo MOS coordena a comunicação entre uma central NCS e uma série de servidores MOS especializados. Neste contexto, o protocolo abrange um modelo de objectos, onde o NCS possui referências para todos os conteúdos que estão incluídos nos Media Object Servers. À medida que um programa está a ser produzido e o seu alinhamento, que inclui todos os respectivos conteúdos, é preenchido, o NCS transfere as referências, como os alinhamentos, para cada MOS e recebe retorno acerca do status de cada peça.

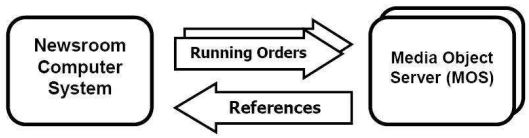


Figura 1. Interacção NCS - MOS (Tipos de Mensagens)

Dado que cada sistema gere a sua própria base de dados de objectos (conteúdos MOS, peças noticiosas, alinhamentos, elementos multimédia, entre outros), o protocolo troca referências e avisos entre as diferentes bases de dados quando um dos sistemas executa um evento (criação, destruição, agendamento ou controlo). O protocolo define um conjunto de notificações, que são enviadas numa ou noutra direcção durante a operação.

A versão 2.8 do protocolo MOS permite uma representação de metadata adicional (incluída no campo mosExternalMetadata), considerada como uma "caixa preta"dentro do protocolo MOS, ou seja, é guardada e trocada de uma forma transparente entre os intervenientes, mas a sua interpretação cai fora do âmbito do protocolo.

O MOS usa dois portos TCP/IP de forma bidireccional, onde as aplicações estão constantemente à escuta de mensagens. Estes portos, designados por MOS Upper Port(10541) e MOS Lower Port(10540), são utilizados para o envio de mensagens do NCS para o MOS e do MOS para o NCS, respectivamente.

Por exemplo, um comando de criação de um alinhamento ("running order") iniciado no NCS e o respectivo ACK, que será enviado pelo MOS, ocorrerão no MOS Upper Port, ao passo que comandos indicando actualizações de objectos localizados no MOS e o respectivo ACK enviado pelo NCS, tomarão lugar através do MOS Lower Port .

A partir versão 2.x, as messagens MOS são enviadas em XML[9], de acordo com as regras definidas no MOS Data Type Definition (DTD). Nas versões mais antigas da norma, 1.x, os campos de dados eram delimitados por um formato próprio.

No protocolo MOS todos os campos são case-sensitive e as mensagens deverão constituir documentos XML bem-formados, mas não é necessário que sejam válidos.

Cada mensagem MOS terá de ter uma raiz (<mos>), seguida dos identificadores do MOS e do NCS (¡mosID¿ e ¡ncsID¿), após o qual virá o tipo de mensagem. Os dados referentes a cada tipo de mensagem vêm em seguida, dentro de elementos XML.

```
<mos>
  <mosID>INESC.TICKER.MOS</mosID>
  <ncsID>LX1-ENPS</ncsID>
  <roStorySend>
     <roID>LX1-ENPS;P_TRAINING\W;56AE879D-DC9A-425C-8619C0324CA23C7B</roID>
     <storyID>LX1-ENPS;P_TRAINING\W\R_56AE879D-DC9A-425C-8619C( ... )D713B</storyID>
     <storySlug>Nova Story Insert</storySlug>
     <storyNum/>
     <storyBody>
        <p>teste de nova historia</p>
     </storyBody>
        <mosExternalMetadata>
        <mosScope>PLAYLIST</mosScope>
        <mosSchema>http://LX1-ENPS/schema/enps.dtd</mosSchema>
        <mosPayload>
           <Tema>Nacional</Tema>
           <Ticket>DIA DA MULHER - Jorge Sampaio condecorou 22 mulheres
                   portuguesas que se distinguiram nas artes e nas letras</Ticket>
           <Approved>0</Approved>
           <ModBy>JORGE</ModBy>
           <ModTime>20040422163917</ModTime>
           <Printed>20040308195450</Printed>
           <StoryProducer>ok</StoryProducer>
           <Owner>JORGE</Owner>
           <Creator>JORGE</Creator>
           <TextTime>10</TextTime>
           <MediaTime>140</MediaTime>
        </mosPayload>
     </mosExternalMetadata>
  </roStorySend>
</mos>
```

Figura 2. Mensagem MOS

O elemento referente à metadata é um dos mais versáteis e nele vão estar inseridas as informações definidas pelo utilizador, em função das necessidades para as quais a solução foi desenhada. Por exemplo, poderá incluir o corpo de

uma notícia, ao mesmo tempo que inclui informações acerca do método como essa notícia pode ser usada na página da Web e no portal WAP. Além disso, pode também conter informações quanto à forma como poderá ser exibida no teletexto, ou através de envio por SMS, desde o tipo de letra à cor utilizada. Não há limites ao tipo de informação que se pretenda colocar dentro desse(s) elementos(s), desde que sejam correctamente processados pelas aplicações que funcionam dentro dos serviços respectivos.

O conjunto de mensagens MOS, que foram divididos em perfis, definem um conjunto de funcionalidades suportadas. Dado que a implementação de todos os perfis não é obrigatória é comum, nas soluções comerciais, apenas uma parte dos perfis serem suportados.

Cada perfil implementa as seguintes funcionalidades:

- Perfil 0: Comunicações básicas
- Perfil 1: Fluxo básico orientado a objectos
- Perfil 2: Fluxo básico de conteúdos / alinhamentos
- Perfil 3: Fluxo avançado orientado a objectos
- Perfil 4: Fluxo avançado de conteúdos / alinhamentos
- Perfil 5: Controlo de elementos
- Perfil 6: Redireccionamento de MOS

Para uma empresa poder afirmar que o seu produto é compatível com MOS, o sistema deverá suportar, no mínimo o perfil 0 e um outro perfil adicional.

3 O SIGDiC e o MOS

O resultado final do projecto SIGDiC deve ser um sistema preparado para ser integrado facilmente numa arquitectura que promova a reutilização de conteúdos, bem como, a integração de processos de trabalho e fluxos de informação.

Baseado nesta premissa, e tendo em conta que o SIGDiC deverá estar preparado para ambientes profissionais que, cada vez mais, usam MOS para interligação entre equipamentos, o suporte para MOS passou a ser um requisito fundamental.

O MOS poderia ser utilizado como protocolo fronteira, permitindo a interligação do SIGDiC com os restantes sistemas proprietários do ambiente de redacção. Contudo, a opção tomada foi mais arrojada e consistiu em trazer o MOS para "dentro"do SIGDiC, utilizando-o como protocolo de interligação interno. Deste modo, a arquitectura SIGDiC assenta sobre uma rede MOS, em que cada elemento se apresenta como um nó dessa rede.

A opção pelo protocolo de comunicações MOS deve-se à enorme implantação que esta tecnologia tem no meio televisivo em particular e em todo o meio noticioso em geral. Actualmente, o MOS é utilizado em ambientes de redacção profissional como protocolo de comunicação entre sistemas heterogéneos, de múltiplos fabricantes. A opção por um protocolo de comunicação que se transformou num standard de facto na área de intervenção do projecto é, com certeza, a opção correcta, contrariamente à opção de definir e implementar um protocolo de comunicação específico e proprietário. Esta opção permite uma maior integração de sistemas, facilitando a sua utilização em ambientes profissionais.

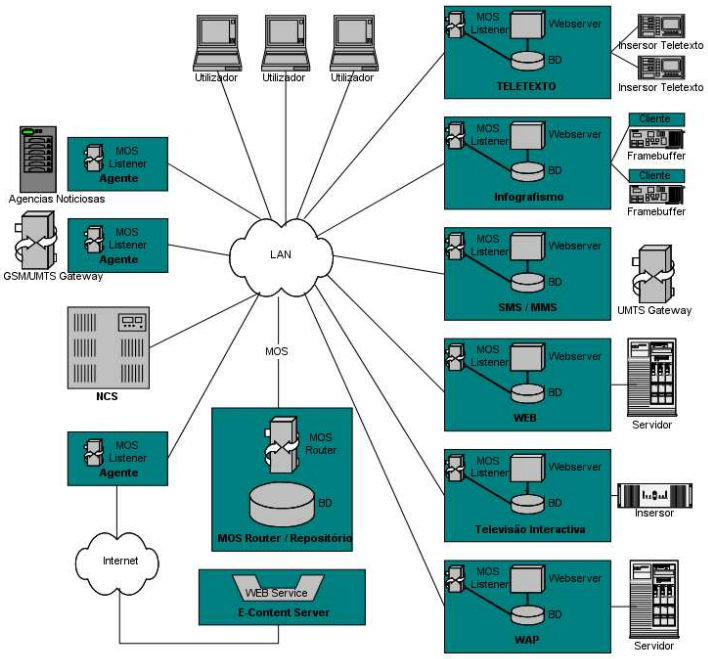


Figura 3. Arquitectura SIGiC

Uma vantagem adicional do protocolo MOS é ser inteiramente baseado em XML. Este aspecto permite passar os pacotes de informação, de forma transparente, através de qualquer rede de dados e, também, que este protocolo seja facilmente transformado num qualquer protocolo proprietário, recorrendo a transformações XML.

Na arquitectura do SIGDiC cada módulo tem funcionalidade própria, mesmo quando isolado. Contudo, quando integrado com os restantes módulos, obtêm-se uma funcionalidade estendida, pelo aumento da integração entre os vários módulos e a consequente automação de processos. Para comunicação entre os vários módulos, será utilizado o protocolo MOS, apresentando-se cada módulo na rede como um equipamento MOS. Assim, é possível operar cada módulo separado da restante arquitectura, como um sistema independente, que pode interoperar directamente com o restante equipamento de redacção.

Apesar de o protocolo MOS não ser um protocolo pensado para troca de conteúdos, a existência do elemento MosExternalMetadata, que não é analisado e que é transportado, transparentemente, nas mensagens, fornece um mecanismo que permite a troca transparente de informação. O SIGDiC explora esta característica para implementar o mecanismo de comunicação interno, que inclui a troca de conteúdos.

O repositório do SIGDiC assume o papel de concentrador de informação e, para além de recepcionar e enviar as mensagens MOS passadas entre módulos,

tem também a função de guardar as várias actualizações realizadas a um conteúdo. Por exemplo, a criação de uma notícia dá origem a uma mensagem MOS de notificação que, ao chegar ao concentrador, é reenviada, de forma automática, para todos os módulos que devam ser notificados e armazenada no repositório.

Por exemplo, a criação de uma mensagem a ser mostrada em rodapé, no módulo de Infografismo do SIGDiC, originaria uma mensagem MOS que, ao ser processada pelo concentrador, originaria um anexo ao conteúdo inicial, que passaria a ser constituído pela notícia e pela sua versão em rodapé.

O concentrador aparece assim como um "potenciador automático"da reutilização de conteúdos, ao manter um repositório permanentemente actualizado, com a informação que se encontra disseminada pelos vários módulos utilizados.

Cada módulo incorpora um componente que designamos por "MOS Listener", o qual pretende implementar o ponto terminal para o protocolo de comunicações. Este componente é sempre constituído por duas partes: 1) uma parte genérica de terminação de protocolo; 2) uma parte específica que implementa as funcionalidades próprias do módulo. De acordo com a mensagem recebida, cada módulo originará diferentes procedimentos, que tratarão a mesma informação de forma também diferente.

4 Implementação da Framework Java para MOS

A adopção do protocolo MOS, como protocolo de comunicação entre todos os módulos do SIGDiC, obriga a criar listeners específicos em cada módulo, que sejam capazes de terminar correctamente o protocolo MOS. Cada listener deverá realizar o processamento das mensagens MOS recebidas, de acordo com a função a desempenhar em cada módulo.

Com o objectivo de promover a reutilização de código fonte, tem vindo a ser desenvolvida uma framework Java para MOS. Esta framework irá implementar toda a parte de terminação do protocolo e fornecer mecanismos expeditos de criação de mensagens, deixando apenas para cada um dos diferentes módulos a implementação do tratamento específico das mensagens recebidas.

As mensagens MOS estão estruturadas em XML. Para podermos manipular facilmente os valores inseridos nos campos da mensagem, optou-se por se utilizar DOM, neste caso JDOM[2].

As principais classes que constituem a framework são:

1. `Start`
   Classe que implementa o método que dá início ao programa. Abre os portos usando as classes MOSUpperPort (para o porto 10541) e MOSLowerPort (para o porto 10540), de forma a receber ligações.
2. `MOSProtocolServer`
   Uma das classes fundamentais porque é a que implementa o protocolo a nível dos sockets. Implementa um servidor multi-thread que resulta na criação de uma nova thread para processar cada pedido. Sempre que o socket recebe um pedido de ligação, é instanciada uma classe do tipo MOSMessageHandler para processar a mensagem.

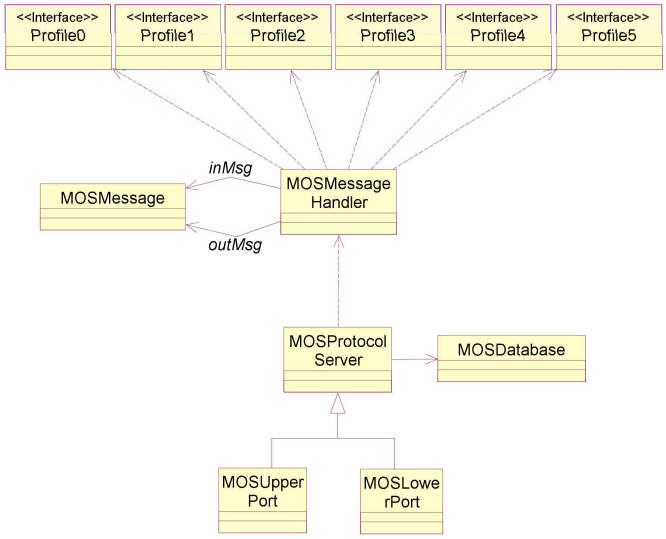


Figura 4. Framework Java

3. `MOSUpperPort,MOSLowerPort`
   Estas duas classes implementam uma especialização de MOSProtocolServer, implementando o serviço em cada um dos portos: MOS Upper Port(10541) e MOS Lower Port(10540).
4. `MOSDatabase`
   Classe utilitária que implementa uma camada de abstracção à base de dados. Implementa métodos utilitários e que encapsulam os métodos JDBC[7], promovendo abstracção de alguma complexidade do JDBC. A configuração da ligação JDBC é obtida, por sua vez, a partir de um ficheiro XML.
5. `Profile0 . . . Profile5`
   Interfaces Java para cada um dos perfis das mensagens MOS. Cada interface inclui a assinatura dos métodos que tratam cada uma das mensagens desse perfil. Qualquer solução deverá implementar as interfaces ProfileX, correspondentes aos perfis que são suportados.
6. `MOSMessageMOSMessage`
   Esta classe está directamente relacionada com o formato de cada mensagem. Inclui diversos construtores para a criação de novas mensagens e funções utilitárias para tratar mensagens, recebidas ou a enviar. A classe MOSMessage é uma subclasse da classe Document, da package JDOM, usada para manter uma representação numa árvore DOM das mensagens XML.
7. `MOSMessageHandler`
   Classe que processa as mensagens recebidas; Invoca o método adequado para o processamento da mensagem; Determina as respostas adequadas a cada mensagem.

Sempre que uma nova mensagem é recebida, o parser do XML é executado através da criação de um novo objecto do tipo MOSMessage. De acordo com o tipo de mensagem MOS recebida, é invocado o método correspondente que irá fazer o tratamento da mensagem, processar o pedido e enviar a resposta.

Em implementações normais de MOS, o programador, praticamente, só terá que implementar o processamento das mensagens recebidas, através da implementação dos métodos das interfaces correspondentes aos Perfis suportados.

5 Conclusões

O protocolo MOS foi desenvolvido por fabricantes, software houses e utilizadores com o objectivo de criar mecanismos que permitissem o controlo sobre os objectos multimédia disponíveis nos servidores de video e geradores de caracteres. Apesar do protocolo não ter sido alvo de um processo de normalização, a sua elevada implantação em meios profissionais de broadcasting, transformou-o num standard *de facto* para ambientes de redacção.

Apesar de não ser um protocolo para troca de conteúdos, o MOS suporta um mecanismo de transporte transparente de informação que pode ser utilizado para estender as funcionalidades do protocolo.

Tirando partido desta funcionalidade foi possível utilizar o protocolo MOS para implementar as comunicações inter-módulos do SIGDiC. Ao implementar cada módulo como um "produto"MOS, cada módulo pode interagir directamente com outros equipamentos profissionais.

Esta possibilidade foi já verificada através de testes em ambiente real, com o protótipo do SIGDiC. Verificou-se que a aplicação de Redacção utilizada na RTP[1], uma solução comercial proprietária, foi capaz de controlar o módulo de infografismo do SIGDiC. O sistema de redacção criou, automaticamente, a informação de noticias passadas no rodapé em programas noticiosos, que eram postas no ar pelo módulo de infografismo.

O desenvolvimento da framework Java, permitiu que o desenvolvimento das várias componentes MOS fosse realmente simplificado, traduzindo-se numa redução substancial do tempo de desenvolvimento.

Referências

[1] AP. Associated Press ENPS - MOS Connection , 2003. `http://www.enps.com/mos/`.

[2] JDOM. JDOM Project Homepage, 2004. `http://www.jdom.org/`.

[3] João Paulo Rodrigues Jorge Bertocchini. Implementação de um protocolo de comunicações baseado em MOS. Projecto/trabalho de fim de curso, FEUP / INESC Porto, 2004.

[4] MOS. MOS Protocol Homepage, 1999. `http://www.php.net`.

[5] João Paulo Rodrigues. SIGDiC - Sistema Integrado de Gestão e Difusão de Conteúdos. Tese de mestrado, INESC Porto, 2004.

[6] Alexandria SC. JavaServide - Java to NT Service Connector, 2003. `http://www.alexandriasc.com/software/JavaService/`.

[7] Sun. Sun JDBC Tutorial , 2002. http://java.sun.com/docs/books/tutorial/jdbc.
[8] W3C. XSL Transformations, 1999. http://www.w3.org/TR/xslt.
[9] W3C. Extensible Markup Language (XML), 2003. http://www.w3.org/XML/.

Documento XML grande: a bela ou o monstro?

Marta H. Jacinto

ITIJ — Instituto das Tecnologias de Informação na Justiça
Ministério da Justiça
1049-068 Lisboa
marta.jacinto@itij.mj.pt

Resumo O ITIJ foi encarregado de enviar os dados da DGRN, que tem inúmeros serviços e vários funcionários em cada um deles, para a BDAP — base de dados que deve conter os dados referentes a todos os funcionários públicos. Para isso foi necessário trabalhar com documentos da ordem dos 22MB e fazer várias transformações sobre eles, algumas resultando em documentos com tamanho na ordem dos 14MB. O facto de as primeiras transformações, feitas em ambiente não académico e para um caso real, demorarem tempos relativamente curtos, motivaram a escrita deste artigo e levaram à avaliação de todas as outras, numa tentativa de responder à questão "Documento XML grande: a bela ou o monstro?". Conclui-se que, apesar de algumas transformações serem feitas mais rapidamente que outras, os documentos XML desta ordem de grandeza são tratáveis.

1 Introdução

O XML tem vindo a ganhar cada vez mais adeptos, conquistando, a pouco e pouco, todas as áreas de informação. O projecto BDAP (Base de Dados de Recursos Humanos da Administração Pública) — descrito na secção 2 — que pretende ser um banco de informação sobre todos os funcionários públicos do país, não é excepção. Para o transporte de informação entre aplicações foi escolhido o XML. Essa escolha adequa-se perfeitamente à diversidade dos vários organismos da Administração Pública, permitindo estabelecer uma plataforma única para o intercâmbio da informação.

O ITIJ (Instituto das Tecnologias de Informação na Justiça), "centro de informática" do Ministério da Justiça, presta serviços a vários serviços daquele Ministério, nomeadamente a DGRN (Direcção-Geral dos Registos e Notariado). Esta última tem vários serviços (Conservatórias do Registo Predial, Conservatórias do Registo Comercial, etc) e dispunha já de uma base de dados com dados sobre os funcionários quando surgiu a iniciativa da BDAP, pelo que incumbiu o ITIJ de fazer as alterações necessárias para recolher os dados adicionais requeridos para a BDAP e posteriormente gerar o XML necessário para o envio. Na secção 3 apresenta-se o formato utilizado para recolher os dados adicionais,

Microsoft Access 2003, e o modo como se gerou automaticamente um dialecto XML standard usando esse programa.

Depois de se dispor dos dados num dialecto XML do Microsoft Access foi necessário transformar esse XML no dialecto XML definido para a BDAP. A complexidade de desenho desta transformação não é grande e o seu processamento é rápido, como se pode ver na secção 4 em que se apresenta o método utilizado para gerar tal XML.

Visto que esta transformação, feita sobre um documento XML de tamanho não académico, demorou pouco tempo, surgiu a ideia de avaliar todas as outras transformações numa tentativa de encontrar uma resposta à questão "Documento XML grande: a bela ou o monstro?". Foi precisamente essa avaliação que motivou a escrita deste artigo.

Uma vez que a introdução de dados no Microsoft Access não é objecto de qualquer validação, foi necessário fazer uma folha de estilos XSL para validar os dados inseridos e gerar um relatório com os erros resultantes da validação, como se mostra na secção 5, permitindo assim a alteração dos dados por parte da DGRN. Nesta mesma secção apresenta-se a forma como se lidou com os erros reportados pelo Instituto de Informática ao receber os ficheiros XML.

Na figura 1 apresenta-se o esquema da arquitectura de todo o sistema que, como enunciado acima, se descreve pormenorizadamente nas secções 2, 3 e 4.

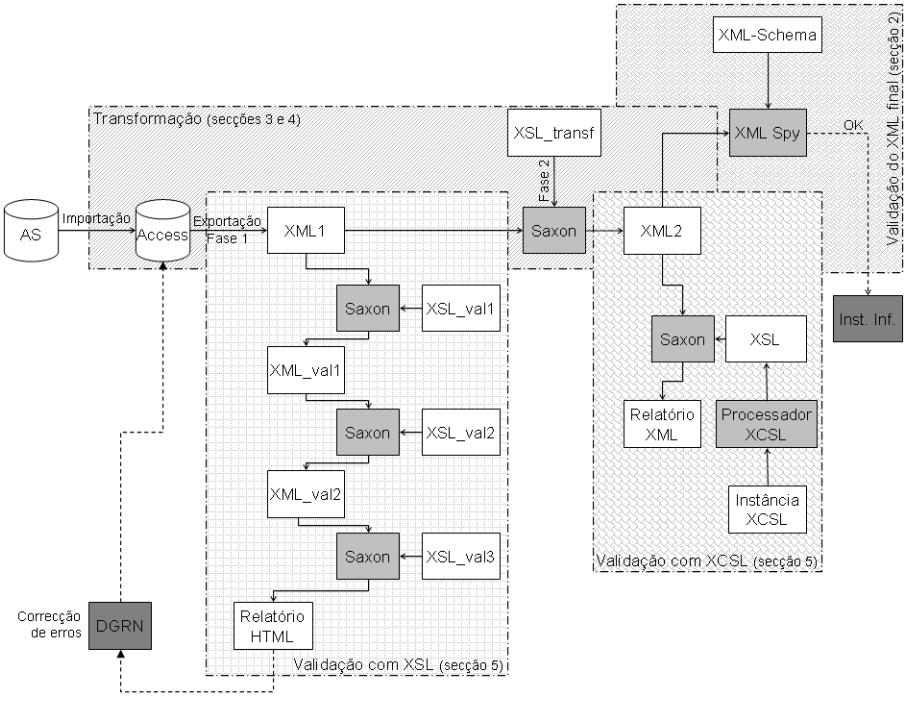


Figura 1. Arquitectura de todo o sistema.

Na secção 6 conclui-se o artigo, apresentando a comparação dos tempos de processamento das várias transformações enumeradas.

Todas as transformações foram feitas usando um PC com processador Pentium 4 com 2,40GHz e 504MB de RAM.

2 BDAP

A BDAP[1] (Base de Dados de Recursos Humanos da Função Pública) criada pelo Decreto-Lei nº 47/98, de 7 de Março obriga a um processo de permanente actualização capaz de fornecer, a todo o momento, a informação necessária à produção de indicadores de gestão e planeamento dos recursos humanos da Administração [1]. É, assim, um sistema de informação que pretende agregar todos os dados relativos aos funcionários públicos, independentemente da natureza do seu vínculo à função pública. Deve conter dados de identificação como o nome, o número de contribuinte, o sexo e a data de nascimento; dados sobre a remuneração e os suplementos de remuneração; as acções de formação; as habilitações literárias; os organismos de funções, vínculo e pagador; etc.

Esta base de dados é da responsabilidade conjunta da DGAP (Direcção-Geral da Administração Pública) e do Instituto de Informática do Ministério das Finanças.

Para o carregamento dos dados há dois processos disponíveis: interactivo — para organismos até 50 trabalhadores e sem sistema de gestão de recursos humanos; envio de um ficheiro XML — para organismos com mais de 50 trabalhadores.

Tendo vários milhares de trabalhadores, a DGRN faz parte deste segundo grupo de organismos.

O Instituto de Informática e a DGAP estabeleceram algumas regras para a escrita destes documentos XML, que constam de [2], para além de outros documentos com regras sobre a recolha dos dados e os vários carregamentos. Foi disponibilizado ainda um esqueleto do documento XML que, apesar de nada dizer quanto à cardinalidade de cada elemento, dá uma ideia dos elementos a utilizar e suas dependências [3]. A seguir apresenta-se um extracto desse esqueleto:

```
<entidaderemetente cod_local_trabalho="" cod_servico="" cod_organismo="">
  <organismo cod_organismo="" num_fiscal="">
     <dataref/>
     <nenvio/>
     <tipo_operacao/>
     <trabalhador num_fiscal="">
        <cenario/>
        <nome/>
        <sexo/>
        ...
        <data_nasc/>
        <nacionalidade/>
        <antiguidade_fp>
```

[1] http://www.bdap.min-financas.pt

```
            ...
         </antiguidade_fp>
         <emprego_organismo>
            ...
            <situacao_profissional>
               ...
            </situacao_profissional>
            ...
            <remuneracao>
               ...
            </remuneracao>
            ...
         </emprego_organismo>
         <habilitacoes_literarias data_obtencao="">
            ...
         </habilitacoes_literarias>
         <formacao_mais_18h data="">
            ...
         </formacao_mais_18h>
      </trabalhador>
   </organismo>
</entidaderemetente>
```

Uma vez que o Instituto de Informática não disponibilizou qualquer XML-Schema, foi necessário escrever um XML-Schema ([4,5]) que agregasse tanto quanto possível a informação constante nos vários documentos já referidos e permitisse depois validar os documentos XML gerados para envio — neste caso não se põe o problema de ajudar o utilizador a escrever documentos mas sim de ter um XML-Schema para ajudar a validar o documento XML final. Nele foram incluídas todas as indicações dos documentos [2] e [6]. Abaixo encontra-se um extracto desse XML-Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="entidaderemetente">
  <xs:complexType>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="organismo" type="torganismo"/>
   </xs:sequence>
   <xs:attribute name="cod_local_trabalho" type="tnum6" use="required"/>
   <xs:attribute name="cod_servico" type="tnum6" use="required"/>
   <xs:attribute name="cod_organismo" type="tnum6" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="tremuneracao_mensal">
  <xs:sequence>
```

```
   ...
   <xs:element name="suplementos_regulares" type="tsuplementos_regulares"
   minOccurs="0" maxOccurs="unbounded"/>
   ...
  </xs:sequence>
 </xs:complexType>
 ...
 <xs:complexType name="tsuplementos_regulares">
  <xs:sequence>
    <xs:element name="codigo" type="tnum3"/>
    <xs:element name="valor_euros" type="treal42"/>
    ...
  </xs:sequence>
 </xs:complexType>
 ...
 <xs:simpleType name="treal42">
  <xs:restriction base="xs:string">
    <xs:pattern value="([0-4]{0,1}[0-9]{0,3}|[0-4]{0,1}[0-9]{0,3},
    [0-9]{0,2})"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

3 Origem dos dados

Quando surgiu a iniciativa da BDAP já havia um conjunto de dados relativos aos funcionários da DGRN em AS. Esses dados foram transferidos para uma base de dados em Microsoft Access, utilizando-se posteriormente o próprio Microsoft Access para recolher os dados para a BDAP.

Depois da recolha colocou-se a questão da transformação dos dados recolhidos para o XML pedido pela BDAP. Este problema foi resolvido em duas fases: 1^a — exportação automática dos dados para um dialecto XML standard do Microsoft Access; 2^a — transformação do dialecto XML gerado automaticamente no dialecto XML da BDAP.

Para a 1^a fase utilizou-se uma opção disponível a partir da versão 2003 do Microsoft Access que permite exportar os dados para XML, ou seja, gerar automaticamente XML a partir de uma base de dados, colocando um elemento raiz com o nome *dataroot*; tantos elementos filho com o nome da tabela (neste caso `T_Bdap`) quantos os registos; e cada um destes com tantos elementos filho quantos os campos da tabela preenchidos no caso desse registo. Os caracteres especiais são traduzidos automaticamente para conjuntos de caracteres, por exemplo "(" é substituído por "_x0028_" e " " é substituído por "_x0020_".

A geração automática de XML a partir da base de dados em Microsoft Access com 38MB demora 7 minutos, gerando um documento da seguinte forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="bdap_access_20041022.xsd"
generated="2004-10-22T15:28:35">
 <T_Bdap>
  <Cod_Serviço_x0028_interno_x0029_>0106401
       </Cod_Serviço_x0028_interno_x0029_>
  <ID>375</ID>
  <Serviço>Conservatória Registo Civil, Predial Castelo Paiva</Serviço>
  ...
  <Dataref>2003-06-30T00:00:00</Dataref>
  <Nenvio>1</Nenvio>
  <Tipo_operacao>A</Tipo_operacao>
  <Data_x0020_Ingresso>1981-01-27T00:00:00</Data_x0020_Ingresso>
  <Cod_Ingresso>1</Cod_Ingresso>
  <N_x00BA__x0020_Fiscal_x0020_Contribuinte_Trab>112529597
       </N_x00BA__x0020_Fiscal_x0020_Contribuinte_Trab>
  <Cenario>1.1</Cenario>
  <Nome_x0020_Completo>Guilhermina Cunha Santos</Nome_x0020_Completo>
  <Cod_categoria_x0028_interno_x0029_>24
       </Cod_categoria_x0028_interno_x0029_>
  <Desc_categoria_x0028_interno_x0029_>ESCRITURARIO SUPERIOR
       </Desc_categoria_x0028_interno_x0029_>
  <Especie_Tipo>CP</Especie_Tipo>
  <Sexo>F</Sexo>
  <Cod_Nacionalidade>201</Cod_Nacionalidade>
  <Cod_Distrito>01</Cod_Distrito>
  <Cod_Concelho>0601</Cod_Concelho>
  <Cod_Freguesia>090601</Cod_Freguesia>
  <Data_x0020_Nascimento>1957-06-18T00:00:00</Data_x0020_Nascimento>
  <Cod_x0020_Relação_x0020_Jurídica>12</Cod_x0020_Relação_x0020_Jurídica>
  ...
 </T_Bdap>
 ...
 <T_Bdap>...</T_Bdap>
</dataroot>
```

Para os cerca de 6200 funcionários inseridos até ao momento, este ficheiro XML ficou com 22MB, o que representa um tamanho considerável para um documento XML. A manipulação deste ficheiro com o XMLSpy[2] é bastante morosa devido à identificação com cores que o programa faz dos elementos e dos atributos, mas é possível.

Repare-se que o facto de os vários campos de um registo serem colocados todos como irmãos e filhos de um mesmo elemento simplifica significativamente as transformações a efectuar sobre este documento para gerar o definitivo, i.é., a 2^a fase que se apresenta na secção seguinte.

[2] http://www.xmlspy.com

4 Transformação

No caso da 2^a fase da transformação dos dados recolhidos para o XML pedido para a BDAP (o definido na secção 2), foi necessário fazer uma folha de estilos XSL [7,8]. Esta folha de estilos permite gerar o XML de destino a partir do obtido na 1^a fase da transformação.

Na construção da folha de estilos houve necessidade de fazer duas abordagens diferentes, consoante os elementos fossem ou não obrigatórios. Assim, a transformação dos elementos obrigatórios seguiu um esquema como exemplificado abaixo:

```
<sub_grupo><xsl:value-of select="Profissao_Sub_Grupo"/></sub_grupo>

<data_inicio>
   <xsl:value-of select="substring(Prof_Data_Inicio_Carreira,1,10)"/>
</data_inicio>

<remuneracao_base_euros>
  <xsl:value-of select="translate(Remuneração_x0020_Base,'.',',')"/>
</remuneracao_base_euros>
```

ou seja, limitámos-nos a escrever para cada um dos elementos de destino o valor original correspondente. No entanto no caso das datas foi necessário usar a função *substring* para retirar apenas os primeiros dez caracteres (uma vez que o campo no Microsoft Access é do tipo data e hora) e no caso dos valores monetários foi necessário usar a função *translate* para traduzir o separador da parte decimal de "." para ",".

No caso dos elementos opcionais foi necessário avaliar, para cada um deles, se o elemento de origem estava ou não preenchido e, caso afirmativo, escrever o elemento de destino com o valor desse elemento. Assim, a transformação seguiu um esquema semelhante ao seguinte:

```
<xsl:if test="string-length(Data_x0020_Saída)&gt;=1">
  <data_cessacao>
     <xsl:value-of select="substring(Data_x0020_Saída,1,10)"/>
  </data_cessacao>
</xsl:if>

<xsl:if test="string-length(Cod_x0020_Suplemento_1)&gt;=1">
   <suplementos_regulares>
     <codigo><xsl:value-of select="Cod_x0020_Suplemento_1"/></codigo>
     <valor_euros>
        <xsl:value-of select="translate(Valor_Suplemento_1,'.',',')"/>
     </valor_euros>
     <ano><xsl:value-of select="Ano_Supl_1"/></ano>
     <mes><xsl:value-of select="Mês_Supl_1"/></mes>
   </suplementos_regulares>
</xsl:if>
```

Nesta 2^a fase, a transformação, na linha de comandos e usando o saxon[3], é feita em 12 segundos, gerando um ficheiro com 14MB. O facto de a transformação exigir poucos cálculos pode justificar esta rapidez.

Para verificar a validade do ficheiro XML final contra o XML-Schema apresentado na secção 2 usou-se o XMLSpy. Esta validação demorou cerca de 1 minuto e meio, evidenciando mais uma vez que os documentos XML grandes são tratáveis.

5 Validação

Como se disse anteriormente, a introdução de dados no Microsoft Access não é objecto de qualquer validação em tempo real, tornando necessário efectuar validações sobre o documento XML.

Houve duas razões para procurar uma alternativa ao XML-Schema para a validação: a validação dos dados contra o XML-Schema pára a cada erro, e era necessário gerar um relatório amigável em HTML com os vários erros para que a DGRN os pudesse corrigir por uma questão de celeridade; não é possível fazer certas validações com um XML-Schema [9].

A solução inicialmente encontrada foi utilizar o XCSL [10,11], uma linguagem de especificação de restrições sobre documentos XML que permite validar todos os tipos de condições conhecidas [12]. Escreveu-se então o documento XCSL para algumas validações, por exemplo:

```
<constraint>
  <selector selexp="//trabalhador"/>
  <let name="chavetrab" value="@num_fiscal"></let>
  <cc>count(//trabalhador[@num_fiscal=$chavetrab])=1</cc>
  <action><message>
       <value selexp="@num_fiscal"/>
      (<value selexp="count(//trabalhador[@num_fiscal=$chavetrab])"/>
      iguais).
  </message></action>
</constraint>
```

que permite avaliar se há algum número de contribuinte fiscal utilizado para mais do que um funcionário, ou seja, encontrar os trabalhadores repetidos. Conforme se pode ver na caixa inferior direita da figura 1, este documento de restrições é transformado por um processador específico resultando num documento XSL. Este último é depois aplicado ao documento XML final (resultante da 2^a fase de transformação) para obter o relatório de erros em XML.

No entanto a utilização desta ferramenta não permitia separar os erros encontrados por serviço e, para cada serviço, por trabalhador, separação esta que facilitaria substancialmente a correcção dos erros. Este facto fez com que posteriormente se optasse por fazer um documento XSL de raiz para as validações sobre

[3] http://users.iclway.co.uk/mhkay/saxon

o documento XML original (obtido na 1ª fase de transformação), deixando em XCSL apenas a validação apresentada como exemplo. A validação do documento XML final contra o documento XSL gerado pelo processador XCSL apenas com aquela restrição demorou 33 minutos.

Por forma a tornar o relatório HTML o mais pequeno possível e apenas com a informação necessária (não interessa que sejam listados os serviços nem os trabalhadores para os quais não sejam detectados erros), a geração do ficheiro HTML foi feita em três fases (de acordo com a caixa inferior esquerda da figura 1):

1. Documento XSL para gerar documento XML intermédio com todos os elementos (mesmo os sem conteúdo) e por ordem de serviço. A seguir apresenta-se um extracto dessa folha de estilos:

```
...
<xsl:template match="dataroot">
 ...
 <xsl:variable name="servico2" select="//Cod_Serviço_BDAP">
 </xsl:variable>
 <xsl:for-each select="T_Bdap[Cod_Serviço_BDAP=$servico2]">
  <xsl:sort order="ascending"/>
  <xsl:apply-templates select="."/>
 </xsl:for-each>
 <xsl:text disable-output-escaping="yes">&lt;/dataroot></xsl:text>
</xsl:template>
...
<xsl:template match="T_Bdap">
 <T_Bdap>
  <Cod_Serviço_x0028_interno_x0029_>
   <xsl:apply-templates select="Cod_Serviço_x0028_interno_x0029_"/>
  </Cod_Serviço_x0028_interno_x0029_>
 ...
```

   Esta fase do processamento é muito rápida, demorando apenas 43 segundos e gerando um ficheiro com 33MB. É de notar que, ao contrário do que acontecia na 2ª fase da geração do XML para envio, não são usadas quaisquer funções nem nenhum elemento *xsl:if*.

2. Documento XSL para gerar um ficheiro XML com elemento raiz *tudo* e um elemento *servico* para cada servico. *servico* vai conter os sub-elementos: *nome* — o nome do serviço; *cod* — o código do serviço; *trabalhador* — elemento para cada trabalhador daquele serviço, independentemente de haver ou não erros de preenchimento relacionados com esse trabalhador. *trabalhador* tem por sua vez um sub-elemento *nome* e tantos sub-elementos *li* quantos os erros encontrados para esse trabalhador. A seguir apresenta-se um extracto dessa folha de estilos:

```
<xsl:template match="dataroot">
 <tudo><xsl:apply-templates select="T_Bdap" mode="m1"/></tudo>
</xsl:template>
<xsl:template match="T_Bdap" mode="m1">
 <xsl:if test="position()=1">
  <xsl:text disable-output-escaping="yes">&lt;servico></xsl:text>
  <nome><xsl:value-of select="Serviço"/></nome>
  <cod>
   <xsl:value-of select="Cod_Serviço_x0028_interno_x0029_"/>
  </cod>
 </xsl:if>
 ...
 <xsl:if test="not(Indicador_Nao_Envio='s' or
       Indicador_Nao_Envio='S')">
  <xsl:apply-templates select="." mode="m2"/>
 </xsl:if>
 ...
</xsl:template>
<xsl:template match="T_Bdap" mode="m2">
 <trabalhador>
  <nome> ... </nome>
  <xsl:if test="string-length(Cod_Local_trabalho)&lt;1">
   <li>Falta preencher o campo "Cod_Local_trabalho"</li>
  </xsl:if>
  ...
  <xsl:if test="(HabiIitações_x0028_1_x0020_a_x0020_9_x0029_=7 or
   HabiIitações_x0028_1_x0020_a_x0020_9_x0029_=8 or
   ...">
   <li>O campo "Área_Temática_(1_a_9)" tem que ser preenchido
   já que o campo "HabiIitações_(1_a_9)" vale
   <xsl:value-of select="
       HabiIitações_x0028_1_x0020_a_x0020_9_x0029_"/>.</li>
  </xsl:if>
  ...
  <xsl:variable name="dreft" select="concat(substring($dref,1,4),
  substring($dref,6,2),substring($dref,9,2))"/>
  <xsl:variable name="mesg" select="concat('deve valer entre ',
       $dnmd,' e ',$dref,'.')"/>
  <xsl:if test="not(
   (concat(substring(Prof_Data_Inicio_Carreira,1,4),
   substring(Prof_Data_Inicio_Carreira,6,2),
   substring(Prof_Data_Inicio_Carreira,9,2)) &gt;= $dnasct + $ia) and
   ...">
   <li>Prof_Data_Inicio_Carreira vale
    <xsl:value-of select="substring(Prof_Data_Inicio_Carreira,1,10)"/>
    e <xsl:value-of select="$mesg"/>
   </li>
  </xsl:if>
  ...
```

Esta fase do processamento demora 1 minuto e 5 segundos, gerando um documento cujo tamanho depende do número de erros encontrados. A diferença de tempo face à fase anterior tem a ver com o facto de neste caso se usarem várias funções e elementos *xsl:if*. Segue-se um elemento *trabalhador* resultado:

```
<trabalhador>
  <nome>MARIA CELESTE BARROSO DA GRAÇA BOSQUET - 142614785</nome>
  <li>Valor_Suplemento_1 vale 44038 e tem que valer entre 0 e 4999.</li>
  <li>Cod_Relação_Jurídica não pode valer 12 para o cenário 1.1.</li>
</trabalhador>
```

3. Documento XSL para gerar um ficheiro HTML com a enumeração dos vários serviços para os quais foram detectados erros, descrevendo os trabalhadores com erros de preenchimento e quais são esses erros. A seguir apresenta-se um extracto dessa folha de estilos:

```
<xsl:template match="/">
  <html>
    ...
    <body><xsl:apply-templates select="tudo"/></body>
  </html>
</xsl:template>
<xsl:template match="tudo">
  <xsl:choose>
    <xsl:when test="count(//li)=0">
      <h3 align="center">
        <font color="green">
           Validação efectuada sem encontrar erros!
        </font>
      </h3>
    </xsl:when>
    <xsl:otherwise>
      <h3 align="center">
        <font color="green">
           Erros no preenchimento da Base de Dados
        </font>
      </h3>
      <xsl:apply-templates select="servico" mode="m1"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="servico" mode="m1">
 <xsl:if test="count(trabalhador[count(li)!=0])!=0">
   <xsl:if test="position()=1">
     <hr color="#800000"/>
     <font color="blue" size="4">
        <xsl:value-of select="nome"/> (<xsl:value-of select="cod"/>)
     </font>
     <br/><br/>
```

```
    </xsl:if>
            ...
    <xsl:apply-templates select="trabalhador" mode="m2"/>
  </xsl:if>
 </xsl:template>
 <xsl:template match="trabalhador" mode="m2">
    ...
    - <xsl:value-of select="."/>
 </xsl:template>
```

Nesta fase, o tempo do processamento depende obviamente do tamanho do ficheiro obtido na fase anterior (do número de erros). Para o caso em que aquele tem aproximadamente 1MB demora menos de dois segundos. A figura 2 apresenta um ficheiro HTML resultado.

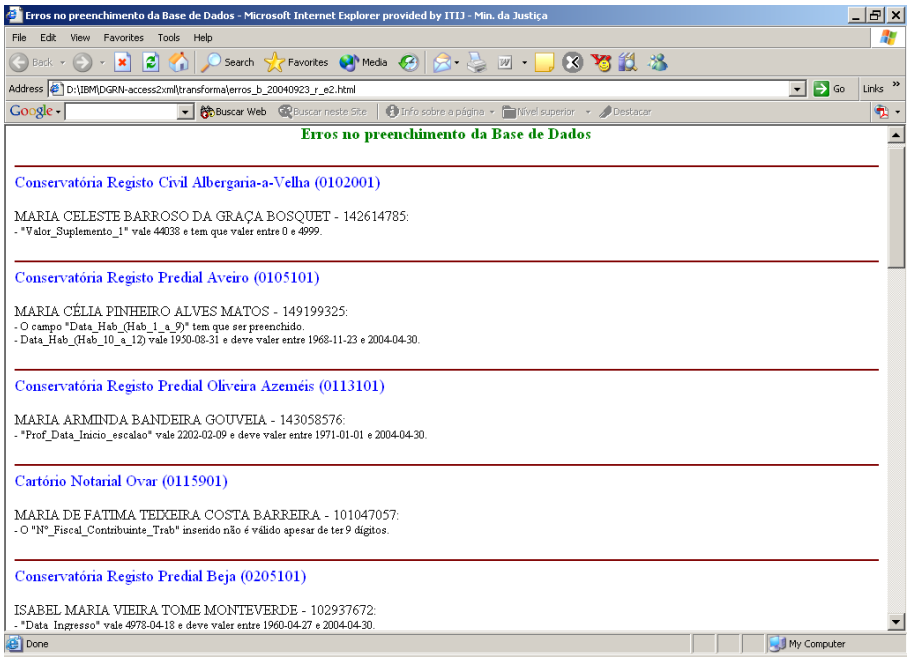


Figura 2. Documento HTML com os erros de validação

Quando os ficheiros XML submetidos a todas aquelas validações foram enviados para o Instituto de Informática, este organismo devolveu uma lista com erros que não constavam dos vários documentos disponibilizados. As novas validações foram incluídas no XML-Schema quando possível e no documento XSL da segunda fase do processo explicado anteriormente.

6 Conclusão

A necessidade de efectuar várias transformações sobre documentos XML de tamanho não académico trouxe a oportunidade de fazer uma avaliação cuidada do tempo desse processamento numa situação real. Verificou-se que a transformação sobre documentos XML grandes (22 MB) é praticamente imediata quando é simples, ou seja, não envolve muitas condições. Quando se aumenta a complexidade da transformação (no caso para analisar a existência de erros), o tempo de processamento aumenta substancialmente, não sendo contudo proibitivo. A única situação em que de facto o processamento de um documento XML é consideravelmente demorado é aquela em que se analisa os elementos repetidos.

Conclui-se então que os documentos XML grandes não são monstros e são processáveis em tempo real.

Referências

1. Instituto de Informática e DGAP: Glossário BDAP (2003)
2. Instituto de Informática: Instruções sobre o Ficheiro XML da BDAP (2003)
3. Instituto de Informática: Código fonte do ficheiro XML. (2003)
4. Duckett, J., Griffin, O., Mohr, S., Norton, F., Ozu, N., Stokes-Rees, I., Tennison, J., Williams, K., Cagle, K.: Professional XML Schemas. Wrox Press (2001)
5. Eric van der Vlist: XML Schema. O'Reilly & Associates, Inc. (2002)
6. Instituto de Informática e DGAP: Regras e definições para preenchimento dos dados (2003)
7. Bradley, Neil: The XSL companion. Addison-Wesley (2000)
8. K. Cagle and M. Corning and J. Diamond and T. Duynstee and O. Gudmundsson and J. Jirat and M. Mason and J. Pinnock and P. Spencer and J. Tang and P. Tchistopolskii and J. Tennison and A. Watt: Professional XSL. Wrox Press (2001)
9. Marta Henriques Jacinto, Giovani Rubert Librelotto, José Carlos Ramalho, Pedro Rangel Henriques: Constraint Specification Languages: comparing XCSL, Schematron and XML-Schema. Actas do XML Europe'02, Barcelona, Espanha (2002)
10. José Carlos Leite Ramalho:Anotação Estrutural de Documentos e sua Semântica. Universidade do Minho (2000)
11. Marta H. Jacinto, Giovani R. Librelotto, José C. Ramalho, Pedro R. Henriques: XCSL: XML Constraint Specification Language. Actas da conferência CLEI02 - XXVIII Conferencia Latino Americana de Informática, Montevideo, Uruguai (2002)
12. Marta Henriques Jacinto:Validação Semântica em Documentos XML. Universidade do Minho (2002)

Information Router: Plataforma *webservice* de comunicação entre aplicações

Pedro Silva[1], José Castro[1], and Ildemundo Roque[1]

Telbit, Tecnologias de Informação
Rua Banda da Amizade, 38 r/c Dto. 3810-059 Aveiro
{psilva, jcastro, iroque}@telbit.pt
http://www.telbit.pt

Resumo O Information Router consiste numa arquitectura genérica de interligação de aplicações recorrendo a *webservices*.
O sistema permite garantir, de uma forma fácil, a interoperabilidade e a simplificação das tarefas de configuração da comunicação entre aplicações distintas.
A arquitectura é centralizada e cria uma camada de direcções entre as várias aplicações, de forma a que as mesmas não necessitem de conhecer as características dos sistemas com as quais vão comunicar.
As possibilidades desta ferramenta vão desde o simples reencaminhamento de correio electrónico para outras formas de comunicação, como as mensagens instantâneas o *SMS* ou directamente para formato físico, até à possibilidade de dar suporte aos utilizadores de uma determinada aplicação de várias formas distintas (correio electrónico, *SMS*, mensagens instantâneas, telefone,...), permitindo que o sistema escolha de acordo com a configuração o método a utilizar.
A arquitectura foi inspirada nos sistemas do tipo *Message Oriented Middleware (MOM)* e motivada pela disponibilização de um *workflow* altamente flexível entre aplicações, total transparência na comunicação e no minimizar as tarefas de configuração específicas a cada aplicação.

1 Introdução

A interligação de aplicações é uma necessidade recorrente em ambientes informáticos. Essas necessidades têm muitas vezes de ser pensadas de raiz, e para serem implementadas é comum recorrer a algum tipo de programação distribuída muitas vezes dependente da tecnologia subjacente.

Necessidades específicas e menos vulgares originam a modificação das aplicações de forma a que exista essa interligação. Esse tipo de modificação, quando possível, implica a alteração do código fonte das aplicações. Após a alteração com os devidos transtornos de *debugging* e testes, a aplicação volta a entrar em produção.

O problema surge quando uma nova interligação com uma determinada aplicação é necessária, pois tal implica voltar a reproduzir o mesmo ciclo. Se a

interligação tiver de ser aplicada rapidamente, tal não será possível devido ao tempo despendido no desenvolvimento. A configuração das interligações em que é a aplicação a gerir as suas ligações retira flexibilidade ao sistema.

O Information Router (IR) surge da necessidade de responder a esses problemas e de dotar a interligação de aplicações de maior flexibilidade. Esse requisito implica também que a configuração da arquitectura seja feita rapidamente, sem impactos negativos quer nas configurações já existentes, quer no funcionamento corrente da mesma. A forma de configuração deve também permitir implementar de forma intuitiva algum tipo, ainda que simples, de *workflow* entre as aplicações.

É uma arquitectura centralizada e genérica pensada para a interligação de aplicações. A mesma foi inspirada em sistemas *MOM* como o *Java Message Service (JMS)*, cuja ideia base é a existência de uma camada de abstracção que passa a existir entre o cliente e o servidor. Essa camada recebe mensagens de um ou mais *produtores de mensagens* e faz o *broadcast* para um ou mais *consumidores de mensagens*.

No entanto, apesar de ser inspirado em sistemas *MOM* é construído sobre um paradigma de programação distribuída, os *webservices*. Ou seja a arquitectura do IR é um modelo híbrido de programação distribuída, em que existe uma desacopolação entre as aplicações inspirada nos sistemas *MOM*.

A exposição da arquitectura do IR é iniciada com uma visão geral da mesma, onde é feito o paralelo com outras arquitecturas baseadas em modelos de programação distribuída. De seguida é descrito o interface de contacto dos clientes para com o núcleo decisor. A organização interna do núcleo decisor é exposta com algum detalhe na secção seguinte, sendo complementada com alguns exemplos simples de aplicação do IR. As tecnologias usadas são apresentadas de seguida, bem como algumas considerações acerca de decisões tomadas na escolha das mesmas. A exposição termina com algumas conclusões relativas à arquitectura.

2 Arquitectura

A arquitectura do IR foi projectada tendo como base duas características, simplicidade e flexibilidade. São características sempre desejáveis em qualquer sistema, embora muitas vezes difíceis de conseguir e ainda mais difíceis de fazer coabitar.

2.1 Visão geral

De forma a que a compreensão da arquitectura seja clara irá ser feito um paralelo com arquitecturas baseadas em modelos de programação distribuída, como o *Remote Method Invocation (RMI)* do *Java* e com uma arquitectura do tipo *MOM*. A grande diferença entre o *RMI* e o *MOM* consiste no facto de que em sistemas *MOM*, o contacto entre as aplicações não acontece directamente sendo usado um *middleware de mensagens* para o efeito, como podemos ver na Figura 1.

O IR tem, de grosso modo, uma forma de funcionamento similar aos sistemas *MOM*. As principais diferenças residem no facto da invocação sobre o núcleo do

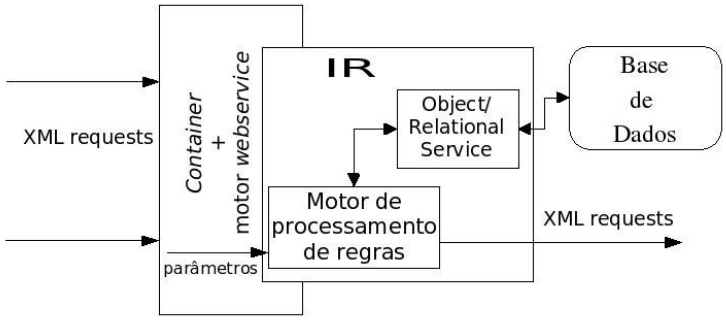


Figura 1. RMI vs MOM

IR ser feita recorrendo a *webservices*, e não usando uma *API* possivelmente dependente da tecnologia. E no facto de o encaminhamento ser *decidido* pelo IR e não por mecanismos de subscrição.

2.2 Os interfaces

O contacto com o núcleo é feito por um interface exposto fixo. Os seus parâmetros são genéricos e assemelham-se (embora não tão complexos) aos cabeçalhos utilizados em protocolos de encaminhamento de dados, onde podemos encontrar estes parâmetros: identificação do cliente, identificação da máquina, dados temporais da invocação, atributos genéricos, acção que desencadeou a invocação, uma mensagem por defeito, etc. É com base nesses parâmetros que o encaminhamento da invocação se dá.

2.3 O núcleo do IR

O IR é uma arquitectura centralizada e como tal, existe um núcleo que controla todos os processos. A única parte visível para o exterior desse núcleo é o interface *webservice*.

Para que uma aplicação possa contactar o IR tem de estar registada no mesmo. Dessa forma o IR passa a ter um nível de consciência adequado a estabelecer a comunicação entre as aplicações. O registo é feito administrativamente. As informações a guardar acerca do cliente são as mais diversas, de entre as quais o endereço *webservice* de destino para um contacto posterior. Desta forma um cliente pode contactar e ser contactado caso exista essa necessidade.

Ainda associado a cada cliente podem existir contactos individuais. Esses contactos individuais definem destinos ainda mais específicos dentro do cliente destino, que o próprio saberá interpretar devidamente. Os exemplos de contactos individuais vão desde endereços de correio electrónico, a contactos de mensagens instantâneas, números de telemóvel, entre outros. Dentro de uma regra é possível

escolher um desses destinos mais específicos quer directamente, quer mediante algum critério (ex. escolha de um sub-conjunto de contactos). A Figura 2 mostra um esquema do núcleo do IR.

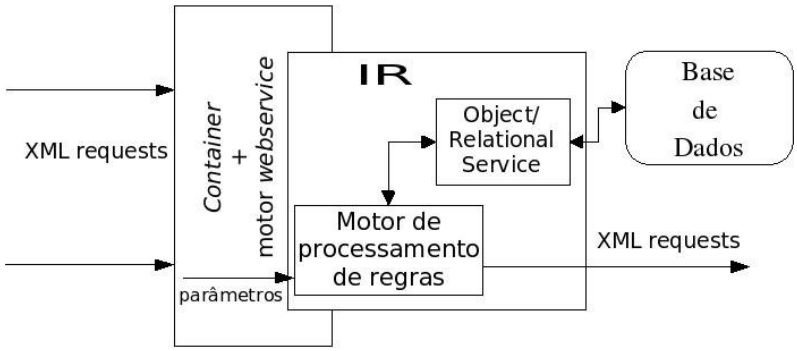


Figura 2. Blocos funcionais do núcleo do IR

2.4 As regras

Associado a um cliente existe um mecanismo que permite decidir que destinos contactar. A decisão é feita mediante um conjunto de regras que quando aplicadas sobre os valores de uma invocação, permitem decidir que clientes contactar. As regras existem no núcleo associadas a um determinado cliente. As mesmas estão organizadas por grupos e dentro de cada grupo as regras não são independentes entre si. Junto com cada regra existem dois valores lógicos, um que decide se o processamento continua para a regra seguinte, e outro que decide se o processamento continua no grupo seguinte. Essa escolha tem em conta a valoração actual da regra. Desta forma para além de poder proporcionar uma organização lógica das regras, podemos flexibilizar e acelerar o processamento das mesmas.

A escolha dos clientes a contactar está presente na própria regra, e os mesmos têm de estar registados no sistema. Ou seja, a validação da regra implica o contacto dos clientes discriminados nas regras. Alguns exemplos dessas regras podem ser vistos na Tabela 1[1].

[1] São exemplos simples de utilização, são também permitidos operadores lógicos encadeados `and, or, not`.

Tabela 1. Exemplos simples da linguagem de regras

Regra	Descrição
`(host match ".*") forward to jabber;`	Qualquer que seja o *host*, encaminha para todos os contactos do cliente *jabber*.
`(client = "vitae") forward to sms match "96.*";`	Se o cliente for o *vitae*, encaminha para o cliente *sms*, cujos contactos se iniciem por 96.

3 Exemplos de aplicação

3.1 Suporte

Um exemplo simples de aplicação, pode ser reconhecido no cenário em que é necessário dar suporte a uma determinada ferramenta a um determinado cliente. O cliente requer suporte por correio electrónico. No entanto a pessoa que lhe dá esse suporte poderá ser contactada de várias formas.

Quando chega uma mensagem de correio electrónico a pedir suporte, a aplicação que gere o endereço de suporte contacta o IR. Mediante as regras associadas à aplicação e mediante os dados que a aplicação envia aquando da invocação, o IR pode decidir contactar o responsável pelo suporte de várias formas, por mensagens instantâneas por correio electrónico, por *SMS* e também pode ser adicionada uma entrada no registo de ocorrências. As decisões podem ser tomadas por exemplo, mediante quem recebe o correio electrónico, consoante a data e hora e pode tomar uma acção diferente consoante seja fim de semana ou dia útil. Uma possível resposta pode também ser encaminhada de forma idêntica. Esses contactos não são feitos directamente pelo IR, o IR contacta aplicações que saibam como o fazer e o que fazer com os mesmos.

3.2 Recepção de *Curriculuns Vitæ*

A recepção de um *Curriculum Vitæ* via um interface *web* mostra um caso simples de utilização. A introdução de um *Curriculum Vitæ* pode fazer despoletar uma invocação sobre o IR. Dentro do mesmo o processamento das regras pode levar ao contacto da pessoa responsável pelos recursos humanos de várias formas, correio electrónico, *SMS*, etc. Pode também ser contactada uma aplicação que trate da impressão directa dos dados mais relevantes de forma a que o *Curriculum Vitæ* possa ser avaliado mais rapidamente. Mais uma vez, esses contactos não são feitos directamente pelo IR, o IR contacta aplicações que saibam como o fazer e o que fazer com os mesmos.

4 Tecnologia

Várias ferramentas foram testadas durante o processo de análise e de decisão acerca da tecnologia a ser utilizada. Algumas hipóteses foram deixadas de parte como é o caso do uso de *Java 2 Enterprise Edition (J2EE)*[2][3]. A complexidade introduzida com o seu uso fazia cair a simplicidade desejada quer para a construção da aplicação, quer depois para o seu ciclo de vida em produção com o respectivo acréscimo de complexidade na manutenção. A complexidade introduzida com o seu uso não era traduzida por quaisquer ganhos.

O núcleo do IR está construído recorrendo à linguagem Java no entanto, como o contacto com o mesmo é feito via *webservices* os clientes podem ser escritos em qualquer linguagem.

Na construção do núcleo foram usadas diversas ferramentas. A combinação `Tomcat + Axis` foi usada para a exposição dos *webservices*. O `Hibernate`[1] foi usado para tratar da persistência dos objectos para o paradigma relacional. A base de dados usada foi o `PostgreSQL` no entanto quase sem qualquer esforço adicional possa ser usada outra qualquer, desde que tenha o suporte por parte do `Hibernate`[1]. Para o processamento das regras a escolha recorreu sobre o `Antlr`, uma ferramenta que permite construir reconhecedores, compiladores e tradutores a partir de uma descrição gramatical.

5 Conclusões

Após a análise da arquitectura e do núcleo podemos verificar que quer a arquitectura, quer o núcleo são simples. O acesso é feito por um interface normalizado suficientemente genérico para acomodar quaisquer tipo de aplicações, esse interface e os parâmetros do mesmo são a única coisa que um cliente necessita de conhecer para contactar o núcleo. Como os interfaces de acesso estão expostos via *webservices* os clientes podem estar escritos em qualquer linguagem.

O funcionamento do núcleo para além de simples é também bastante flexível. A flexibilidade advém da facilidade com que se adiciona um novo cliente e respectivos dados associados, do encaminhamento baseado por regras, da sua organização por grupos e da influência que a valoração de uma regra tem no processamento da regra e do grupo de regras seguinte. Um novo cliente e os dados referentes ao mesmo podem adicionado através do interface de gestão do núcleo do IR. As regras permitem que várias possibilidades de análise sobre as mensagens sejam definidas e que novos destinos sejam adicionados, isto sem que seja necessário alterar quer os clientes, quer o núcleo. A organização por grupos e a influência que a valoração de uma regra tem na avaliação da regra e do grupo de regras seguinte, permite uma organização das mesmas de uma forma inteligente para minimizar a quantidade de regras processadas.

Os dados a ser armazenados acerca das aplicações cliente foram pensados segundo o paradigma da programação orientada aos objectos (a persistência dos mesmos é completamente deixada a cargo do `Hibernate`[1]), e são os suficientes para garantir a flexibilidade necessária. O processamento das regras é feito de

forma célere sem grandes impactos negativos no funcionamento do núcleo com vários pedidos. O mesmo pode ser optimizado com uma organização inteligente das regras.

As limitações da arquitectura prendem-se com o facto da mesma ainda ter sido sujeita a poucos testes. O seu uso para integração de aplicações reais está de momento a decorrer, o que impede uma análise mais profunda das mesmas. No entanto uma das limitações que ainda existe está relacionada com o tratamento de erros, neste ponto do desenvolvimento do IR, se acontecer algum erro no processo a aplicação não tem conhecimento do mesmo não podendo fazer nova invocação.

Sendo uma arquitectura ainda em amadurecimento existe ainda margem de progressão, especialmente no que diz respeito a testes reais de funcionamento da aplicação sobre situações de sobrecarga, definição de *workflow* mais complexo e de situações de interligação de aplicações menos vulgares.

Referências

1. Bauer, C., King, G.: Hibernate in Action. Manning Publications Co. (2004)
2. Johnson, R.: Expert one-on-one J2EE Design and Development. Wrox Press (2002)
3. Johnson, R., Hoeller, J.: Expert one-on-one J2EE Development whithout EJB. Wrox Press (2004)
4. Recomendação W3C: *SOAP Version 1.2 Part 0: Primer*, Junho 2003.
5. Recomendação W3C: *SOAP Version 1.2 Part 1: Messaging Framework*, Junho 2003.
6. Recomendação W3C: *SOAP Version 1.2 Part 2: Adjuncts*, Junho 2003.
7. *W3C Draft: SOAP Version 1.2 Usage Scenarios*, Junho 2002.

Uma Arquitectura Web para Serviços Web

Sérgio Nunes[1] e Gabriel David[2]

[1] FLUP
[2] FEUP/INESC-Porto
Universidade do Porto
Porto, Portugal

Resumo A evolução dos Serviços Web foi, nos últimos tempos, controlada pelas principais empresas da indústria dos computadores. O conceito de Serviço Web (SW) está associado à disponibilização de funcionalidades através de interfaces programáveis via Internet. Na definição apresentada pelo W3C, é referido o uso de URL, HTTP e XML em "conjunto com outras normas relacionadas com a Web". No entanto, apesar do recurso às designadas tecnologias Web, o estilo arquitectural proposto para os SW não segue aquele desenvolvido para a WWW.
Neste artigo, evidenciam-se estas diferenças e descreve-se uma nova abordagem para o desenho de SW, com base no estilo arquitectural REST. Este estilo está na base do desenho da WWW e apresenta qualidades importantes para sistemas distribuídos de grande dimensão. No final, é apresentado um exemplo concreto de um SW, desenvolvido com base nestes conceitos.

1 Introdução

Com o desenvolvimento da Internet e das tecnologias paralelas, surgiram novas abordagens ao problema da interoperabilidade entre sistemas de informação. A ubiquidade da "rede das redes", "indiferente"a fronteiras empresariais ou tecnológicas, surge como uma alternativa para a camada de comunicação. Por outro lado, a evolução tecnológica e a adopção generalizada da eXtensible Markup Language (**XML**), como linguagem de estruturação de documentos, veio permitir o desenvolvimento de novas arquitecturas, actualmente designadas de Serviços Web (**SW**).

Os Serviços Web representam uma tecnologia emergente que tem sido alvo de atenção por parte de toda a indústria dos computadores [1]. Empresas como a Microsoft, IBM e Sun têm apostado e investido neste conceito. As primeiras iniciativas remontam a 1998, com o trabalho desenvolvido pela Microsoft em parceria com a Userland Software, que resultou numa primeira publicação do protocolo SOAP [2]. A partir de 2001, o desenvolvimento de especificações passou a ser realizado em consórcios empresariais, como o W3C e a OASIS.

De uma forma simplificada, os Serviços Web procuram facilitar a integração de aplicações distribuídas utilizando XML sobre a World Wide Web (**Web**). No entanto, de um ponto de vista prático, as estratégias para a implementação deste

conceito divergem, resultando nalguma ambiguidade a este nível. Não existe uma noção clara do que é a implementação de um Serviço Web. Implementações com diferenças importantes ao nível da arquitectura e das tecnologias utilizadas têm sido catalogadas como "Serviços Web".

Um outro aspecto que importa referir é o facto de, associado ao trabalho desenvolvido pelos consórcios empresariais, terem sido produzidas inúmeras especificações. Em finais de 2004, existem mais de 30 especificações (p.e.: WS-Federation, WSDL, BPEL4WS, WS-I Basic Profile) relativas à visão dos Serviços Web proposta pelos principais consórcios! Este aspecto tem contribuído para um maior afastamento entre os programadores, que desenvolvem as implementações reais, e as especificações propostas pelas grandes empresas.

Neste artigo, é feita uma comparação entre a visão tradicional dos Serviços Web e uma outra que tem por base o modelo arquitectural REST. É feita uma comparação em termos da arquitectura das duas abordagens. Uma arquitectura deste tipo determina como os elementos do sistema são identificados e distribuídos, como interagem por forma a constituir um sistema, qual a granularidade da comunicação necessária para interacção e quais os protocolos de interface utilizados para comunicação [3].

2 Serviços Web

A Figura 1 representa o modelo apresentado pelo W3C [4] relativo ao processo geral para estabelecimento de um Serviço Web.

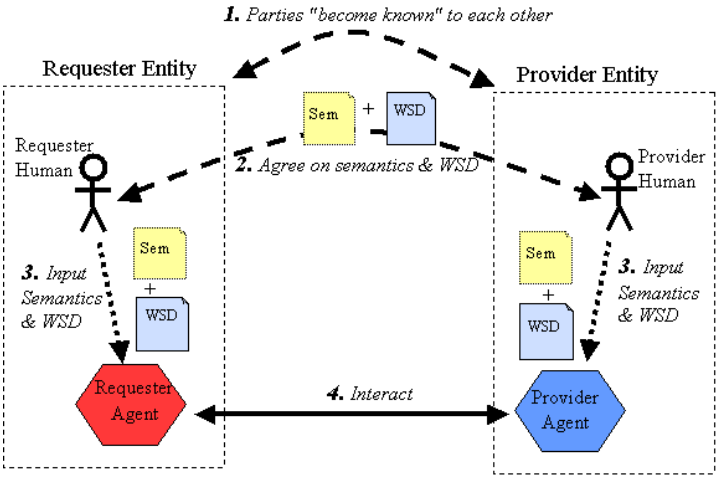


Figura 1. Processo geral para estabelecer um Serviço Web [4]

É possível identificar quatro fases genéricas necessárias [4]: (1) o cliente e o servidor tornam-se conhecidos um do outro (ou pelo menos um conhece o outro); (2) o cliente e o servidor estabelecem um acordo em relação à descrição e semântica do serviço que vai gerir a interacção entre os agentes; (3) são desenvolvidas implementações de acordo com a descrição e semântica acordadas; (4) o cliente e o servidor trocam mensagens que concretizam a interacção entre as partes.

Considerando os objectivos deste trabalho, observa-se com maior detalhe a fase de interacção. De acordo com a visão tradicional para os Serviços Web, a fase de interacção é implementada com recurso à troca de mensagens SOAP. SOAP [5] é uma linguagem de anotação, baseada em XML Schema [6], para a descrição de protocolos de comunicação. Permite a definição de interfaces públicas e dos detalhes associados à invocação e resposta. Em particular, permite a definição dos tipos e estruturas de dados a utilizar. O SOAP pode ser definido como um "*protocol framework*".

A Figura 2 apresenta uma mensagem SOAP correspondente à invocação de um procedimento.

```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap=" http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
        <u:getStudentInfo xmlns:u="http://univ.example.com">
            <u:code>123456789</u:code>
        </u:getStudentInfo>
    </soap:Body>
</soap:Envelope>
```

Figura 2. Mensagem SOAP para a invocação do método getStudentInfo com o argumento 123456789

Na Figura 3 apresenta-se a resposta ao pedido. O código aqui apresentado é uma versão simplificada daquele que seria necessário num cenário real.

```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
        <u:getStudentInfoResponse xmlns:u="http://univ.example.com">
            <u:code>123456789</u:code>
            <u:name>João Silva</u:name>
            <u:age>19</u:age>
        </u:getStudentInfoResponse>
    </soap:Body>
</soap:Envelope>
```

Figura 3. Mensagem SOAP de resposta ao pedido apresentado na Figura 2

O SOAP não define, nem está limitado, a um mecanismo de transporte específico. A especificação permite a integração com diversos protocolos para a transferência das mensagens, por exemplo: HTTP, SMTP e MQSeries. Actualmente, a generalidade das implementações utilizam o protocolo HTTP, em particular o método POST, para a troca de mensagens SOAP. Este cenário é ilustrado na Figura 4.

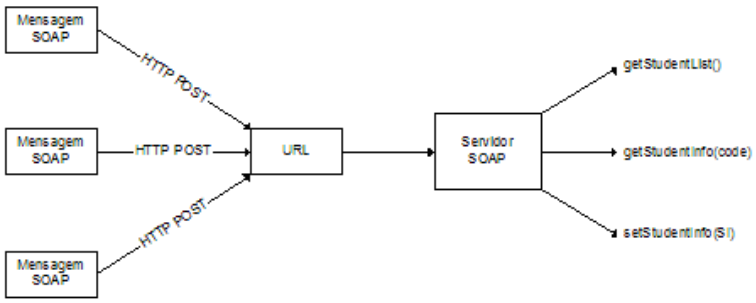


Figura 4. Arquitectura de um Serviço Web tradicional (baseado em [7])

Não é um requisito da especificação a utilização do mesmo URL para o envio de todas as mensagens. No entanto, esta é uma estratégia comum entre os promotores do SOAP. A generalidade dos IDE e das bibliotecas disponíveis geram código segundo esta arquitectura.

3 Representational State Transfer

Representational State Transfer, ou simplesmente REST, é um termo definido por Roy Fielding [8] para descrever um estilo arquitectural de sistemas de informação distribuídos.

A primeira edição do REST, originalmente designada por "HTTP Object Model", foi desenvolvida entre 1994 e 1995, como um meio para transmitir os conceitos que se encontram na base da Web durante o desenvolvimento do protocolo HTTP. Fielding [3] afirma que "*the name "Representational State Transfer"is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user.*".

Resumindo, REST representa um modelo de como a Web deve funcionar. A existência deste modelo permite identificar áreas onde os protocolos existentes falham, comparar soluções alternativas e assegurar que novas extensões não violam os princípios nucleares que contribuíram para o sucesso da Web. O REST

não é uma norma nem uma especificação, mas sim um conjunto de restrições que induzem determinadas propriedades nos sistemas.

Uma descrição detalhada do modelo REST está fora do âmbito deste artigo, pelo que se referem apenas os aspectos mais relevantes para motivar a discussão a propósito dos Serviços Web. Uma apresentação exaustiva do estilo arquitectural REST pode ser encontrada na proposta original publicada por Roy Fielding [8].

O modelo REST utiliza um conjunto de interfaces genéricas para promover interacções sem estado (*stateless*) através da transferência de representações de recursos, em vez de operar directamente sobre esses recursos. O conceito de recurso é a principal abstracção deste modelo. Entre as restrições definidas pelo REST, destacam-se três e refere-se a aplicação no caso concreto da Web:

- **Identificação global.** Todos os recursos são identificados através do mesmo mecanismo global. Independentemente do contexto de aplicação, tecnologia ou implementação concreta, cada recurso disponível na Web é identificado utilizando um URL [9].
- **Interfaces uniformes.** A interacção entre os agentes é feita com recurso ao protocolo HTTP [10], utilizando um conjunto de métodos pré-definidos: GET, POST, PUT e DELETE.
- **Interacções** *stateless***.** O estado concreto de uma aplicação Web é mantido com base no estado de um conjunto de recursos. O servidor conhece o estado dos seus recursos mas não mantém informação sobre as sessões dos clientes.

O caso particular das interfaces uniformes pode ser comparado à abstracção que se verifica na linguagem SQL com as operações SELECT, INSERT, UPDATE e DELETE. O recurso a um conjunto pré-definido de operações, permite que qualquer interveniente na comunicação entenda a operação e aja de acordo. Existe uma semântica estabelecida ao nível da infraestrutura. No caso da Web, intermediários como as *caches* e *proxies*, podem agir de acordo com a operação executada e contribuir para a escalabilidade de toda a infraestrutura, sem afectar a definição da respectiva operação abstracta. Por exemplo, o método GET, idempotente por definição, pode ser servido por *caches* intermédias sem intervenção do servidor responsável pelo recurso. Por outro lado, os métodos POST, PUT e DELETE só podem ser processados pelo servidor uma vez que alteram o estado do recurso.

Assim, quando um agente do sistema encontra um URL ou um método HTTP sabe o que fazer para o processar. O facto de existirem acordos públicos, partilhados por todos os interveniente, contribui de forma positiva para a escalabilidade e heterogeneidade do sistema global.

A Web é o maior sistema de informação distribuído em funcionamento. O REST procura identificar e sistematizar as características arquitecturais que permitiram o crescimento exponencial verificado na última década.

4 Serviços Web segundo o modelo REST

Observando as especificações actuais associadas aos Serviços Web [11] e, em particular, os modelos de programação induzidos pelos IDE [12], é possível iden-

tificar um modelo diferente daquele seguido na implementação da Web. Nesta secção, apresenta-se uma arquitectura alternativa para o desenho de Serviços Web tendo por base os pressupostos estabelecidos com o modelo REST.

Como se viu na Secção 2, segundo a abordagem tradicional ao desenvolvimento de Serviços Web, os recursos estão escondidos atrás de métodos que podem ser invocados. Num Serviço Web compatível com a arquitectura REST os recursos a disponibilizar devem ser expostos e ter uma identificação global.

No cenário utilizado como exemplo, pretende-se disponibilizar informação sobre alunos. Os recursos correspondem a documentos XML com a informação sobre os alunos e são disponibilizados com a estrutura apresentada na Figura 5. Cada URL identifica de forma única a informação sobre um estudante.

```
...
http://univ.example.com/studentInfo/123456787
http://univ.example.com/studentInfo/123456788
http://univ.example.com/studentInfo/123456789
...
```

Figura 5. Recursos expostos de um Serviço Web com uma arquitectura REST

Depois de expostos os recursos, a interacção é realizada utilizando as operações genéricas do protocolo HTTP. Assim, para obter a representação da informação relativa a um estudante, utiliza-se o método GET sobre o URL concreto. Na Figura 6 ilustra-se essa situação.

```
GET http://univ.example.com/studentInfo/123456789
_____________________________________________________

<?xml version="1.0" ?>
<u:StudentInfo xmlns:u="http://univ.example.com">
    <u:code>123456789</u:code>
    <u:name>João Silva</u:name>
    <u:age>19</u:age>
</u:StudentInfo>
```

Figura 6. Pedido da representação de um recurso num Serviço Web com uma arquitectura REST e respectiva resposta

Os restantes métodos do protocolo HTTP permitem: pedir a criação de uma representação para um novo recurso (POST); actualizar a representação de um recurso existente (PUT); apagar um recurso (DELETE). Enquanto que os métodos DELETE e PUT operam directamente sobre um recurso identificado por um URL, o método POST é aplicado sobre um URL disponibilizado pelo servidor para a criação de novos recursos (ex: http://univ.example.com/studentInfo).

Apesar de não serem relevantes para esta discussão, é importante referir que na especificação HTTP são definidos mais métodos para além dos quatro aqui apresentados. De notar que, na navegação Web, apenas os métodos GET e POST são implementados pelos navegadores.

Uma outra característica que advém da adopção do modelo REST é a possibilidade que o cliente tem de navegar entre os recursos. Utilizando a especificação XLink [13] e devido à existência de um mecanismo de identificação global (URL), são estabelecidas ligações entre os recursos disponíveis. É da responsabilidade do cliente navegar de recurso em recurso, reunindo a informação que necessita ou activando os estados que pretende. Desta forma, através da utilização de referências, é possível evitar a repetição de informação e todos os problemas associados a esta opção. Por outro lado, é possível implementar serviços mais sofisticados envolvendo a partilha de recursos entre aplicações diferentes.

Na Figura 7, ilustra-se o resultado de um pedido da lista de estudantes disponíveis. O pedido é enviado utilizando o método GET num URL disponibilizado pelo servidor. Reparar no recurso à especificação XLink para representar as ligações para a representação da informação de cada estudante. Com esta informação, o cliente pode efectuar operações sobre os recursos "descobertos".

```
GET http://univ.example.com/studentInfo
```

```
<?xml version="1.0" ?>
<u:StudentInfos  xmlns:u="http://univ.example.com"
                 xmlns:xlink="http://www.w3.org/1999/xlink">
     <u:StudentInfo id="123456787" xlink:href="http://univ.example.com/StudentInfo/123456787"/>
     <u:StudentInfo id="123456788" xlink:href="http://univ.example.com/StudentInfo/123456788"/>
     <u:StudentInfo id="123456789" xlink:href="http://univ.example.com/StudentInfo/123456789"/>
     ...
</u:StudentInfos>
```

Figura 7. Pedido de uma lista de recursos num Serviço Web com uma arquitectura REST e respectiva resposta

Como se procurou ilustrar, com recurso a tecnologias estabelecidas e aptas a serem utilizadas (URL, HTTP, XML Schema e XLink), é possível desenvolver Serviços Web com uma arquitectura compatível com o modelo REST. Nesta secção foi feita uma apresentação resumida desta abordagem alternativa. Mais informações sobre o desenvolvimento de Serviços Web compatíveis com o modelo REST podem ser encontradas em [14], [15], [16] e [17].

5 Discussão e Análise dos Resultados

Na definição apresentada pelo W3C para Serviços Web [4], refere-se o "[...] recurso a mensagens SOAP, tipicamente transmitidas sobre HTTP e codificadas em XML, em conjunção com outras especificações Web.". No entanto, para além das tecnologias concretas, um aspecto fundamental da Web é a arquitectura que está na base deste sistema distribuído. Durante o desenvolvimento das principais

tecnologias (nomeadamente URL e HTTP), os conceitos que regem a arquitectura base da Web estiveram presentes de forma tácita. Em 2000, Roy Fielding [8] sistematizou e publicou esta arquitectura com a designação REST.

De uma forma resumida, REST representa um modelo de como a Web deve funcionar de forma a garantir os requisitos inicialmente estabelecidos, em particular as propriedades de escalabilidade e heterogeneidade. No entanto, o trabalho desenvolvido pelos principais consórcios e empresas, no âmbito dos Serviços Web, não tem tido em conta as restrições definidas pelo estilo arquitectural REST.

Na realidade, as especificações actuais para os Serviços Web recorrem às principais tecnologias Web de forma preferencial mas opcional, permitindo a integração com outros protocolos, como o SMTP. Uma designação mais fiel da situação real seria a de "*Serviços Internet*".

É também importante referir que, apenas na segunda revisão da especificação SOAP (versão 1.2), foi possível passar a utilizar a linguagem numa forma consistente com a arquitectura REST. A versão 1.1 do SOAP, a especificação nuclear dos Serviços Web, não era compatível com o modelo REST [4].

A estratégia actual representa uma evolução das abordagens já existentes baseadas no conceito de *Remote Procedure Call* (RPC). A principal diferença introduzida pelo REST é a uniformização da interface de comunicação [8]. Ao contrário do que acontece com o SOAP, o modelo REST não permite a definição de interfaces específicas.

Como resultado, no cenário tradicional, temos duas camadas específicas em cada sistema distribuído: (1) protocolo de interacção (SOAP) e (2) representação dos dados (XML). Num modelo REST, a camada de interacção é normalizada. Assim, apenas a representação dos dados é específica em cada implementação. Com a Figura 8 procura-se ilustrar esta diferença.

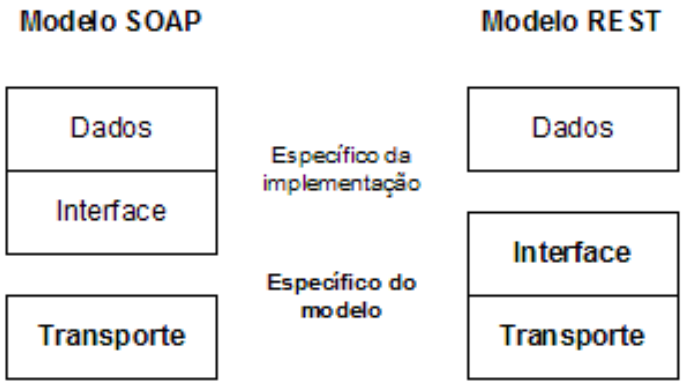


Figura 8. Organização das camadas no modelo SOAP e no modelo REST (adaptado de [20])

No caso do SOAP, as interfaces não são definidas previamente. Para cada implementação é possível definir os modelos de dados e as interfaces a utilizar. Em particular, é possível definir os mecanismos de endereçamento e os métodos disponíveis para invocação. No modelo REST, a interface faz parte da infraestrutura e é estática. Isto é, as implementações concretas apenas diferem na camada de dados.

Esta diferença permite que o SOAP funcione sobre qualquer protocolo, uma vez que os detalhes da invocação estão dentro da mensagem e são independentes da camada inferior. Por outro lado, o modelo REST implica a utilização de um protocolo comum entre todos os intervenientes, incluindo os agentes intermédios (*caches*, *proxies*, *firewalls*). Com esta abordagem, é possível eliminar uma camada de complexidade extra no desenvolvimento das soluções. Os serviços recorrem à interface disponível, neste caso HTTP, evitando a especificação de soluções próprias. Desta forma, reduzem-se as barreiras entre serviços diferentes, aumentando o grau de interoperabilidade.

Por exemplo, numa abordagem SOAP os métodos getStudent, getFaculty, getCourse apenas têm significado semântico num contexto limitado, o dos sistemas que os implementam. Numa abordagem REST, o método GET mantém o mesmo significado apesar de poder ser aplicado a diferentes recursos (estudantes, docentes, disciplinas). A interoperabilidade com outros sistemas está presente por natureza.

Um outro aspecto importante para esta discussão tem a ver com a complexidade das especificações propostas pelos consórcios empresariais. Este facto tem afastado as pequenas empresas e programadores da designada via tradicional. Em relação a este aspecto, a abordagem REST apresenta-se como uma solução mais simples. Por um lado, promove tecnologias testadas e disponíveis a baixo custo, por outro lado baseia-se num paradigma bem conhecido. A propósito desta questão, um aspecto importante referido por Tim O'Reilly [18], é o facto da Amazon disponibilizar serviços Web em ambos os estilos (REST e SOAP) [19] e 85% da utilização ser através da interface REST.

6 Conclusões

Mantendo-se fieis à arquitectura REST, as tecnologias Web (URI e HTTP) foram capazes de se adaptar ao crescimento exponencial que se verificou durante os últimos 10 anos de existência da Web. Outras abordagens para o suporte de sistemas distribuídos, bem sucedidas na integração em redes locais (CORBA, DCOM, RMI), tiveram um sucesso muito limitado quando transpostas para sistemas de grande dimensão, à escala da Internet.

Com este trabalho, procurou-se contribuir para uma reflexão crítica a propósito das recentes estratégias no âmbito da interoperabilidade entre sistemas distribuídos. O paradigma baseado no modelo REST tem por base uma arquitectura e um conjunto de tecnologias que estão na origem do sucesso de um dos maiores sistemas distribuídos desenvolvidos. Confrontando este modelo com aquele que tem vindo a ser proposto pelas grandes empresas e consórcios, as diferenças são

evidentes. No entanto, é ainda cedo para afirmar a superioridade de uma ou outra abordagem para a implementação de Serviços Web.

Neste contexto, importa referir que as duas abordagens não são exclusivas. Desenvolvimentos recentes apontam para uma aproximação entre as duas. Por um lado, recorrer à utilização dos paradigmas introduzidos com o modelo REST, em particular as interfaces genéricas e a exposição dos recursos. Por outro lado, aproveitar o trabalho desenvolvido para o SOAP nas áreas da segurança e orquestração de serviços, entre outras.

Referências

1. Wong, W., Kane, M., Ricciuti, M.: The new buzz - "Web services"try to rise above high-tech din CNET News.Com. (2000) `http://news.com.com/2009-1017-275484.html?legacy=cnet&tag=tp_pr` [02-12-2004]
2. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. W3C Note, World Wide Web Consortium. (2000) `http://www.w3.org/TR/SOAP` [02-12-2004]
3. Fielding, R. T., Taylor, R. N.: Principle Design of the Modern Web Architecture ACM Transactions on Internet Technology. 2:2 (2002) 115–150
4. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture W3C Working Group Note, World Wide Web Consortium. (2004) `http://www.w3.org/TR/ws-arch/` [24-11-2004]
5. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F.: SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation, World Wide Web Consortium. (2003) `http://www.w3.org/TR/soap12-part1/` [02-12-2004]
6. Fallside, D. C.: XML Schema Part 0: Primer W3C Recommendation, World Wide Web Consortium. (2001) `http://www.w3.org/TR/xmlschema-0/` [16-05-2003]
7. Costello, R. L.: REST (Representational State Transfer). (2002) `http://www.xfront.com/REST.html` [02-12-2004]
8. Fielding, R. T.: Architectural styles and the design of networked-based software architectures Dissertação de Doutoramento Dept. of Information and Computer Science, University of California, Irvine (2000)
9. Berners-Lee, T., Fielding, R. T., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax RFC 2396, Internet Engineering Task Force. (1998) `http://www.ietf.org/rfc/rfc2396.txt` [26-05-2004]
10. Fielding, R. T., Gettys, J., Mogul, J. C., Nielsen, H. F., Masinter, L., Leach, P. J., Berners-Lee, T.: Hypertext Transfer Protocol - HTTP/1.1 RCF 2616, Internet Engineering Task Force. (1999) `http://www.ietf.org/rfc/rfc2616.txt` [26-05-2004]
11. Cabrera, L. F., Kurt, C., Box, D.: An Introduction to the Web Services Architecture and Its Specifications White Paper, Microsoft Developer Network. (2004) `http://msdn.microsoft.com/library/en-us/dnwebsrv/html/introwsa.asp` [02-12-2004]
12. Lopes, C. J. F., Ramalho, J. C.: Web Services: Metodologias de Desenvolvimento Actas da XATA 2004 - XML, Aplicações e Tecnologias Associadas. Porto, Portugal. (2004)
13. DeRose, S., Maler, E., Orchard, D.: XML Linking Language (XLink) Version 1.0 W3C Recommendation, World Wide Web Consortium. (2001) `http://www.w3.org/TR/xlink/` [02-12-2004]

14. Prescod, P.: Second Generation Web Services XML.com (2002) `http://www.xml.com/pub/a/ws/2002/02/06/rest.html` [20-11-2004]
15. Prescod, P.: REST and the Real World XML.com (2002) `http://www.xml.com/pub/a/ws/2002/02/20/rest.html` [20-11-2004]
16. Baker, M.: RESTWiki. (2004) `http://rest.blueoxen.net` [20-11-2004]
17. Prescod, P.: Paul's REST Resources. (2004) `http://www.prescod.net/rest/` [02-12-2004]
18. O'Reilly, T.: REST vs. SOAP at Amazon. (2003) `http://www.oreillynet.com/pub/wlg/3005` [02-12-2004]
19. Amazon.com: Amazon.com Web Services (2004) `http://www.amazon.com/webservices` [02-12-2004]
20. Baker, M.: Protocol independence. (2004) `http://www.markbaker.ca/2002/09/Blog/2004/10/29#2004-10-protocols` [20-01-2005]

Production Scheduling Concepts Modeling through XML

Leonilde Varela[1], Joaquim Aparício[2], Carmo Silva[3]

Dept. Production Systems, University of Minho,
[1] Azurém Campus, 4800-058 Guimarães, Portugal
[3] Gualtar Campus, 4710-057 Braga, Portugal
{Leonilde, Scarmo}@dps.uminho.pt
http://www.dps.uminho.pt/pessoais
[2] Dept. Computer Science, New University of Lisbon,
Quinta da Torre, 2829-516, Monte de Caparica, Portugal
Jna@di.fct.unl.pt
http://www.di.fct.unl.pt/

Abstract. Production scheduling objectives can be better satisfied through the execution of the most suitable methods available for the resolution of each particular problem. In this paper we make a contribution to this through scheduling knowledge modelling and communication through XML (eXtensible Markup Language) and related technologies based on a production scheduling problems classification nomenclature developed. We aim at standard representation of scheduling problems, methods and related concepts, including standard access to scheduling methods implementations through the Internet. This kind of data modelling is the support for a web-based decision support system for solving such problems. Such a system has the purpose of allowing different users to easily access, share and disseminate knowledge about production scheduling through the Internet.

1 Introduction

Scheduling problems are a part of a much broader class of combinatorial optimisation problems. Good production schedules strongly contribute to the increase of a companies' success. The objective in a production scheduling problem consists on finding a solution for allocating a set of machines to process a set of jobs in a given time horizon in order to obtain good system performance based on single or multi criteria.

Scheduling problems are characterized by several constrains, namely job ready times, due dates, penalties for delays, and many other constraints related to task processing and production resources. These include job operations precedence constraints and sometimes precedence constraints among jobs. Typical examples of performance criteria include the minimization of maximum job flow and maximum lateness of jobs in the production system.

The wide spectrum of scheduling problems calls for a standardized nomenclature for problem representation, based on attributes that enable defining problem classes to which real problem instances belong. Such a nomenclature is important not only for problem representation but also for problem resolution. In our work, we developed a nomenclature for such purpose based on work presented in some important and well-known literature sources on scheduling [3,4,13,15,17].

The scheduling objectives can be better satisfied through the execution of the most suitable methods available for the resolution of each particular problem. In this paper we make a contribution to this through scheduling knowledge modelling and communication through XML (eXtensible Markup Language) and related technologies based on the classification nomenclature mentioned. We aim at standard representation of scheduling problems, methods and other scheduling concepts, including standard access to scheduling methods implementations through the Internet. This kind of data modelling is the support for a web-based decision support system for solving such problems. Such a system has the purpose of allowing different users to easily access, share and disseminate knowledge about production scheduling through the Internet.

This paper is organized as follows. The next section briefly describes the nature of production scheduling problems and the classification model used. Section 3 presents the XML-based modeling of the main production scheduling concepts, which include the problems, the methods and the results specification. Section 4 briefly refers to the remote methods invocation process, illustrated through the XML-RPC communication protocol, and finally, in section 5 we present the conclusion.

2 Production Scheduling

Production Scheduling problems have a set of characteristics that need specification. These characteristics can be organized into classes. One such class of factors, which we call the α class, characterizes the production environment, i.e. the system and machines available. Another one, the β class, deals mainly with characterization of jobs and processing requirements. Some important processing requirements that frequently have to be taken into account for processing jobs have to do with resources other than machines, i.e. operators, tools, handling devices buffers among others. These must also be specified and are considered in our nomenclature for problem definition [19,20]. The third, the γ class, specifies the performance measure or evaluation criterion. Typical examples of such measures are the maximum flow time, the makespan and the mean and maximum lateness of jobs.

The classification nomenclature is used as a basis for the XML-based problem specification model underlying this work. It includes a wide range of problem classification factors, which may be combined in different ways, resulting in many distinct scheduling problem classes. For example, class F2|n|Cmax, refers to the problem of processing n jobs on a pure flow shop, with two machines, always available, with jobs being ready at time zero for processing. The performance measure consists on minimizing the maximum completion time or makespan (Cmax).

3 XML Modeling

The eXtensible Markup Language (XML) has been having a wide acceptance and caused a great impact on Internet real world applications, since its release by the World Wide Web Consortium (W3C) in 1998 [16]. This specification language enables describing structures and meanings of data, with a simple syntax, and is an ideal candidate format for exchanging and processing data through the Internet. Other advantages of XML based representation are its openness, simplicity and scalability [2,6,11,16,18]. These were some of the most important reasons for having chosen it for the development of our web system. For details about XML and related technologies (DTD, XSL, XML Schemas, Namespaces, etc.) see, for example, Ceponkus and Hoodbhoy [6] or Ramalho and Henriques [18], among many other references [2,5,6,16].

3.1 Problems

Following the lines already presented in [19,20] problems are classified and modelled by a DTD (Document Type Definition) (see code below). Elements introduced in the referred DTD are expected to become part of a common namespace. Elements on the problem DTD file precisely characterize a scheduling problem class, meaning that from the point of view of the system and interaction with the system a problem must be described according to that grammar.

```
Elements and attributes declaration -->
<!ELEMENT problems (problem+)>
<!ELEMENT problem (alpha?, beta?, gamma)>
<!ATTLIST problem
 problem_class CDATA #REQUIRED
 preferred (true | false) "false">
<!-- Alpha elements -->
<!ELEMENT alpha (alpha1?,alpha2?)>
<!-- Alpha1 -->
<!ELEMENT alpha1 EMPTY>
<!ATTLIST alpha1 system_type (0 | P | PMPM |…) "0">
<!-- Alpha2 -->
<!ELEMENT alpha2 EMPTY>
<!ATTLIST alpha2
 value (0 | m) "0"
   machines_quantity CDATA #REQUIRED>
<!-- Beta elements -->
<!ELEMENT beta (beta1?, beta2?, ... , beta12?)>
…
<!-- Beta4 -->
<!ELEMENT beta4 (job_time*)>
<!ATTLIST beta4 value (0 | pj) "0">
<!ELEMENT job_time EMPTY>
<!ATTLIST job_time
   name CDATA #REQUIRED
   time CDATA #REQUIRED>
```

```
<!-- Beta5 -->
<!ELEMENT beta5 (job_date*)>
<!ATTLIST beta5 value (0 | dj) "0">
<!ELEMENT job_date EMPTY>
<!ATTLIST job_date
   name CDATA #REQUIRED
   date CDATA #REQUIRED>
…
<!-- Beta7 -->
<!ELEMENT beta7 EMPTY>
<!ATTLIST beta7 value (0 | nj) "0"
 jobs_quantity CDATA #REQUIRED>
<!—Beta8 -->
<!ELEMENT beta8 (job_priority*)>
<!ATTLIST beta8 value (0 | dj) "0">
<!ELEMENT job_priority EMPTY>
<!ATTLIST job_priority
   name CDATA #REQUIRED
   priority CDATA #REQUIRED>
…
<!-- Gamma element -->
<!ELEMENT gamma EMPTY>
<!ATTLIST gamma measure (Cmax | SumCj | Cmean | SumWjCj
| Fmax | SumFj | Fmean | SumWjFj | Lmax | SumLj |  Lmean
| SumWjLj | Tmax | SumTj | Tmean | SumWjTj | Emax |
SumEj...) “Cmax”>
```

From the code above, we can read that in order to define a problem we must optionally defined the alpha, beta and the gamma factors since there is always a corresponding factor value defined by default.

Our sample problem is known to belong to class F2|n|Cmax, and can be defined by an XML file as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE problems SYSTEM "problem.dtd">
<problems>
    <problem problem_class="F2|n|Cmax">
        <alpha>
            <alpha1 system_type ="F"/>
            <alpha2 machines_quantity="2"/>
        </alpha>
        <beta>
          …
         <beta4 value="0">
      <job_time name="J1" time="10"/>
        </beta4>
        …
        <beta7 value="n" jobs_quantity="4"/>
        …
        </beta>
        <gamma measure="Cmax"/>
    </problem>
</problems>
```

3.2 Methods

In the Internet many implementations may exist for a same method. From the point of view of the system two implementations of a given method may differ if, for example, they differ only on its outputs. Unfortunately not all implementations work in the same way and in order for the system to use those implementations in a programmatic way, they must also be described within the system. This description must include (among other things) the actual call to the running method. Implementations are described in a DTD file, and an example for the implementation for the Johnson's rule is given below.

```
<!ELEMENT methods (method)*>
<!ELEMENT
method(id,name,url?,problem_class,method_class?,referen
ce,complexity?,protocol?,signature?,gantt?)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT problem_class (#PCDATA)>
<!ELEMENT method_class (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT complexity (#PCDATA)>
<!ELEMENT protocol (#PCDATA)>
<!ELEMENT signature (input,output)>
<!ELEMENT input (param | array | matrix)+>
<!ELEMENT param (#PCDATA)>
<!ATTLIST  param  name  CDATA  #REQUIRED  type  CDATA
#REQUIRED control (submit) #IMPLIED>
<!ELEMENT array (item+)>
<!ATTLIST  array  from  CDATA  #FIXED  "1"  to  CDATA
#REQUIRED control (submit) #IMPLIED>
<!ELEMENT item (#PCDATA)>
<!ATTLIST  item  name  CDATA  #REQUIRED  type  CDATA
#REQUIRED>
<!ELEMENT matrix (item+)>
<!ATTLIST matrix lines CDATA #REQUIRED columns CDATA
#REQUIRED control (submit) #IMPLIED>
<!ELEMENT output (param | array | matrix)+>
<!ELEMENT gantt (#PCDATA)>
```

Many scheduling methods may be more or less adequate to solve a given class of problems. In the methods knowledge base the system records the scheduling method(s) that can be used for solving a certain problem class. Searching for the adequate methods, for a given problem, is performed by matching the problem details with the methods' characteristics, a process performed by the built-in prolog engine. The methods can be available in the methods' knowledge base but they can also be found in any other sites, made available and accessible through the Internet.

The example code below shows a sample of the XML document about scheduling methods. It illustrates the information related to the implementation of the Johnson's Rule [1,3,4,8,9,10,12,15,17] for solving problem instances belonging to the

F2|n|Cmax class described in section 2. This document is validated against the corresponding DTD, previously shown, before being put in the corresponding knowledge base component.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE methods SYSTEM “methods.dtd”>
<methods>
 <method>

 <id>32</id><name>Johnson</name><url>http://localhost:60
02/RPC2</url>
   <prob_class>F2|n|Cmax</prob_class>
   <method_class>Sequencing Rule</method_class>
   <reference>Johnson,1954</reference>
   <complexity>Maximal         Polynomially         Sol
able</complexity>
<protocol>XML-RPC</protocol>
   <signature>
      <input>
         <param name=”n” type=”integer”/>
         <param     name=”m”     type=”integer”     con-
trol=”submit”/>
         <matrix lines=”m” columns=”n”>
            <item     name=”job”     type=”string”/><item
name=”machine” type=”string”/>
            <item name=”p” type=”double”/>
         </matrix>
      </input>
      <output>
         <param name=”Cmax” type=”double”/>
         <param name=”sequence” type=”string”/>
         …
         <matrix lines=”m” columns=”n”>
            <item name=”start” type=”double”/>
            <item name=”finish” type=”double”/>
            …
         </matrix>
      </output>
   </signature>
   <gantt>yes</gantt>
 </method>…
</methods>
```

The inputs include the definition of a parameter n, for the number of jobs to be processed, a parameter m, for the number of machines, and a set of three items organized as a matrix structure, which represent the job name, the machine name and the processing time p of each job on each machine. There is also the definition for the method’s output following the same lines. After a method definition has been inserted in the corresponding KB it becomes immediately accessible to any further information retrieval. For the example given, after the insertion of the Johnson’s method definition any search for methods that is a match for the F2|n|Cmax problem class will include this method in the search results. The methods’ definitions are also used

in the automatic generation of interfaces for methods invocation and corresponding inputs insertion and subsequently outputs presentation.

3.3 Results

The production scheduling problems' results may be defined in many different ways. Gantt charts are very typical and useful way or expressing these problem results.
Gantt charts are time-based diagrams, which can also automatically generated by the system, given the outputs provided by methods' invocation. This is easily achieved because the methods' output data is expressed in XML documents, enabling an easy way of outputs conversion into different desired problem results presentation forms, namely Gantt charts.
The code below illustrates the DTD for specifying the problem results through such Gantt charts.

```
<!ELEMENT gantt (problem*)>
<!ELEMENT problem (job+)>
<!ATTLIST problem
        id CDATA #REQUIRED
        class CDATA #REQUIRED>
<!ELEMENT job (machine+)>
<!ATTLIST job
        id CDATA #REQUIRED
        name CDATA #REQUIRED>
<!ELEMENT machine (start+,finish+)>
<!ATTLIST machine
        id CDATA #REQUIRED
        name CDATA #REQUIRED>
<!ELEMENT start (#PCDATA)>
<!ELEMENT finish (#PCDATA)>
```

A possible corresponding XML example, for representing a given problem instance is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gantt SYSTEM "gantt.dtd">
<gantt>
  <problems>
    <problem id=”0010” class="F3|n|Cmax">
        <results>
            <job id="J1" name=”Job1”/>
              <machine id="M1" name=”Machine1”/>
              <start>0</start>
              <finish>3</finish>
            </job>…
          </results>
    </problem>
  </problems>
</gantt>
```

These time-based charts continue to be largely used due to its expressive power, enabling a very easy way to comparing results obtained from the invocation of several different implemented methods available. Other alternatives for displaying those outputs are possible, including direct outputs presentation through XML documents.

4 Methods Invocation

The main purpose of this work consists on trying to improve the resolution of production scheduling problems. Therefore, we decided to develop a web system, based on XML modeling and related technologies.
The main element of the web system architecture is an interface component for introduction, validation, and transformation of production scheduling data. This interface is mainly controlled by DTD and XSL (eXtensible Stylesheet Language) documents stored in a knowledge base. The scheduling information is also stored in XML documents and these documents are verified using DTDs, before being put in the corresponding XML knowledge base. The XML and related documents may either be located on the server or on the client side. In this work the documents are stored on the server (e.g. XML, DTD, XSL and other documents) in order to achieve easy and efficient data transferring.
The system is being implemented as a web service and allows the execution of either local or remote scheduling methods, namely through the XML-RPC protocol.
The term Web Services has emerged as a general category for loosely coupled, dynamically connected web-based services and are a set of tools that let us build distributed applications on top of existing web infrastructures.
These services use XML to encode both the message wrapper and the content of the message body. As a result, the integration is completely independent of operating system, language or other middleware product used by each component participating in the service. The only fundamental requirement is that each component has the ability to process XML documents and that each node connected in a distributed system supports HTTP as a default transport layer.
These Web Services are most commonly used to invoke remote application services or methods using a Remote Procedure Call (RPC) interaction implemented using only XML messages [5].
The XML-RPC protocol consists on a possible communication protocol that can be used for remote methods invocation and can be defined as the sequence and structure of requests and responses required to invoke communications on a remote machine. Several other protocols that could also be used exist, namely SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery, and Integration of business for the web), WSDL (Web Services Description Language), or other well known, like CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invocation) or DCOM (Distributed Component Object Model). The XML-RPC protocol is one possible approach that makes it easy for computers to call procedures on other computers [14] and is being used in this work for illustrating the remote scheduling methods invocation process.

The extensible markup language provides a vocabulary for describing remote procedure calls, which are then transmitted between computers using the Hyper Text Transfer Protocol (HTTP).
XML-RPC clients make procedure requests of XML-RPC servers, which return results to the XML-RPC clients. XML-RPC clients use the same HTTP facilities as web browser clients, and XML-RPC servers use the same HTTP facilities as web servers.
XML-RPC requires a minimal number of HTTP headers to be sent along with the XML method request. The code below shows an example that joins the headers and XML payload to form a complete XML-RPC request for solving a given problem belonging to the F2|n|Cmax problem class.

```
POST /rpchandler HTTP/1.0
User-Agent: AcmeXMLRPC/1.0
Host:localhost:5001
Content-Type: text/xml
Content-Length: 832
<?xml version="1.0"?>
<methodCall>
<methodName>JohnsonRule</methodName>
<params>
  <param><value><int>4</int></value></param>
  <param><value><int>2</int></value></param>
  <param>
   <value>
    <array>
      <data>
         <value><string>J1</string></value>
         <value><string>M1</string></value>
         <value><double>3</double></value>
         …
      </data>
    </array>
   </value>
  </param>
 </params>
</methodCall>
```

Upon receiving an XML-RPC request, an XML-RPC server must deliver a response to the client. The response may take one of two forms: the result of processing the method or a fault report, indicating that something has gone wrong in handling the request from the client. As with an XML-RPC request, the response consists of HTTP headers and an XML payload.
The code below shows a complete response from an XML-RPC server, including both the HTTP headers and the XML payload.

```
HTTP/1.0 927 OK
Date: Fri, 25 Oct 2002 07:38:05 GMT
Server: MyCustomXMLRPCserver
Connection: close
Content-Type: text/xml
Content-Length: 868
```

```
<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value><string>L1,L3,L4,L2</string></value>
  </param>
  <param><value><double>39</double></value></param>
  <param>
   <value>
    <array>
     <data>
        <value><string>L1</string></value>
        <value><string>M1</string></value>
        <value><double>0</double></value>
        <value><double>3</double></value>
        …
      </data>
     </array>
    </value>
   </param>
 </params>
</methodResponse>
```

The response is provided to a call to the method JohnsonRule, which returns a solution to a F2|n|Cmax problem.

By using the XML-RPC protocol we are able to invoke scheduling methods implemented on different programming languages. Moreover, these methods, local or remotely available, may be running on different platforms.
As referred previously, our web scheduling system includes a Knowledge Base component that encompasses all the knowledge and modules necessary for remote methods invocation and for performing a set of other system functionalities. This component is controlled by ASP (Active Server Pages) and corresponding server-side XML-RPC components.
On the other hand, there are also correspondent XML-RPC components for each of the methods servers waiting for RPC requests.
This environment is heterogeneous as servers can use their own technology, i.e. use different implementation languages or/and different operating systems.
More detailed information about the XML-RPC protocol can be obtained from http://www.xmlrpc.com.

5 Conclusion

In production enterprises, it is important nowadays, as a competitive strategy, to explore and use software applications, now becoming available through the Internet and Intranets, for solving scheduling problems, which can be achieved in an easy way by using web service technology.
This work is based on an XML-based specification framework for production scheduling concepts modeling, which is used as specification framework for production

scheduling decision-making through a web service. Some of the important functions include the ability to represent scheduling problems and the identification of appropriate available methods for solving them.
The XML-based data modeling is used in order to make possible flexible communication among different scheduling applications. This modeling and specification contributes to the improvement of the scheduling process, by allowing an easy selection of several alternative methods available for problems solving, as well as an easy maintenance of the knowledge base supporting the scheduling service. This is achieved by providing a user-friendly way of new knowledge insertion. This knowledge primarily includes scheduling problems and solving methods as well as its implementations, available through the Internet. These may be implemented on different programming languages and running on different platforms. Such implementations are easily accessible through the web service referred in this paper for solving scheduling problems, through the invocation of local or remotely available scheduling methods. The system allows comparing different solutions obtained by running different methods for a same scheduling problem, and to choose the best solution found to solve the problem, according to a given performance measure to be reached.
The XML based specification can be generated and visualized by computers in appropriate and different ways. An important issue is that the data representation model is general, accommodating a large variety of scheduling problems, which may occur in different types of production environments.

References

1. Artiba, A., Elmaghraby, S.: The Planning and Scheduling of Production Systems. Chapman & Hall, UK (1997).
2. Abiteboul, S., et al.: Data on the Web - From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, USA (2000).
3. Blazewicz, J., et al.: Scheduling Computer and Manufacturing Processes. Springer-Verlag, Germany (1996).
4. Brucker, P.: Scheduling Algorithms. Springer-Verlag, Germany (1995).
5. Carlson, D.: Modeling XML Applications with UML – Practical e-Business Applications. Addison-Wesley, USA (2001).
6. Ceponkus, A., Hoodbhoy, F.: Applied XML. Wiley Computer Publishing, USA (1999).
7. Chrétienne, P., et al.: Scheduling Theory and its Applications. John Wiley & Sons Inc., England (1995).
8. Conway, R. W., Maxwell, W. L., Miller, L. W.: Theory of Scheduling. Addison-Wesley Publishing Company, Inc., England (1967).
9. French, S.: Sequencing and Scheduling – An Introduction to Mathematics of the Job-Shop. John Wiley and Sons, Inc. (1982).
10. Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: A survey. In Annals of Discrete Mathematics (1979).
11. Harper, F.: XML Standards and Tools. eXcelon Corporation, USA (2001).
12. Ignall, E., Schrage L.: Application of the Branch-and-Bound Technique to Some Flow-Shop Problems. In: Operations Research Vol. 13 (3) (1965).
13. Jordan, C.: Batching and Scheduling. Springer-Verlag, Germany (1996).

14. Laurent, S., et al.: Programming Web Services with XML-RPC. O'Reilly & Associates, Inc. (2001).
15. Morton, T., Pentico, D.: Heuristic Scheduling Systems. John Wiley & Sons Inc., USA (1993).
16. Pardi, W.: XML - Enabling Next-generation Web Applications. Microsoft Press, USA (1999).
17. Pinedo, M.: Scheduling Theory, Algorithms and Systems. Prentice-Hall Inc., USA (1995).
18. Ramalho, J. C., Henriques, P.: XML & XSL da Teoria à Prática. FCA, Editora de Informática Lda., Portugal (2002).
19. Varela, L., Aparício, J., Silva, S.: An XML Knowledge Base System for Scheduling Problems. In: Proceedings of the Innovative Internet Computing System Conference.: Springer-Verlag in the Lecture Notes in Computer Science series, Kuhlungsborn, Germany (2002) 61-70.
20. Varela, L., Aparício, J., Silva, S.: Scheduling Problems Modeling with XML. In: Proceedings of the 4th International Meeting for Research in Logistics. International Meeting for Research in Logistics, Inc., Lisbon, Portugal (2002) 897-909.

XESB (XML Editor Schema Based)
Um editor para as massas

Ricardo Queirós
c0165013@alunos.dcc.fc.up.pt

José Paulo Leal
zp@ncc.up.pt

Referência XESB:
http://www.dcc.online.pt/~c0165013/

Abstract. Este documento descreve a especificação e implementação duma aplicação Web para edição assistida de documentos XML chamada XESB. O objectivo principal é a criação de um editor destinado a utilizadores sem conhecimentos de XML. A aplicação utiliza técnicas de edição estrutural para garantir a validade do documento durante o processo de edição. A definição de tipos de documentos é feita em XML Schema. Como motivação, este editor poderá vir ser integrado em sistemas Web que requeiram a configuração de documentos em linguagem XML, ou mais genericamente, poderá servir como um serviço interactivo de produção de documentos XML.

1. Introdução

Os documentos XML foram pensados para serem processados automaticamente e poderem também ser lidos por pessoas, mas não foram pensados para poderem ser editados. As regras de boa-formação tornam a interpretação dos documentos não ambígua mas exigem maior rigor por parte de quem os produz. A especificação XML determina regras rígidas para que os documentos possam ser considerados bem-formados que são difíceis de manter por um utilizador humano. Esse rigor pode facilmente ser obtido se os documentos forem gerados por um programa mas exige muita disciplina a uma pessoa que produza um documento XML num editor de texto genérico. Por outro lado as anotações XML tornam os documentos pouco sucintos o que, não sendo um problema para um computador, exige muito mais esforço para uma pessoa. Um documento XML é consideravelmente mais extenso do que um documento escrito numa linguagem convencional [2], já que as anotações devem ser auto-explicativas. Esta verbosidade está associada aos objectivos iniciais do desenho do XML, nomeadamente a facilidade de leitura dos documentos XML por um humano e a não necessidade de concisão das anotações [3]. Existe pois, uma

dificuldade intrinseca à edição de documentos XML, devido quer ao rigor da sua sintaxe quer à verbosidade das suas anotações.

Observando os vários cenários de necessidade de autoria de documentos XML, bem como a dificuldade intrínseca do seu processo de edição, pode-se concluir que é importante existir editores que produzam documentos XML válidos. Actualmente existem vários editores que permitem a criação e edição de documentos XML, desde editores de texto genéricos que extendem as suas capacidades de edição através de *plugins* e que permitem integrar *parsers* que irão conduzir o processo de edição, a editores específicos de XML. A generalidade destes editores exige o conhecimento da norma XML, limitando assim o leque de possíveis utilizadores do editor, e obrigam a uma prévia instalação de software inibindo assim o seu uso em situações de transição onde apenas se quer criar um número limitado de documentos.

Perante este cenário partiu-se para a criação de um novo tipo de editor chamado XESB – XML Editor Schema Based, que se baseia numa aplicação Web (acessível via navegador) e que não requer o conhecimento de qualquer formalismo. O editor utiliza definições de tipo de documentos que guiam todo o processo de edição garantindo que os documentos criados são sempre válidos.

1.1 Requisitos de desenho

Tendo por objectivo o desenvolvimento de um sistema de edição de documentos XML destinado a utilizadores sem conhecimentos do formalismo, foram identificados os seguintes requisitos de desenho:

- **Abstracção do conhecimento da sintaxe:** Abstrair o utilizador do conhecimento da sintaxe através de interfaces WYSIWYG; esta abordagem alarga o leque de potenciais utilizadores, fazendo com que os mesmos não se percam com os detalhes sintácticos das tecnologias inerentes, concentrando-se apenas no seu conteúdo;
- **Validação:** Guiar o processo de edição de um documento através duma definição formal da linguagem, obtendo assim, por construção, documentos válidos;
- **Operações genéricas sobre fragmentos de documentos XML:** Providenciar operações genéricas (copiar, mover, inserir, remover, clonar) sobre fragmentos de documentos XML flexiblizando assim o processo de edição de um documento e promover também operações entre documentos;
- **Procura:** Disponibilizar mecanismos de procura sobre o texto do documento e que beneficiem da natureza estrutural do mesmo;
- **Modelos:** Usar definições de tipo de documento (DTD) ou documentos esquemáticos (XML Schema) como modelos para instanciar documentos. Esta instanciação é automática e configurável. Os documentos gerados irão herdar as características da "classe" à qual foram instanciados;
- **Inferência de esquemas:** Inferir esquemas a partir de documentos instâncias;
- **Transformação:** Gerar interfaces familiares para edição de documentos XML através da transformação dos documentos usando folhas de estilo XSL. Por omissão deverão existir stylesheets genéricas;

- **Vistas:** Suportar várias perspectivas dum documento XML;
- **Exportação:** Providenciar mecanismos de exportações dum documento para vários formatos (PDF, PS, TXT, SVG, etc.).

1.2 Abordagem: aplicação Web e edição estrutural

A Web é um meio universal para partilha e publicação de documentos. Partindo dessa característica de ubiquidade, e derivado dos requisitos de desenho referidos anteriormente, foi desenvolvida uma aplicação Web para edição assistida de documentos XML. A aplicação utiliza técnicas de edição estrutural para garantir a validade do documento durante o processo de edição. A definição de tipos de documentos é feita em XML Schema, podendo-se utilizar DTD's que são convertidos para a linguagem de esquemas definidas pelo W3C. O suporte de DTD's é justificado pela sua preponderância nas aplicações XML [4]. Estes formalismos definem a estrutura e o tipo de conteúdo que os documentos a ele instanciados poderão ter. Este editor poderá vir ser integrado em sistemas Web que requeiram a configuração de documentos em linguagem XML, mais genericamente poderá servir como um serviço interactivo de produção de documentos XML. Seguidamente justificam-se as principais características do editor, nomeadamente, ser uma aplicação web e usar edição estrutural. Os modelos a considerar eram:

- Uma aplicação tradicional instalada no computador do utilizador;
- Uma aplicação baseada num navegador web, usando as suas possibilidades de processamento de documentos e a web apenas para descarregar o código da aplicação;
- Uma aplicação cliente/servidor em que o processamento do documento é mantido do lado do servidor e apenas a visualização e interacção do lado do cliente.

No caso de uma aplicação autónoma é difícil, converter o XML para um formato visual (XHTML) onde fosse possível interagir com o documento. Converter o documento XML em XHTML é a abordagem mais imediata, mas as bibliotecas que suportam a navegação sobre documentos em aplicações autónomas, como a classe JeditorPane do pacote Swing suportam apenas versões antigas do HTML e não suportam JavaScript.

Uma aplicação assente integralmente no navegador iria cingir a portabilidade da aplicação a navegadores com forte suporte para as tecnologias DOM, XML, XSL e JavaScript. Um exemplo interessante desta abordagem, é o editor BitFlux XML Editor [5] , que permite a edição de documentos XML e que é escrito em JavaScript. Este editor está limitado ao navegador Mozilla.

Adoptando o modelo cliente-servidor beneficiámos simultaneamente da capacidade de navegação e interacção do XHTML nos navegadores e da maior capacidade para processamento do lado do servidor. Para esse processamento é utilizado a linguagem JAVA devido ao seu grande suporte para XML.

No XESB o utilizador deve apenas executar operações sobre o documento compatíveis com o documento esquemático associado. Esta ideia é o conceito base da edição estrutural [6] ou edição dirigida pela sintaxe. A ideia central da edição estrutural é condicionar a edição dum texto numa linguagem mediante uma interface gráfica que permita apenas expansões sinctaticamente válidas. Este processo é designado de compilação incremental. Esta baseia-se na existência duma definição formalda linguagem que permite determinar se um texto lhe pertence ou não. Nos editores estruturais clássicos essa definição formal é uma gramática, em geral livre de contexto. Para o caso do XML a definição formal de linguagem é, por exemplo, um DTD ou um XML Schema, complementando as regras básicas de boa formação do XML. Um editor estrutural é um editor de texto especializado para uma dada linguagem e dirigido pela sintaxe da mesma. Quer isso dizer que o editor vai conduzindo o utilizador de acordo com a gramática da linguagem, preenchendo automaticamente quaisquer anotações ou valores fixos de elementos e atributos que se encontrem definidos no esquema e podendo indicar as alternativas disponíveis em cada momento. Assim, o utilizador não se perde com detalhes sintácticos do documento, já que o editor só admite uma estrutura correcta. podendo concentrar-se no conteúdo (semântica) do que está a escrever [7].

A edição estrutural faz mais sentido em documentos XML do que em linguagens de programação. porque nestas últimas a gramática é sucinta e é normalmente bem conhecida por quem a usa. Por isso a edição estrutural nunca chegou a ser muito usada nos ambientes de programação para os quais foi concebida pois torna o processo de edição de programas mais complexo e moroso. Nos documentos XML, o cenário é diferente. O XML têm regras de boa-formação rigorosas, mas os documentos não tem de ser sucintos. Um documento XML é consideravelmente mais extenso, já que as anotações devem ser auto-explicativas, exigindo muito mais esforço de edição se forem escritos num editor de texto convencional. Existindo um DTD ou Schema associado ao documento XML é vantajoso usar edição estrutural pelas seguintes razões:

- Possibilitam a validação estrutural do documento durante a edição;
- Disponibilizam ajuda contextual ao utilizador, prevenindo-o sempre que este incorrer em erro;
- Fornecem a lista de anotações válidas num determinado ponto do documento, evitando a memorização da sintaxe da linguagem a editar;
- Reconhecem e realçam no texto, as anotações de início e fecho dos elementos, os atributos e as entidades, definidos por um DTD ou Schema[1].

1 Podia ser feito recorrendo apenas a expressões regulares

2. Estado de Arte

O XML é um formalismo para documentos de texto anotados. Por esse facto é possível editar um documento XML utilizando um editor de texto, como por exemplo o Notepad. Estes editores possuem apenas operações genéricas de edição, como a inserção, remoção e substituição de caracteres e a sua selecção. Nestes casos a edição de XML é morosa e é necessário conhecer bem o formalismo, já que este tipo de editores não fornece qualquer ajuda em termos da sintaxe do XML, nem possui qualquer tipo de suporte para a integração de DTD's ou XML Schema com vista à validação do documento. A difusão do XML levou a que alguns editores de texto começassem a suportar XML adaptando algumas funcionalidades a este género de documentos. Estas funcionalidades estão dependentes da linguagem a editar. No contexto do XML, salientam-se o realce das anotações usando cor, a capacidade de expandir e colapsar elementos, a indentação, a criação automática de anotações de fecho e navegação entre anotações, a inserção de entidades, entre outras. Esta adaptação surge, muita vezes, na forma de *plugins* que extendem as capacidades do editor de texto genérico. Um exemplo é o editor Emacs através do *plugin* PSGML [8] que permite uma edição de documentos XML mais rápida e com um melhor suporte.
Alguns editores de texto genéricos utilizam expressões regulares para dotar o editor de características importantes na edição de documentos duma linguagem, e podem ser facilmente extendidos para XML. Este mecanismo, no entanto, não resolve questões relacionadas com a boa-formação ou validação de um documento XML. Para essa verificação, é necessário extender estes editores com módulos/*plugins* ou simplesmente aplicações externas, que irão dotar os mesmos, com capacidades extra para a edição assistida de documentos XML. Se o documento XML tiver associada uma especificação da estrutura (DTD ou XML Schema), é necessário haver uma validação estrutural do documento, sendo esta assegurada pelo *parser* validador que está associado ao editor. Nestes editores de texto genéricos e extensíveis, destacam-se o Emacs e o jEdit.
Actualmente, existem já editores específicos para linguagens XML. Esta nova abordagem oferece um conjunto de funcionalidades específicas que permitem uma edição mais rápida e consistente dos documentos.Contudo ao utilizar estes editores, perdem-se as capacidades genéricas dos editores de texto. Eis uma enumeração de algumas das características principais existentes nos editores XML actuais:

- Suporte a transformação de documentos XML(XSLT/XSL-FO) e depuramento na sua execução;
- Suporte DTD, XML Schema, RELAX-NG, etc;
- Integração de ferramentas de controlo de versões e de deployment (CVS e Ant);
- Importação e exportação para BD's relacionais;
- Múltiplas visões do documento;
- Fácil edição de documentos (controlos avançados, ambiente gráfico, etc.)

Neste tipo de editores destacam-se o Authentic da Altova, o xMetal da SoftQuad e o BitFlux da Oscom.

3. Tecnologias usadas no XESB

Na implementação do XESB optou-se por usar standards, API's e pacotes que implementam essas API's, bem como, ferramentas "open-source" e multi-plataforma. Garante-se assim uma maior portabilidade da aplicação.

Foi adoptado o modelo cliente-servidor, em que o cliente é um navegador Web (como o Mozilla, IE, etc) e o servidor é um contentor de servlets/JSP (como o Tomcat, Resin, etc.). A lógica de negócio fica a cargo dos JavaBeans e a camada de apresentação através da mescla das especificações JSP / XHTML apoiadas pela validação JavaScript e a formatação CSS. A escolha do JAVA como linguagem de programação do lado do servidor deve-se à característica de portabilidade intrínseca à linguagem e sobretudo ao facto de haver um bom suporte para XML no Java. O desenvolvimento do trabalho foi feito no ambiente Eclipse. Para o processamento dos documentos XML e a sua associação com uma especificação da estrutura (DTD ou XML Schema) utilizaram-se as seguintes tecnologias:

- Linguagens de transformação e formatação: XSL (XSLT, XPATH e XSL-FO);
- Interfaces aplicacionais Java a documentos XML: JAXP (DOM/SAX, TRAX);
- Pacotes de Processamento de documentos XML: Xerces (Parser XML) , Xalan (Processador XSL) e NekoDTD (conversão DTD para XML Schema).

Em termos de navegadores o suporte é garantido aos navegadores Mozilla (FireFox) a partir da versão 0.8 e Internet Explorer a partir da versão 5.0, o que representa quase 95% dos navegadores em utilização [9].

4. XESB

O XML é uma especificação de documentos anotados que permite definir uma linguagem usando DTD ("Document Type Definition") ou o Schema. Estas definições correspondem à especificação da gramática se quisermos criar uma analogia [10] com as linguagens de programação. O XESB permite a edição de documentos XML tipificados e apesar de suportar as duas variantes – DTD e XML Schema – este último é a base para a compilação incremental. Esta opção deve-se ao facto de uma definição de tipo de documento XML Schema ser em si um documento XML (reutilização de API's para acesso e manipulação); ao suporte para diferentes tipos de dados; ao suporte de namespaces e às possibilidades de desenvolvimento modular que estes têm face aos DTDs.

O XESB é constituído por vários componentes que partilham um repositório de dados comuns contendo informações sobre um conjunto linguagens e documentos associados. A Fig. 1 ilustra uma visão funcional dos módulos constituintes do XESB.

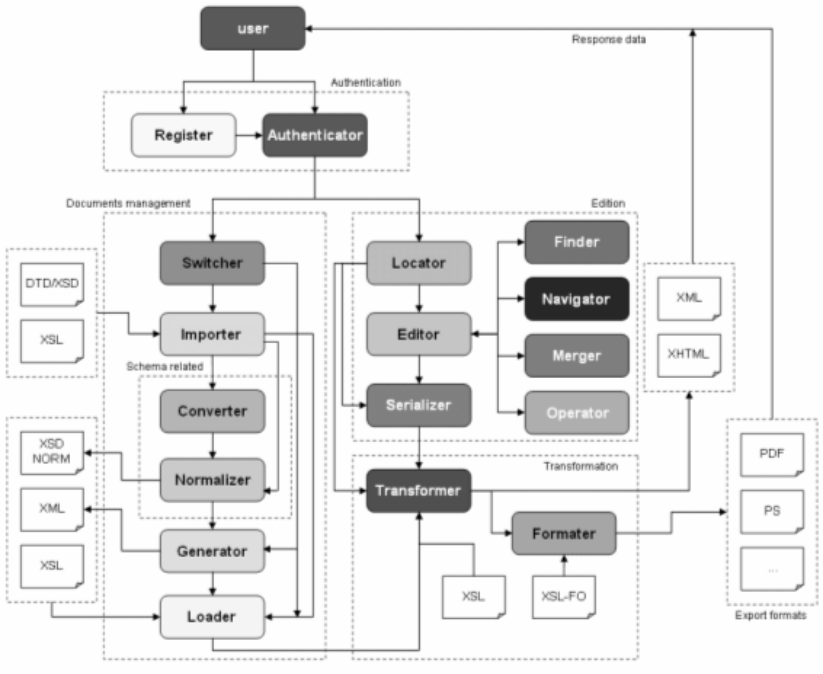


Fig. 1. Arquitectura funcional do sistema XESB

A arquitectura funcional do XESB foca quatro grandes áreas:

- **Autenticação**: Filtra o uso da aplicação a utilizadores com registo, disponibilizando uma área de trabalho no servidor (workspace);

- **Gestão de documentos:** Controla todo o processamento dos documentos desde o carregamento para memória dos documentos XML até à criação de um documento instância. Trata também da normalização do documento esquemático associado e da conversão de DTD's para XML Schema;

- **Edição:** Controla o processo de edição, mapeando todas as acções do utilizador. Localiza o elemento a editar, combina os vários fragmentos dos documentos para resposta ao utilizador. Controla também o processo de procura e navegação no documento, bem como a serialização dos documentos em memória.

- **Transformação:** Define as transformações a serem aplicadas às respostas preparadas pela área de edição. A linguagem objectivo das transformações é o XHTML. Inclui opções para a exportação do documentos para outros formatos (PDF, PS, etc.).

O utilizador do protótipo deverá fazer um registo prévio criando assim uma área de trabalho, constituida por um espaço físico em disco (servidor) para armazenamento dos documentos gerados. Se já possuir registo, é necessária apenas a entrada dos dados de autenticação. Para a criação de documentos instância deverá associar um tipo de documento (existem 3 pré-definidos) e um "template" (se não existir nenhum para determinado tipo é carregado um por omissão). Após estas associações é gerado um esquema normalizado (baseado no tipo de documento seleccionado) e um novo documento instância apenas com a estrutura e pronto para ser editado. No caso de se querer criar um novo tipo de documento é necessária a sua importação prévia para o sistema XESB. Aceitam-se dois formalismos: DTD (sofre uma conversão para XML Schema) ou XML Schema. O processo restante é igual ao referido anteriormente.

Após a selecção de um XML Schema e de uma stylesheet que irá definir uma interface para a edição, o utilizador ao interagir com elementos de formulário vai produzir documentos XML que estão de acordo com a definição formal associada. Esta abordagem permite assim abstrair o utilizador dos detalhes sinctáticos do documento, podendo concentrar-se no seu conteúdo

Após a geração do documento instância, este é extendido por um conjunto de atributos associado a um namespace do sistema XESB: http://www.alunos.dcc.fc.up.pt:8080/xesb/ . Este conjunto tem como missão o controlo de pedidos e respostas que resultam da interacção do utilizador com o editor.

```
<nomeElemento  xesb:id | xesb:gid | xesb:find |
xesb:select | xesb:*>...</nomeElemento>
```

O atributo id identifica um elemento editável. A Fig. 2 demonstra todo o fluxo de informação que é executado aquando da interacção com o utilizador. Após a selecção de um elemento para edição é enviado para o servidor o identificador do elemento a ser editado (1).

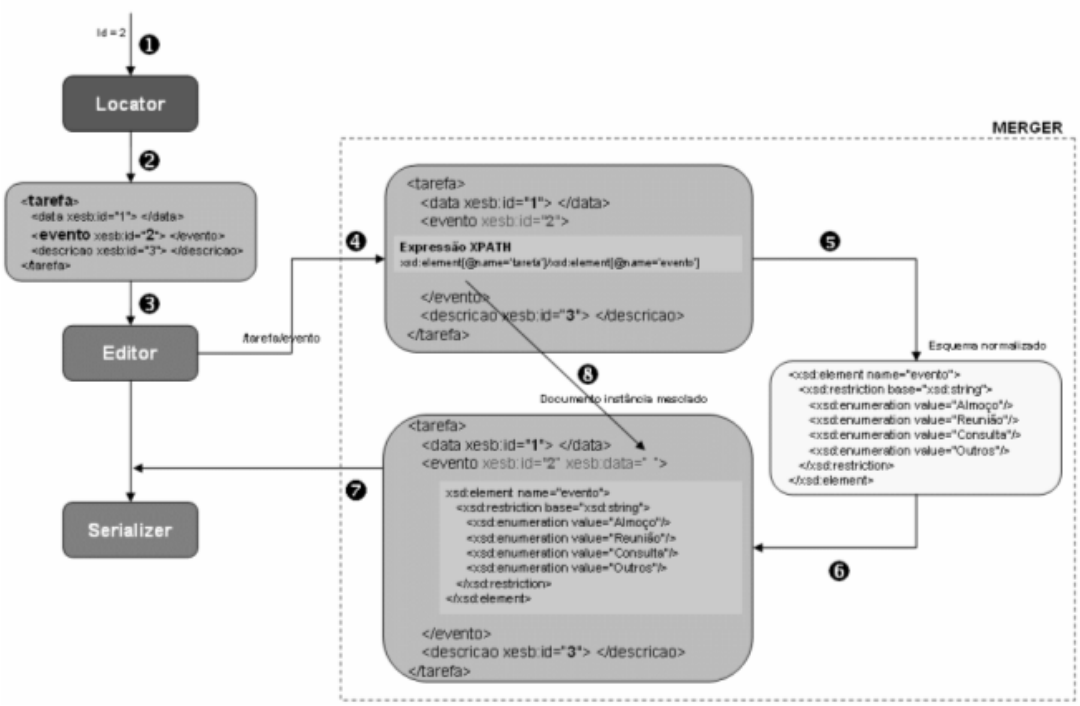


Fig. 2. Pedido de edição de um elemento

O módulo Locator localiza (2) o elemento através do identificador (atributo id) e constroi o caminho absoluto do elemento a editar passando a informação para o Editor (3) que, por sua vez, verifica que se trata de um pedido de edição de um elemento. O Editor precisa de saber que restrições esse elemento tem na definição do esquema e envia para o componente Merger o caminho absoluto do elemento a editar na árvore do documento instância (4). O Merger recebe a informação, mapeia o caminho para uma expressão Xpath que aponta para a declaração do elemento no esquema (5). Note-se que como o esquema está normalizado, sabe-se que toda a informação respeitante ao elemento encontra-se agregada a ele como um nó filho. Não é pois necessário estar atento a referências e apontadores pois toda a informação está compactada no mesmo local (a principal razão da normalização do esquema). De seguida, faz-se a fusão do fragmento resultante da execuçaão da expressao Xpath com a árvore do documento (6). Após a fusão o Merger redirecciona o fluxo para o Serializer (7) que grava as alterações que serão posteriormente transformadas via Transformer. De realçar que se o elemento a editar já contiver algum valor este é adicionado no documento resultado como valor do atributo xesb:data (8). Desta maneira consegue-se passar o valor actual, de novo para o utilizador.

O Transformer controla a transformação do documento mesclado (XML + XSD). A Fig. 3 retrata os templates utilizados neste processo. A parte do documento que diz

respeito ao XML fica a cargo do template carregado em memória e que é específico ao tipo de documento (se não se tiver associado nenhum é utilizado um por omissão – xesbDefault.xsl), enquanto que qualquer excerto XSD é tratado pelos ficheiros de transformação xesbEdit.xsl e xesbFunc.xsl, ou seja, quaisquer elementos/atributos do namespace do XML Schema são transformados por estes templates que têm por missão a transformação de declarações de elementos (e respectivas restrições) em elementos de formulário associados com módulos pré-definidos e codificados em JavaScript para validação do lado cliente.

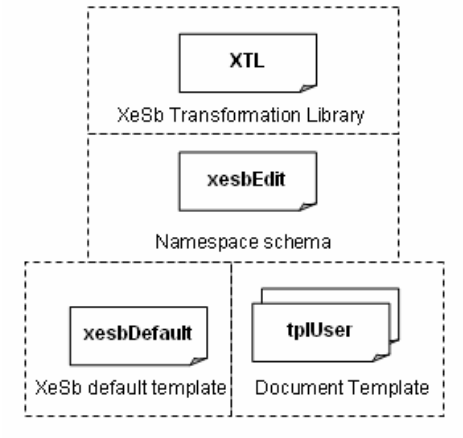


Fig. 3. Templates do componente Transformer

A linguagem objectivo das transformações é XHTML que é seguidamente consumido pelo navegador Web usado pelo utilizador. Este componente utiliza o módulo Formatter, no caso da necessidade de exportação dos dados para formatos diferentes, tais como, PDF, PS, TXT, AWT, SVG, MIF ou PCL. Neste caso a linguagem objectivo da transformação (usa-se o ficheiro xesbDefaultFO.xsl) é o XSL-FO e, juntamente com o documento instância XML, servem de input para um formatador[2] que transforma os dados para o output desejado. De salientar ainda neste componente Transformer a existência de uma biblioteca de funções (XTL - XESB Transformation Library), que permite a criação de templates específicos para novos tipos de documentos. O ficheiro editado é mantido na área de trabalho para posterior acesso ou mesmo o seu descarregamento para o computador do utilizador.
Em termos de visualização do documento, o XESB disponibiliza o modo de edição onde o utilizador interage com formulários, o modo em árvore onde o écran é sub-dividido em 2 frames (Fig.4): árvore do tipo de documento e o formulário de edição e uma visualização do código do documento instância.

[2] Usou-se o open-source da Apache chamado FOP e que é uma implementação parcial do standard XSL-FO 1.0 do W3C

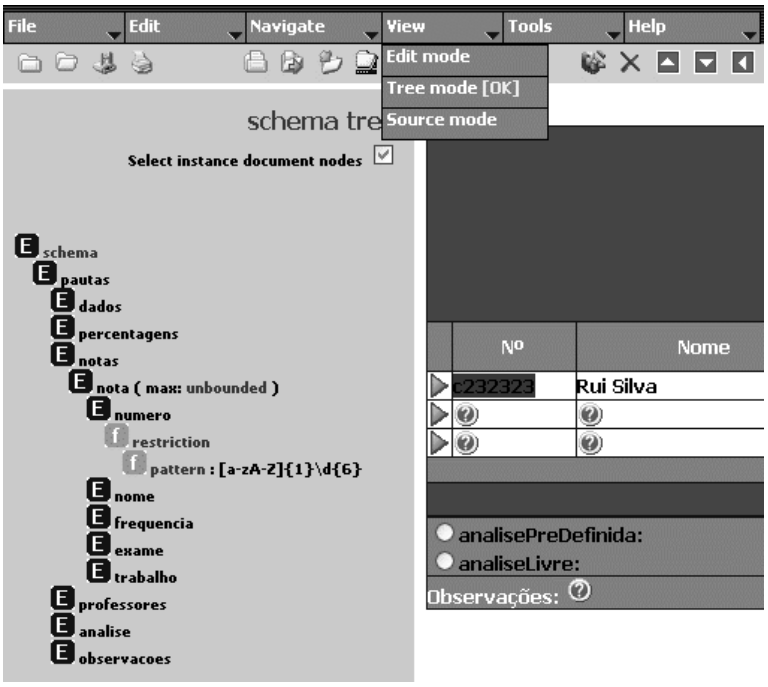


Fig. 4. Visualização em árvore do tipo de documento a editar

5. Conclusão

O XESB é um protótipo de um editor de XML que serviu essencialmente para validar uma abordagem: a utilização de edição estrutural e estruturação como aplicação web cliente-servidor. Embora esta abordagem tenha sido claramente validada pelo XESB este protótipo revelou algumas deficiências que esperamos colmatar em projectos futuros nesta área.

- **Suporte da especificação XML Schema:** O XESB não tem suporte a algumas áreas constituintes da especificação XML Schema, nomeadamente, o modelo de conteúdo de opção (elementos choice como nós internos), as definições de restricões de identidade (unique, key, keyref, selector e field), as notações (notation), a gestão de documentos (redefine, import, include) e uma característica da declaração de elementos (substitutionGroup). O facto de não aceitar modelos de conteúdo de opção em nós internos e o fraco suporte para conteúdos mistos, são talvez a maior limitação deste editor. Estas limitações devem-se essencialmente ao esforço de desenvolvimento necessário face ao tempo disponível para a sua

implementação. O suporte a outros formalismos esquemáticos (RELAX NG, Schematron, etc.) é também desejável.

- **Edição de documentos esquemáticos e de transformação:** O XESB permite a edição de documentos XML, mas os documentos esquemáticos e de transformação são também documentos XML, podendo-se assim beneficiar e utilizar as capacidades do próprio XESB. Mas devido às limitações apresentadas anteriormente e à estrutura geral do sistema, essa funcionalidade, que se considera primordial, não está incluída;

- **Inferência de documentos esquemáticos através de vários documentos instância:** A existência dum processo de inferência de um documento esquemático deverá ter como base vários documentos instância XML, garantindo assim uma melhor caracterização do mesmo. O TRANG um software de conversão da Thai Open Source Software Center Ltd poderá vir a ser integrado. A ideia principal é que através de um (ou vários) documentos XML se possa editar a própria linguagem definida no documento inferido, para tal é necessária a inclusão do Schema do próprio documento esquemático;

- **Interface gráfico para procura XPath:** A procura através de expressões XPath é uma das mais importantes funcionalidades de um editor XML. Com vista a alargar este mecanismo para utilizadores leigos nesta tecnologia era interessante incluir uma interface gráfica que permitisse a manipulação das expressões abstraindo o utilizador do formalismo inerente.

Em suma, existem várias funcionalidades a integrar, em trabalho futuro, tendo em vista a evolução do editor XESB, com vista a assumir-se como um ambiente integrado para o desenvolvimento de aplicações XML.

Referências

[1] W3C World Wide Web Consortium (http://www.w3.org).
[2] JavaML: A Markup Language for Java Source Code, Greg J. Badros - Dept. of Computer Science and Engineering - University of Washington - Seattle, WA USA
[3] Professional XML, Mark Birbeck and others, Wrox, 2ª edição, 2001.
[4] The XML Web: a First Study - Laurent Mignet, Denilson Barbosa, Pierangelo Veltri - 2003.
[5] BitFlux XML Editor - http://bitfluxeditor.org/
[6] The Synthesizer Generator: A System for Constructing Language Based Editors. Texts and Monographs in Computer Science. Thomas Reps and Tim Teitelbaum Springer Verlag, 1989.
[7] INES - Ambiente para Construção Assistida de Editores Estruturados baseados em SGML, Alda Reis Lopes, José Carlos Ramalho, Pedro Henriques - Departamento de Informática - Universidade do Minho – 1997
[8] Psgml: a gnu emacs major mode for editing sgml, Lennart Staflin.
[9] W3Schools (http://www.w3schools.com).
[10] Anotação estrutural de documentos e sua semântica, José Carlos Leite Ramalho, PhD thesis, Escola de Engenharia - Departamento de Informática - Universidade do Minho, 2000.

XML Processing and Logic Programming

Jorge Coelho[1] and Mário Florido[2]

[1] Instituto Superior de Engenharia do Porto & LIACC
Porto, Portugal
[2] University of Porto, DCC-FC & LIACC
Porto, Portugal
{jcoelho,amf}@ncc.up.pt

Abstract. In this paper we describe a constraint solving module which we call CLP(Flex), for dealing with the unification of terms with flexible arity function symbols. This approach results in a flexible and high declarative model for XML processing.

Keywords: XML Processing Languages, Logic Programming, Constraint Logic Programming

1 Introduction

XML is a notation for describing trees with an arbitrary finite number of leaf nodes. Thus a constraint programming language dealing with terms where function symbols have an arbitrary finite arity should lead to an elegant and declarative way of processing XML.

In this paper we describe a constraint logic programming language, CLP(Flex), similar in spirit to mainstream CLP languages but specialized to the domain of XML processing. Its novel features are the use of *flexible arity function symbols* and a corresponding mechanism for a *non-standard unification* in a theory with flexible arity symbols and variables which can be instantiated by an arbitrary finite sequence of terms. We translate XML documents to terms and use the new unification to process parts of these terms.

Unification with flexible arity symbols is no new notion. An unification algorithm for these terms was defined in [18] where it was used as a *Mathematica* package incorporated in the *Theorema* system (see [5]). Here we changed the algorithm presented in [18] to give the solutions incrementally, an essential feature to use it in a non-deterministic backtracking-based programming language such as *Prolog*. The main contributions of this paper are the presentation of the implementation and use of a constraint programming module, based on unification of terms with function symbols of flexible arity, as a highly declarative model for XML processing. Note that the work described in this paper was first presented in a previous paper from the authors ([8]).

This article focuses on language design, shows its adequacy to write applications that handle, transform and query XML documents, and sketches solutions to implementation issues. A distribution of our language can be found in:

http://www.ncc.up.pt/xcentric/

We assume that the reader is familiar with logic programming ([19]) and CLP ([16, 15]), and knows the fundamental features of XML ([25]).

2 Related Work

Mainstream languages for XML processing such as XSLT ([26]), XDuce ([13]), CDuce ([1]) and Xtatic ([27]) rely on the notion of trees with an arbitrary number of leaf nodes to abstract XML documents. However these languages are based on functional programming and thus the key feature here is pattern matching, not unification. The main motivation of our work was to extend unification for XML processing, such as the previous functional based languages extended pattern matching. Constraints revealed to be the natural solution to our problem.

Languages with flexible arity symbols have been used in various areas: Xcerpt ([3]) is a query and transformation language for XML which also used terms with flexible arity function symbols as an abstraction of XML documents. It used a special notion of term (called *query terms*) as patterns specifying selection of XML terms much like Prolog goal atoms. The underlying mechanism of the query process was *simulation unification* ([4]), used for solving inequations of the form $q \leq t$ where q is a query term and t a term representing XML data. This framework was technically quite different from ours, being more directed to query languages and less to XML processing. The Knowledge Interchange Format KIF ([12]) and the tool Ontolingua [11] extend first order logic with variable arity function symbols and apply it to knowledge management. Feature terms [24] can also be used to denote terms with flexible arity and have been used in logic programming, unification grammars and knowledge representation. Unification for flexible terms has as particular instances previous work on word unification ([14, 23]), equations over lists of atoms with a concatenation operator ([9]) and equations over free semigroups ([20]). Kutsia ([18]) defined a procedure for unification with sequence variables and flexible arity symbols applied to an extension of *Mathematica* for mathematical proving ([5]). From all the previous frameworks we followed the work of Kutsia because it is the one that fits better in our initial goal, which was to define a highly declarative language for XML processing based on an extension of standard unification to denote the same objects denoted by XML: trees with an arbitrary number of leafs. Although our algorithm is based on this previous one it has some differences motivated by its use as a constraint solving method in a CLP package:

– Kutsia algorithm gave the whole set of solutions to an equality problem as output. We changed that point accordingly to the standard *backtracking* model of *Prolog*. We give as output one answer substitution and subsequent calls to the same query will result in different answer substitutions computed by backtracking. When every solution is computed the query fails indicating that there are no more solutions.

- a direct consequence of the previous point is that our implementation deals with infinite sets of solutions (see example 46). It simply gives all solutions by backtracking.
- Kutsia algorithm was a new definition of unification for the case of terms with flexible arity symbols. Our implementation transforms the initial set of constraints into a different (larger) set of equalities solved by standard unification and uses standard *Prolog* unification for propagating substitutions.

Finally we should refer that the use of standard terms (with fixed arity function symbols) to denote XML documents was made before in several systems. For example Pillow ([22]) used a low level representation of XML where the leaf nodes in the XML trees were represented by lists of nodes and Prolog standard unification was used for processing. In [2] and [7] XML was represented directly by terms of fixed arity. A last reference to some query languages for XML (such as XPathLog [21]) where Prolog style variables are used as an extension to XPath in a query language for XML.

3 Terms with Flexible Arity Symbols and Sequence Variables

3.1 Constraint Logic Programming

Constraint Logic Programming (CLP) [16] is the name given to a class of languages based on the paradigm of rule-based constraint programming. Each different language is obtained by specifying the domain of discourse and the functions and relations on the particular domain. This framework extends the logic programming framework because it extends the Herbrand universe, the notion of *unification* and the notion of equation, accordingly to the new computational domains. There are many examples of CLP languages, such as, Prolog III [10] which employs equations and disequations over rational trees and a boolean algebra, CLP(R), [17] which has linear arithmetic constraints over the real numbers and ECLiPSe, [6], that computes over several domains: a Boolean algebra, linear arithmetic over the rational numbers, constraints over finite domains and finite sets. Prolog itself can be viewed as a CLP language where constraints are equations over an algebra of finite trees. A complete description of the major trends of the fundamental concepts about CLP can be found in [16].

3.2 CLP(Flex)

The idea behind CLP(Flex) is to extend Prolog with terms with flexible arity symbols and sequence variables. We now describe the syntax of CLP(Flex) programs and their intuitive semantics.

In CLP(Flex) we extend the domain of discourse of Prolog (trees over uninterpreted functors) with finite sequences of trees.

Definition 31 *A* sequence $\tilde{t}$*, is defined as follows:*

– ε *is the empty sequence.*
– $t_1, \tilde{t}$ *is a sequence if* t_1 *is a term and* $\tilde{t}$ *is a sequence*

Example 31 *Given the terms* $f(a)$, b *and* X, *then* $\tilde{t} = f(a), b, X$ *is a sequence.*

Equality is the only relation between trees. Equality between trees is defined in the standard way: two trees are equal if and only if their root functor are the same and their corresponding subtrees, if any, are equal.

We now proceed with the syntactic formalization of CLP(Flex), by extending the standard notion of Prolog term with flexible arity function symbols and sequence variables.

We consider an alphabet consisting of the following sets: the set of standard variables, the set of sequence variables (variables are denoted by upper case letters), the set of constants (denoted by lower case letters), the set of fixed arity function symbols and the set of flexible arity function symbols.

Definition 32 *The set of terms over the previous alphabet is the smallest set that satisfies the following conditions:*

1. *Constants, standard variables and sequence variables are terms.*
2. *If* f *is a flexible arity function symbol and* $t_1, \ldots, t_n$ $(n \geq 0)$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*
3. *If* f *is a fixed arity function symbol with arity* n, $n \geq 0$ *and* $t_1, \ldots, t_n$ *are terms such that for all* $1 \leq i \leq n$, t_i *does not contain sequence variables as subterms, then* $f(t_1, \ldots, t_n)$ *is a term.*

Terms of the form $f(t_1, \ldots, t_n)$ where f is a function symbol and $t_1, \ldots, t_n$ are terms are called *compound terms.*

Definition 33 *If* t_1 *and* t_2 *are terms then* $t_1 = t_2$ *(standard Prolog unification) and* $t_1 = * = t_2$ *(unification of terms with flexible arity symbols) are constraints.*

A constraint $t_1 = * = t_2$ or $t_1 = t_2$ is solvable if and only if there is an assignment of sequences or ground terms, respectively, to variables therein such that the constraint evaluates to *true*, i.e. such that after that assignment the terms become equal.

Remark 31 *In what follows, to avoid further formality, we shall assume that the domain of interpretation of variables is predetermined by the context where they occur. Variables occurring in a constraint of the form* $t_1 = * = t_2$ *are interpreted in the domain of sequences of trees, otherwise they are standard Prolog variables. In CLP(Flex) programs, therefore, each predicate symbol, functor and variable is used in a consistent way with respect to its domain of interpretation.*

CLP(Flex) programs have a syntax similar to Prolog extended with the new constraint $= * =$. The operational model of CLP(Flex) is the same of Prolog.

3.3 Constraint Solving

Constraints of the form $t_1 = * = t_2$ are solved by a non-standard unification that calculates the corresponding minimal complete set of unifiers. This non-standard unification is based on Kutsia algorithm [18]. As motivation we present some examples of unification:

Example 32 *Given the terms $f(X, b, Y)$ and $f(a, b, b, b)$ where X and Y are sequence variables, $f(X, b, Y) = * = f(a, b, b, b)$ gives three results:*

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

Example 33 *Given the terms $f(b, X)$ and $f(Y, d)$ where X and Y are sequence variables, $f(b, X) = * = f(Y, d)$ gives two possible solutions:*

1. $X = d$ *and* $Y = b$
2. $X = N, d$ *and* $Y = b, N$ *where N is a new sequence variable.*

Note that this non-standard unification is conservative with respect to standard unification: in the last example the first solution corresponds to the use of standard unification.

4 XML Processing in CLP(Flex)

In CLP(Flex) there are some auxiliary predicates for XML processing. Through the following examples we will use the builtin predicates *xml2pro* and *pro2xml* which respectively convert XML files into terms and vice-versa. We will also use the predicate *newdoc(Root,Args,Doc)* where *Doc* is a term with functor *Root* and arguments *Args* (this predicate is similar to =.. in Prolog).

4.1 XML as Terms with Flexible Arity Symbols

An XML document is translated to a term with flexible arity function symbol. This term has a main functor (the root tag) and zero or more arguments. Although our actual implementation translates attributes to a list of pairs, since attributes do not play a relevant role in this work we will omit them in the examples, for the sake of simplicity. Consider the simple XML file presented bellow:

```
<addressbook>
        <record>
                <name>John</name>
                <address>New York</address>
                <email>john.ny@mailserver.com</email>
        </record>
...
</addressbook>
```

The equivalent term is:

```
addressbook(record(
               name('John'),
                address('New York'),
                email('john.ny@mailserver.com')),
                ...)
```

4.2 Using Constraints in CLP(Flex)

One application of CLP(Flex) constraint solving is XML processing. With non-standard unification it is easy to handle parts of XML files. In this particular case, parts of terms representing XML documents.

Example 41 ***Address Book translation.*** *In this example we use the address book document of the previous example. In this address book we have sometimes records with a phone tag. We want to build a new XML document without this tag. Thus, we need to get all the records and ignore their phone tag (if they have one). This can be done by the following program (this example is similar to one presented in XDuce [13]):*

```
translate:-
  xml2pro('addressbook.xml','addressbook.dtd',Xml),
  process(Xml,NewXml),
  pro2xml(NewXml,'addressbook2.xml').

process(A,NewA):-
  findall(Record,records_without_phone(A,Record),LRecords),
  newdoc(addressbook,LRecords,NewA).

records_without_phone(A1,A2):-
  A1 =*= addressbook(_,record(name(N),address(A),_,email(E)),_),
  A2 = record(name(N),address(A),email(E)).
```

Predicate translate/0 first translates the file "addressbook.xml" into a CLP(Flex) term, which is processed by process/2, giving rise to a new CLP(Flex) term and then to the new document "addressbook2.xml". This last file contains the address records without the phone tag.

Example 42 ***Book Stores.*** *In this example we have two XML documents with a catalogue of books in each ("bookstore1.xml" and "bookstore2.xml"). These catalogues refer to two different book stores. Both "bookstore1.xml" and "bookstore2.xml" have the same DTD and may have similar books. A sample of one of this XML documents can be:*

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
 <book number="1">
   <name>Art of Computer Programming</name>
   <author>Donald Knuth</author>
```

```
  <price>140</price>
  <year>1998</year>
 </book>
 ...
 <book number="500">
  <name>Haskell:The Craft of Functional Programming (2nd Edition)</name>
  <author>Simon Thompson</author>
  <price>41</price>
  <year>1999</year>
 </book>
</catalog>
```

1. *To check which books are cheaper at bookstore 1 we have the following program:*

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

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

   The predicate best_prices/1 *returns the cheaper books at "bookstore1.xml", one by one, by backtracking.*
2. *To get all the books from one author, the author of a book or all the pairs author/book, we have the following code:*

```
books_from(Author,Book):-
  xml2pro('bookstore1.xml','bookstore1.dtd',Xml),
  process2(Xml,Author,Book).

process2(Xml,Author,Book):-
  Xml =*= catalog(_,book((name(Book),author(Author),_)),_).
```

   Here books_from/2 retrieves, by backtracking, every Author/Book names from file "bookstore1.xml".

The previous programs are rather simple. This stresses the highly declarative nature of CLP(Flex) when used for XML processing.

4.3 The Unification Algorithm

The unification algorithm, as presented in [18], consists of two main steps, *Projection* and *Transformation*. The first step, *Projection* is where some variables are erased from the sequence. This is needed to obtain solutions where those

variables are instantiated by the empty sequence. The second step, *Transformation* is defined by a set of rules where the non-standard unification is translated to standard Prolog unification.

Definition 41 *Given terms T_1 and T_2, let V be the set of variables of T_1 and T_2 and A be a subset of V.* Projection *eliminates all variables of A in T_1 and T_2.*

Example 43 *Let $T_1 = f(b, Y, f(X))$ and $T_2 = f(X, f(b, Y))$. In the projection step we obtain the following cases (corresponding to $A = \{\}$, $A = \{X\}$, $A = \{Y\}$ and $A = \{X, Y\}$):*

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

Our version of Kutsia algorithm uses a special kind of terms, here called, *sequence terms* for representing sequences of arguments.

Definition 42 *A* sequence term, *$\bar{t}$ is defined as follows:*

- *empty is a* sequence term.
- *$seq(t, \bar{s})$ is a* sequence term *if t is a term and $\bar{s}$ is a* sequence term.

Definition 43 *A sequence term in* normal form *is defined as:*

- *empty is in normal form*
- *$seq(t_1, t_2)$ is in normal form if t_1 is not of the form $seq(t_3, t_4)$ and t_2 is in normal form.*

Example 44 *Given the function symbol f, the variable X and the constants a and b:*

$$seq(f(seq(a, empty)), seq(b, seq(X, empty)))$$

is a sequence term *in normal form.*

Note that sequence terms are lists and sequence terms in normal form are flat lists. We introduced this different notation because sequence terms are going to play a key role in our implementation of the algorithm and it is important to distinguish them from standard Prolog lists. Sequence terms in normal form correspond trivially to the definition of sequence presented in definition 31. In fact sequence terms in normal form are an implementation of this definition. Thus, in our implementation, a term $f(t_1, t_2, \ldots, t_n)$, where f has flexible arity, is internally represented as $f(seq(t_1, seq(t_2, \ldots, seq(t_n, empty)\ldots))$, that is, arguments of functions of flexible arity are always represented as elements of a sequence term.

We now define a normalization function to reduce sequence terms to their normal form.

Definition 44 *Given the sequence terms $\bar{t}_1$ and $\bar{t}_2$, we define sequence term* concatenation *as $\bar{t}_1 + +\bar{t}_2$, where the ++ operator is defined as follows:*

$$\begin{array}{rl} empty & ++\ \bar{t} = \bar{t} \\ seq(t_1,\bar{t_2})\ ++\ \bar{t_3} & = seq(t_1,\bar{t_2}++\bar{t_3}) \end{array}$$

Definition 45 *Given a sequence term, we define sequence term* normalization *as:*

$$\begin{array}{ll} normalize(empty) & = empty \\ normalize(t) & = seq(t,empty),\ if\ t\ is\ a\ constant\ or\ variable. \\ normalize(t) & = seq(f(normalize(t_1)),empty),\ if\ t = f(t_1). \\ normalize(seq(t_1,\bar{t})) & = normalize(t_1)\ ++\ normalize(\bar{t}) \end{array}$$

Proposition 41 *The normalization procedure always terminates yielding a sequence in normal form.*

Transformation rules are defined by the rewrite system presented in figure 1. We consider that upper case letters (X,Y,...) stand for sequence variables, lower case letters (s,t,...) for terms and overlined lower case letters ($\bar{t}, \bar{s}$) for *sequence terms*. These rules implement Kutsia algorithm applied to sequence terms by using standard Prolog unification. Note that rules 6, 7, 8 and 9 are non-deterministic: for example rule 6 states that in order to solve $seq(X, \bar{t}) = * = seq(s_1, \bar{s})$ we can solve $\bar{t} = * = \bar{s}$ with $X = s_1$ or we can solve $normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$ with $X = seq(s_1, seq(X_1, empty))$. At the end the solutions given by the algorithm are normalized by the *normalize* function. When none of the rules is applicable the algorithm fails. Kutsia showed in [18] that this algorithm terminated if it had a cycle check, (i.e. it stopped with failure if a unification problem gave rise to a similar unification problem) and if each sequence variable does not occur more than twice in a given unification problem.

For the sake of simplicity, the following examples are presented in sequence notation, alternatively to the *sequence term* notation.

Example 45 *Given $t = f(X, b, Y)$ and $s = f(c, c, b, b, b, b)$ the projection step leads to the following transformation cases:*

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

Using the transformation rules we can see that only the first and third unifications succeed. For $f(X, b, Y) = * = f(c, c, b, b, b, b)$ we have the following answer substitutions:

- $X = c, c$ *and* $Y = b, b, b$
- $X = c, c, b$ *and* $Y = b, b$

Success

(1)	t	$=*=$	s	$\Longrightarrow$ True, if $t == s$ [1]
(2)	X	$=*=$	t	$\Longrightarrow X = t$ if X does not occur in t.
(3)	t	$=*=$	X	$\Longrightarrow X = t$ if X does not occur in t.

Eliminate

(4)	$f(\bar{t})$	$=*=$	$f(\bar{s})$	$\Longrightarrow \bar{t} = * = \bar{s}$
(5)	$seq(t_1, \bar{t}_n)$	$=*=$	$seq(s_1, \bar{s}_m)$	$\Longrightarrow t_1 = * = s_1$, $\bar{t}_n = * = \bar{s}_m$
(6)	$seq(X, \bar{t})$	$=*=$	$seq(s_1, \bar{s})$	$\Longrightarrow X = s_1$, if X does not occur in s_1, $\bar{t} = * = \bar{s}$. $\Longrightarrow X = seq(s_1, seq(X_1, empty))$, if X does not occur in s_1, $normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$, where X_1 is a new variable.
(7)	$seq(t_1, \bar{t})$	$=*=$	$seq(X, \bar{s})$	$\Longrightarrow X = t_1$, if X does not occur in t_1, $\bar{t} = * = \bar{s}$. $\Longrightarrow X = seq(t_1, seq(X_1, empty))$, if X does not occur in t_1, $normalize(\bar{t}) = * = normalize(seq(X_1, \bar{s}))$, where X_1 is a new variable.
(8)	$seq(X, \bar{t})$	$=*=$	$seq(Y, \bar{s})$	$\Longrightarrow X = Y$ $\bar{t} = * = \bar{s}$. $\Longrightarrow X = seq(Y, seq(X_1, empty)$, $normalize(seq(X_1, \bar{t})) = * = normalize(\bar{s})$, where X_1 is a new variable and X, Y are distinct. $\Longrightarrow Y = seq(X, seq(Y_1, empty))$, $normalize(\bar{t}) = * = normalize(seq(Y_1, \bar{s}))$, where Y_1 is a new variable and X, Y are distinct.

Split

(9)	$seq(t_1, \bar{t})$	$=*=$	$seq(s_1, \bar{s})$	$\Longrightarrow$ if $t_1 = * = s_1 \Longrightarrow r_1 = * = q_1$ then $normalize(seq(r_1, \bar{t})) = * = normalize(seq(q_1, \bar{s}))$ $\vdots$ $\Longrightarrow$ if $t_1 = * = s_1 \Longrightarrow r_w = * = q_w$, $normalize(seq(r_w, \bar{t}_n)) = * = normalize(seq(q_w, \bar{s}))$, where t_1 and s_1 are compound terms.

Fig. 1. Transformation rules

- $X = c, c, b, b$ *and* $Y = b$

  And for $f(X, b) = * = f(c, c, b, b, b, b)$ *we have:*

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

Example 46 *In some cases we can have an infinite set of solutions for the unification of two given terms. For example when we solve* $f(X, a) = * = f(a, X)$ *the solutions are:*

- $X = a$
- $X = a, a$
- $X = a, a, a$
- $X = a, a, a, a$
- $\ldots$

In the previous example Kutsia algorithm with the cycle check fails immediately after detecting that it is repeating the unification problem. Our implementation gives all solutions by backtracking. The correctness of the non-standard unification algorithm in figure 1 is presented in [8].

5 Conclusion

In this paper we present a constraint solving extension for Prolog to deal with terms with flexible arity symbols. We show an application of this framework to XML processing yielding a highly declarative language for that purpose. Some points can be further developed in future work:

- an extension with further built-in predicates and constraints, such as predicates to deal with XML types (DTDs and XML-Schema);
- XML attributes are ignored in our language. We just translate them to lists of pairs. More declarative representation of attributes, such as sets of equalities, and an extension to unification to deal with this new constraints would be a relevant feature which is left for future work;
- finally we note that, CLP(Flex) may have applications in other areas different from XML-processing.

Acknowledgements The work presented in this paper has been partially supported by funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

[1] == denotes syntactic equality (in opposite with = which denotes standard unification)

References

1. Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. In *Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming*, pages 51–63, Uppsala, Sweden, 2003. ACM Press.
2. H. Boley. Relationships between logic programming and XML. In *Proc. 14th Workshop Logische Programmierung*, 2000.
3. F. Bry and S. Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *2nd Annual International Workshop Web and Databases*, volume 2593 of *LNCS*. Springer Verlag, 2002.
4. F. Bry and S. Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *International Conference on Logic Programming (ICLP)*, volume 2401 of *LNCS*, 2002.
5. B. Buchberger, C. Dupre, T. Jebelean, B. Konev, F. Kriftner, T. Kutsia, K. Nakagawa, F. Piroi, D. Vasaru, and W. Windsteiger. The Theorema System: Proving, Solving, and Computing for the Working Mathematician. Technical Report 00-38, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, 2000.
6. A M Cheadle, W Harvey, A J Sadler, J Schimpf, K Shen, and M G Wallace. ECLiPSe: An Introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London, London, 2003.
7. J. Coelho and M. Florido. Type-based XML Processing in Logic Programming. In V. Dahl and P. Wadler, editors, *Practical Aspects of Declarative Languages*, volume 2562 of *Lecture Notes in Computer Science*, pages 273–285, New Orleans, USA, 2003. Springer Verlag.
8. Jorge Coelho and Mario Florido. CLP(Flex): Constraint Logic Programming Applied to XML Processing. In *Ontologies, Databases and Applications of SEmantics (ODBASE)*, LNCS, Agia Napa, Cyprus, 2004. Springer Verlag.
9. A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.
10. A. Colmerauer. Prolog III Reference and Users Manual, Version 1.1. In *PrologIA*, Marseilles, 1990.
11. A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46(6):707–727, 1997.
12. M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual TR Logic-92-1. Technical report, Stanford University, Stanford, 1992.
13. Haruo Hosoya and Benjamin Pierce. XDuce: A typed XML processing language. In *Third International Workshop on the Web and Databases (WebDB2000)*, volume 1997 of *Lecture Notes in Computer Science*, 2000.
14. J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, 1990.
15. J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proceedings of the Fourteenth Annual ACM Symp. on Principles of Programming Languages, POPL '87*, pages 111–119, Munich, Germany, 1987. ACM Press.
16. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

17. Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap. The CLP(R) Language and System. In *Trans. Program. Lang. Syst.*, volume 14, pages 339–395. ACM, 1992.
18. T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proceedings of Joint AICS'2002 - Calculemus'2002 conference*, volume 2385 of *Lecture Notes in Artificial Intelligence*, pages 290–304, Marseille, France, 2002. Springer Verlag.
19. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.
20. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik USSR*, 103:147–236, 1977.
21. Wolfgang May. XPathLog: A Declarative, Native XML Data Manipulation Language. In *International Database Engineering & Applications Symposium (IDEAS '01)*, Grenoble, France, 2001. IEEE.
22. Pillow: Programming in (Constraint) Logic Languages on the Web. http://clip.dia.fi.upm.es/Software/pillow/pillow.html.
23. Klaus U. Schulz. Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11(2):149–184, 1993.
24. Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.
25. Extensible Markup Language (XML). http://www.w3.org/XML/.
26. XSL Transformations (XSLT). http://www.w3.org/TR/xslt/, 1999.
27. Xtatic. http://www.cis.upenn.edu/~bcpierce/xtatic/.

Um Processador Construtivista na Web para documentos anotados (PcWDa)

Mariana Isabel Ferreira, Sandra Cristina Lopes, and Pedro Rangel Henriques

Universidade do Minho, Departamento de Informática
4710-057, Braga, Portugal
prh@di.uminho.pt

Resumo A notação XML, cada vez mais, desempenha um papel importante na representação textual da informação estruturada ou semi-estruturada, sendo a norma mais vulgarizada para marcação de documentos. Torna-se por isso indispensável a existência de ferramentas que possibilitem desenvolver(editar e validar) eficientemente documentos XML.
Neste artigo pretende-se discutir a especificação e o desenvolvimento de uma dessas ferramentas usando tecnologias Java/ XML, que possa ser utilizada em qualquer plataforma a partir de um browser, para apoio ao ensino da *construção estruturada de documentos* (via anotação). O Processador deve permitir aos alunos aprender, segundo uma abordagem construtivista—em que o próprio aprendiz vai criando os seus objectos e vai absorvendo o conhecimento com as experiências bem ou mal sucedidas que realiza—a criar documentos anotados de uma família entre um conjunto de hipóteses pré-definidas (relatórios, cartas, fichas de leitura, etc.). À medida que o aluno edita um documento, o Processador irá mostrando alternativas e fornecendo explicações, contudo deixará que seja ele a tomar decisões e a percorrer os seus próprios caminhos. Para isso, é necessário definir, à partida, o DTD de cada família de documentos e, para cada elemento do DTD, indicar as informações (explicação do seu significado, exemplos, etc.) a mostrar. O modo de interacção—da *edição guiada com ajuda automática* à *edição livre com ajuda sob pedido*—deve ser facilmente controlado (pelo aluno, ou pelo tutor) de modo a poder adaptar-se a diferentes fases ou estilos de aprendizagem.

1 Introdução

O projecto a apresentar ao longo deste artigo consiste no desenvolvimento de um processador de texto (editor/estruturador), construtivista, para ser usado tanto por utilizadores praticamente leigos como por utilizadores que dominam a linguagem XML [7,11,6,17], tendo em vista fins educativos (no âmbito das disciplinas de *Introdução à Informática*, na unidade lectiva de *Processamento de Documentos*). Para ensinar, num ambiente construtivista[3], a criar documentos, partindo da definição clara e explícita da sua estrutura à qual se junta depois

progressivamente o conteúdo, parece-nos fundamental usar uma abordagem baseada na anotação do documento a produzir—neste processo a ênfase não é na linguagem de anotação, mas sim na tarefa de identificação das partes que vão formar o todo. Assim, se decidiu usar a metalinguagem XML, um standard para anotação de documentos, como suporte ao processador em causa; XML, além de ser a norma actualmente mais vulgarizada, assegura a longevidade dos documentos pois permite o seu armazenamento e manuseamento num formato universal[1], aumentando a sua portabilidade. Contudo é importante deixar claro que, apesar de termos optado pela codificação dos documentos em XML, não é de todo nosso intuito desenvolver uma ferramenta para apoio ao ensino de XML; esta norma é aqui apenas um *meio*, que pode até ser transparente para o aprendiz, e não é de todo um *fim*.

Por razões referidas à frente, a ferramenta que vai apoiar tal forma de ensino deve estar disponível na Web, daí a termos designado por PcWDa—*Processador Construtivista na Web para Documentos Anotados*.

Interfaces educativas[19] servem como meio à negociação de significado e à construção de conhecimentos específicos em contextos específicos. O melhor software educativo (uma possível forma de realizar as ditas Interfaces) deve estar orientado para ensinar as capacidades de raciocínio e não para a simples memorização de factos.

Numa perspectiva construtivista[15], a aprendizagem é concebida como um processo de acomodação e assimilação, em que os alunos modificam as suas estruturas cognitivas internas em função das suas experiências pessoais. Nesta teoria, os alunos são encarados como participantes activos, aprendendo de uma forma que depende do seu estado cognitivo concreto. Os conhecimentos prévios, interesses, expectativas e ritmos de aprendizagem são levados em conta nesta forma de aquisição de conhecimento. Segundo esta abordagem, a aprendizagem é entendida essencialmente como um processo de revisão, modificação e reorganização dos esquemas de conhecimento inicial dos alunos e de construção de outros novos; e o ensino é visto como um processo de ajuda prestado a esta actividade construtiva do aluno. Neste contexto, o professor é encarado como o mediador entre os conteúdos e os alunos, cabendo-lhe organizar ambientes de aprendizagem estimulantes que facilitem esta construção cognitiva.

Nos dias de hoje, estamos perante um contexto educativo em que o software que impera segue uma abordagem construtivista. Assim e continuando a seguir as ideias dos autores já citados, a aplicação que vai ser apresentada deve obedecer às seguintes condicionantes:

- O aprendiz deve sempre questionar-se sobre as consequências das suas atitudes, e, a partir dos seus erros ou acertos, ir construindo os seus conceitos; assim, o tutor ou a ferramenta de apoio ajuda-lo-á nesse processo em vez de servir apenas para verificar o que o aluno assimilou;
- Uma nova matéria deve ser apresentada do todo para as partes, com ênfase nos conceitos gerais;

[1] Independente da plataforma de trabalho ou do fornecedor das aplicações.

– A aprendizagem por descoberta predominará, devendo criar-se um ambiente rico em situações que o aluno tem de explorar;
– A negociação social do conhecimento e o estímulo à colaboração com os outros devem ser francamente favorecidos.

Ao aluno deve ser permitido ter algum *grau de iniciativa*. O editor a desenvolver deve obrigar a uma interacção muito grande do aprendiz com o objecto de estudo; este deve estar integrado no contexto educativo do sujeito, dentro das suas condições, de forma a estimulá-lo e desafiá-lo.

Foram várias as razões que nos levaram a desenvolver o sistema de raiz, em vez de adaptar um editor já existente, como o XML-SPY [2] ou WORD com templates. A primeira delas prende-se com a necessidade de obter uma aplicação acessível na Web para se integrar com a plataforma de apoio ao grupo de disciplinas em causa[2], também ela disponibilizada na Web para ser acedida em qualquer lugar, a qualquer hora, via um *browser* vulgar independente da plataforma e da tecnologia de implementação. Outra forte razão, associada à anterior, é a questão de precisarmos de uma ferramenta que seja livre, sem necessidade de pagar direitos de autor. Por fim, também era desejável termos total controlo sobre o programa, para podermos satisfazer os requisitos inerentes ao modelo de aprendizagem que pretendemos seguir.

Na hora de concluirmos a escrita deste artigo, tivemos conhecimento de um processador muito parecido, o XESB [16], em desenvolvimento (em fase de conclusão) no âmbito de um projecto de mestrado na Faculdade de Ciências da Universidade do Porto. Ambos baseiam-se na Web para uma ampla disponibilização, de fácil acesso, e ambos são editores dirigidos pela sintaze que procuram libertar o o utilizador do conhecimento dos detalhes sintácticos do formalismo de anotação (que é de certa forma encapsulado), evidenciando a estrutura em si. O PcWDa, tal como o XESB, utiliza a definição do tipo de documento a processar[3] que guia todo o processo de edição, permitindo que se crie sempre documentos válidos, ou seja, documentos que obedecem a uma definição formal. É interessante constatar, pela leitura da dissertação acima referida, que ambos se guiam pelo mesmo tipo de preocupação; reconhecendo que as aplicações anteriormente desenvolvidas assumem que o utilizador sabe a semântica do documento, o que nem sempre acontece, pretendem proporcionar *a abstracção do conhecimento da sintaxe com vista a evidenciar a semântica do domínio*, De facto um pessoa que vai editar uma carta, pode ter conhecimento das parte constituintes, e da forma como se agrupam dentro, mas nem sempre sabe o significado de cada uma, o que é fundamental para produzir um documento válido[4].

Nas secções seguintes discutiremos os requisitos e características do Processador (2), a sua arquitectura e modelo UML(3) e a tecnologia utilizada na sua imple-

[2] SI^3 - um Sistema de Informação para apoio às disciplinas de Introdução à Informática para as Ciências Sociais.
[3] Está assim limitado a um conjunto pré-definido de tipos.
[4] Entende-se por documento válido, um documento cuja estrutura e conteúdo respeitam uma determinada definição de tipo de documento.

mentação (4). Por último será feita uma breve conclusão (5) em que se dará particular importância à forma de abordar o problema, à concepção do sistema e às decisões tecnológicas.

2 Requisitos e Funcionalidades do Processador

O Processador PcWDa será constituído por uma interface em que o utilizador começará por escolher um dos modelos de documentos existentes numa base de dados. De seguida, poderá escrever o documento pretendido, estruturando-o à sua maneira através da selecção de uma das alternativas oferecidas; para isso contará com a ajuda de uma *janela de dicas* onde serão apresentadas informações e exemplos dependentes do contexto em que o utilizador se encontra a escrever.

Para cada família de documentos suportada pelo PcWDa, terá de existir na base de dados um DTD [13] que descreva a sua estrutura; o DTD comporta-se assim como o modelo formal de cada tipo de documento. É verdade que nos tempos que correm podiamos ter optado pelo uso de XML-Schemas [10], uma especificação mais completa e rigorosa do tipo de documentos que além da definição estrutural (dos elementos e seus atributos) já permite incluir algumas restrições sintácticas e até semânticas, mas não há dúvida que o uso de XML-Schemas é muito mais pesado, quer em termos de quem constrói a descrição, quer em termos de quem desenvolve a programação dos documentos anotados. Porém o argumento mais forte para não o fazer é o facto explícitado na Introdução de não ser nosso intuito ensinar XML e a tecnologia associada. Assim sendo, constata-se que os DTD's são mais que suficientes para definir os nossos tipos de documentos, sendo simples e concisos quando comparados com os XML-Schemas. Além disso são o formalismo mais antigo pois têm uma relação intrínseca com o SGML (parte constituinte dele como se vê na referência acima citada). Um estudo descrito em [14] feito com uma amostra de 200 mil documentos XML mostra que em 2003 o uso dos DTD's em relação a outras alternativas para definir novos tipos de documentos era muito superior. Da amostra considerada, 49% usavam DTD's, 0,1% usavam XML-Schemase 50,9% não eram validados. Daqui conclui-se que os DTD's vão continuar a ser uma opção importante para a validação de documentos XML. Contudo, caso seja necessário existe sempre a possibilidade de converter um DTD para XML-Schema ou outra gramática estruturada; esta conversão pode ser feita através de mecanismos, sendo o NekoDTD [9] um exemplo.

Para que o utilizador se aperceba se o que está a fazer se encontra correctamente estruturado[5] poderá validar, a qualquer momento, o seu trabalho e verificar os erros.

No entanto, à medida que o utilizador edita o documento, o PcWDa irá mantendo controlo total sobre o estado da edição de forma a actualizar as informações de ajuda a fornecer. Estas ajudas aparecerão em duas janelas distintas: a janela de mensagens que dará indicações ao utilizador em relação à estrutura do documento, apontando as alternativas válidas em cada instante; e a janela de

[5] isto é, se está de acordo com as regras impostas pelo modelo

exemplos onde aparecerão indicações sobre o significado de cada elemento e onde se mostrará exemplos de preenchimento da parte do documento que está a ser editado.

O documento em edição terá uma representação interna à qual poderão estar associadas várias representações visuais do documento.

Para que se perceba o funcionamento pretendido do processador, vamos construir, bem à moda do *use-cases* do UML [4], um cenário de trabalho; imaginemos que o utilizador pretende desenvolver um **memorando**. No momento inicial e após escolher, no `Menu de Famílias de Documentos`, o tipo Memorando[6], a *janela de mensagens* da aplicação indicará quais os *elementos* que o utilizador pode introduzir—neste caso apenas a raiz, `Memo`. Neste momento e graças às regras semânticas associadas aos elementos do DTD fica disponível, à laia de ajuda, a possibilidade de pedir explicações sobre cada um dos *elementos* alternativos. Na *janela de exemplos* será apresentada uma explicação acerca do seu significado—*para que serve*? *como/quando se usa*?—e um exemplo de um documento XML desse tipo[7].

Quando o utilizador escolhe um *elemento*, a representação interna do documento é actualizada com essa informação na posição correcta da *hierarquia de elementos* do documento; deste modo, é de imediato actualizada a janela que indica os *elementos* que a seguir podem ser introduzidos pelo utilizador—após a escolha de Memo, surgirão os elementos `Remetente`, `Destinatario` e `Mensagem`.

Ao escolher o elemento `Remetente`, ou `Destinatario`, já não é indicado mais nenhum sub-elemento, mas é dito que se trata de um local para inserção de texto envolvido nas respectivas marcas (que são inseridas automaticamente); nessa altura, aparece na 2ª janela o exemplo respectivo e está, como acima, disponível ajuda semântica que será mostrada na mesma janela se o aprendiz o solicitar. Se a escolha anterior tivesse recaído na terceira hipótese, `Mensagem`, porque este *elemento* tem ainda *sub-elementos* (ver o DTD no apêndice A.2), aparece na *janela de mensagens* a indicação de que neste ponto podem ser inseridos um, ou mais, `Paragrafo`; na *janela de exemplos* tudo se passa como nas outras duas situações descritas anteriormente. A selecção de um *elemento* `Paragrafo` é tratada exactamente como no caso de `Remetente`, ou `Destinatario`, visto que este também não tem mais sub-elementos.

Resumindo, conforme o posicionamento do utilizador no documento é mostrada informação sobre os *elementos* que podem ser introduzidos e um exemplo relativo ao *elemento* em que se encontra. Além desta ajuda, ao longo da edição do documento, mal o utilizador o submeta para validação são mostrados todos os erros cometidos e apresentados exemplos relacionados com o elemento onde o erro ocorreu.

---

[6] Ver DTD no apêndice A.2.
[7] Ver XML no apêndice A.1.

Além da representação visual standard[8]—em que o documento estruturado em edição é mostrado na janela própria, com as marcas inseridas em torno dos blocos de texto respectivos—é possível associar outros conjuntos de *regras de transformação* que produzirão outras *vistas* do documento (umas poderão ser diferentes visualizações, que serão projectadas em janelas próprias, outras serão descrições que serão enviadas para ficheiro).

Em suma, deste cenário de utilização (e de outros análogos) podem-se enunciar as funcionalidades que se quer obter com a ferramenta PcWDa a desenvolver:

- ajudar o utilizador na edição de documentos, fornecendo-lhe, em função do tipo, informação sintáctica (sobre a estrutura) e semântica contextual;
- validar os documentos, de acordo com o seu tipo, mediante pedido do utilizador;
- gerar automaticamente diversas *vistas* do documento (representações visuais, ou outras).

3 Arquitectura do PcWDa e Modelo UML

A representação interna de um documento pode ser validada através do DTD que especifica as regras sintácticas e que, no nosso caso, está também associado à descrição semântica e a outras informações de ajuda. À representação interna poderão estar associados vários esquemas de transformação (escritos em CSS [5], XSL [20,17,12], etc.) que produzirão as diversas representações visuais do documento, ou outros resultados, conforme se especificou no fim da Secção anterior. Essas visualizações, ou transformações, irão sendo actualizadas cada vez que ocorre um evento de edição do documento.

O diagrama da Figura 1 mostra a arquitectura interna do PcWDa, em termos dos blocos presentes e da sua interligação.

A árvore que representa internamente o documento em edição (designada abreviadamente por `RI`) é a componente central da arquitectura, como se detecta na Figura 1.

O objecto responsável pelo suporte à introdução de dados que realiza a leitura da informação introduzida pelo utilizador, o `Input`, vai inserindo as alterações na `RI`.

Por seu lado, o objecto responsável pelo envio de informação que mostra o do documento em cada estado de edição e escreve os avisos e mensagens de erro, o `Output`), trabalha também directamente sobre a `RI`, refrescando a saída cada vez que ocorre uma alteração na `RI`, ou quando o validador é invocado. Este objecto usará duas janelas: uma *janela central* para o documento em edição; e uma *janela de mensagens* para as demais informações sobre a estrutura e os erros.

O `Validador` não é mais do que um *parser* XML que verifica a estrutura actual

[8] Ver, no apêndice A.1, o esquema CSS que está a ser usado para obter esta visualização.

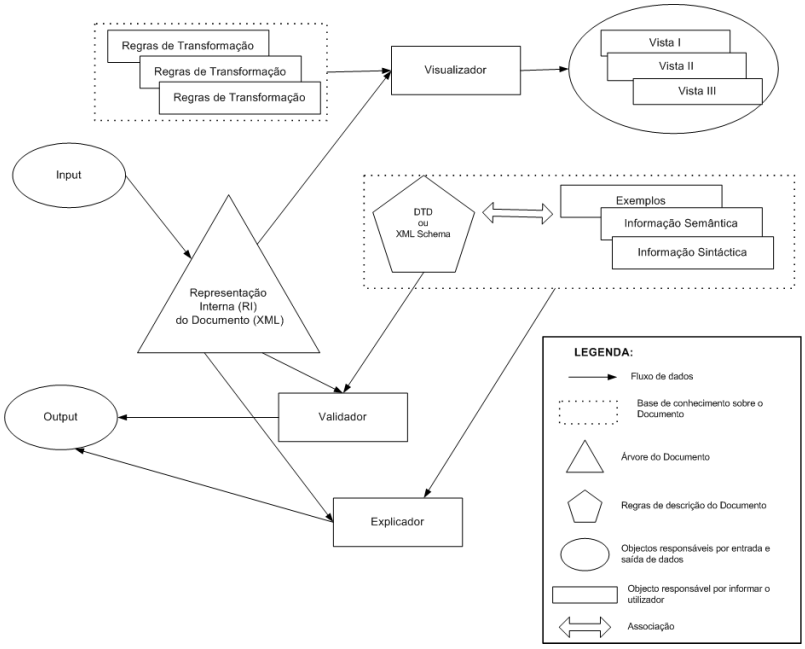


Figura 1. Arquitectura do PcWDa

da `RI`, comparando com a informação fornecida por um outro objecto do sistema que contém o DTD relativo ao tipo do documento de trabalho.
O `Explicador` é mais um objecto da arquitectura, responsável também por comunicar informação ao utilizador do PcWDa, mas desta vez sob pedido e usando uma janela de apresentação distinta das anteriores, a *janela de exemplos*. Para desempenhar a sua tarefa, o `Explicador` recorre à `RI` e ao DTD, para se situar no contexto estrutural, e usa a informação disponibilizada pelos objectos associados que contém as regras semânticas e os exemplos.
Reconhece-se, por fim, na Figura 1 o conjunto de objectos que, detendo informação relativa aos vários esquemas de transformação, são responsáveis pela produção das demais *vistas* sobre a `RI`.

De modo a implementar, com rigor e simplicidade (sistematicamente), o funcionamento requerido seguindo a arquitectura descrita, é necessário trabalhar com um bom modelo de dados.
Para isso, uma vez que se vai usar a linguagem de programação orientada a objectos Java, optámos pela modelação em UML [4], que é actualmente uma das abordagens mais em voga na área de *engenharia de software*. Já na secção 2 nos referimos aos *use-cases*, embora tenhamos ficado pela descrição de um cenário, sem apresentar os diagramas elaborados. Depois, da grande variedade de instrumentos de especificação que o UML oferece, usámos o diagrama de classes para

formalizar a definição das componentes da arquitectura do sistema. Destas duas peças UML derivou-se a implementação Java do PcWDa.

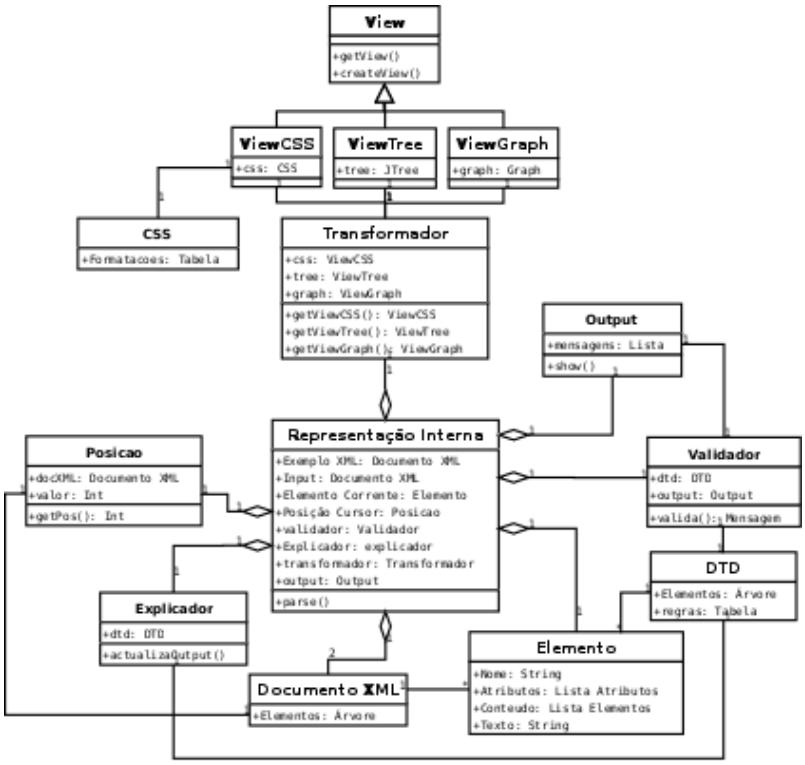


Figura 2. Diagrama de Classes do PcWDa

No diagrama de classes do PcWDa (Ver Figura 2) incluiu-se uma classe para cada uma das oito componentes da arquitectura da Figura 1 de modo a especificar com rigor: a informação associada a cada componente (através dos *atributos da classe*); e a respectiva funcionalidade (através dos *serviços/métodos* disponibilizados). Na próxima secção (ver Figura 4) é apresentado o diagrama com as classes Java que serão usadas.

4 Implementação do PcWDa: tecnologia, desenho e interface

Para a implementação do PcWDa assumiu-se que um ambiente de desenvolvimento orientado aos objectos seria a abordagem ideal. Então, escolheu-se como

base de trabalho, conforme já acima fora anunciado, a linguagem de programação Java [8]; para a camada de interacção gráfica, optou-se por Java Swing [22], sendo o JApplet [22] usado para garantir que o processador vai ser executado a partir de um *browser da Web*.

Decidido o ambiente de programação, era fundamental optar por um formato para representação interna do documento XML a editar. Como seria de imaginar, a escolha recaiu no modelo proposto pelo consórcio W3C, a estrutura em árvore designada por Document Object Model, DOM [21]. Passamos assim a juntar-nos à grande família de programadores que usa esta representação interna normalizada para fácil travessia estrutural dos documentos XML (tratasse na realidade de percorrer uma árvore irregular generalizada), com acesso simples aos elementos da estrutura, aos seus atributos e respectivo conteúdo. Ficou então implícito o recurso à respectiva API, *Application Programming Interface*. Esta interface, que pode ser usada pelos programas em Java, providencia as *estruturas de dados* e os *métodos* necessários para armazenar em memória e manusear os *elementos*, os *atributos*, e o *conteúdo* de um documento XML. Usando a API DOM podemos *criar* a árvore, *mantê-la* (adicionando, modificando ou eliminando elementos e o respectivo conteúdo) e *manipulá-la* (navegando na estrutura para aceder a toda uma sub-árvore ou ao conteúdo de um elemento).

Retomando o **memorando**, introduzido na secção 2 para exemplificar a edição de um documento XML, a sua representação em árvore seria a que se vê na Figura 3. A raiz da árvore é o elemento inicial do DTD, `Memo`, e as folhas são o texto que está associado a cada elemento; de cada nodo intermédio saiem tantos filhos quantos os sub-elementos que o respectivo elemento tiver. A maneira como os nodos e a própria estrutura arbórea são efectivamente implementados em memória não é imposta pela norma DOM, ficando ao cuidado de quem programou a API.

A camada interactiva do PcWDa comunica com o utilizador através de uma interface gráfica, baseada no actual paradigma da *interacção via janelas*, concebida de acordo com as *boas práticas* recomendadas nesta área (veja-se, por exemplo, [18]). Assim a interface da aplicação estará dividida nas seguintes componentes (janelas, ou áreas de trabalho):

Barra de Opções esta barra de topo, contendo botões que assinalam as principais operações disponíveis, corresponde ao Menu Principal do PcWDa;

Barra de Ferramentas a segunda barra no topo contém vários botões que dão acesso directo às operações mais usadas para manipulação do texto;

Área da Estrutura do Documento esta janela, à esquerda, é usada para mostrar a estrutura em árvore do documento XML;

Área de Texto janela central para se escrever e editar os documentos XML;

Área de Sugestão de Elementos janela na base onde surgem as sugestões sobre os *elementos* XML que podem ser inseridos no documento;

Área de Dicas janela, à direita, com sugestões, exemplos e informações semânticas relativas ao contexto do documento;

Área de Visualização janelas, que poderão ser sobrepostas à central, usadas para visualização das diferentes vistas do documento;

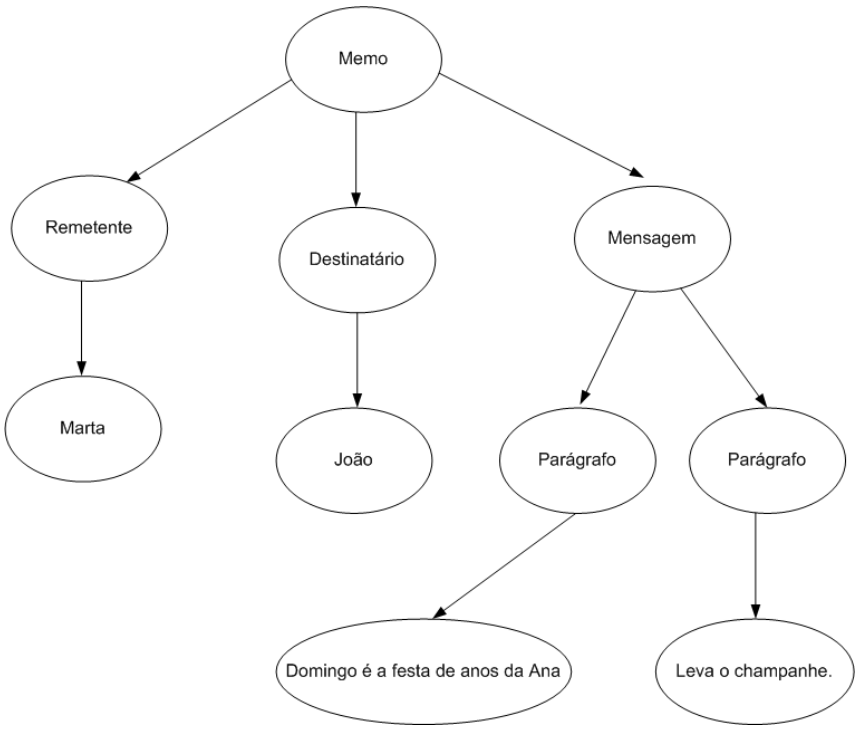


Figura 3. Estrutura em árvore do Memorando XML

Área de Mensagens janela, também à direita, onde se mostrarão as mensagens de erro, sugestões para correcção e outros avisos;

Área do Cursor pequena zona onde se mostra ao utilizador qual a posição actual do cursor.

Tomadas as decisões de base, relativas às ferramentas de implementação e ao formato de representação interna, e estabelecida a organização do écran para interacção, é altura de definir as classes concretas que irão realizar o modelo de classes UML referido na secção 3. A Figura 4 esquematiza a estrutura de classes que se vai usar.

A representação interna do documento (`RI`) corresponde à árvore DOM – Document Object Model [1]) existente na classe `XMLPane`. O `Validador` corresponde ao *parser* que se encarrega da validação, juntamente com o DTD que é suportado pela classe `DocumentElements`, também responsável por ir mostrando ao utilizador os elementos disponíveis para inserção. O `Explicador` é, essencialmente, representado pela classe `XMLExample`, que se encarrega de mostrar, na área própria, um documento XML exemplo e as explicações semânticas dependentes do contexto da edição. As Regras de Transformação, definidas em CSS ou outra linguagem similar, são representadas pela classe `VisualRepresentation`,

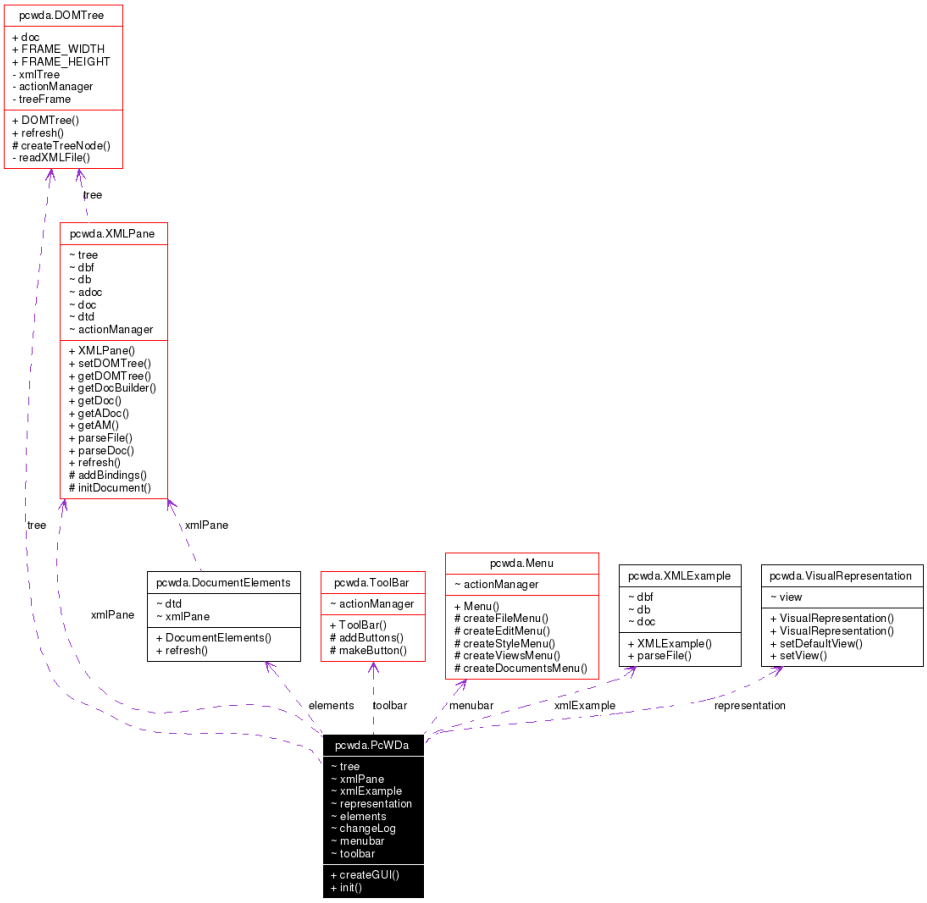


Figura 4. Diagrama de classes Java.

que é responsável por produzir a visualização standard do documento e demais vistas de apresentação ou exportação. O `Output` corresponde à classe `changeLog` que notifica o utilizador de eventuais erros e lhe envia outros avisos referentes ao processo geral de interacção.

Todos os componentes referidos são componentes Swing.

Associando correctamente as funções de entrada/saída (leitura/escrita) de cada um destes objectos com a respectiva janela da interface, obtém-se, finalmente, a ferramenta pretendida.

5 Conclusão

Com o intuito de beneficiar o sítio WWW dinâmico para suporte a disciplinas do grupo de *Introdução à Informática*, está em desenvolvimento uma ferramenta para apoio à unidade de *Processamento de Documentos*. O pacote de software a integrar no sítio—baptizado de PcWDa—deverá estar acessível na Web e ajudar o professor a ensinar a estruturar documentos (usando anotação explícita) como forma alternativa ao tradicional ensino do Word, típico desse grupo de disciplinas.

Discutimos no artigo os cuidados a ter com a sua concepção e desenho, para tornar o PcWDa uma ajuda efectiva, auxiliando o aprendiz a vencer a inércia e o facilitismo e a preocupar-se mais com a estrutura e com o conteúdo do que com o aspecto (o arranjo gráfico). Para tal, falámos nas funcionalidades a incluir no PcWDa e na arquitectura pensada para as realizar.

Na medida em que a ferramenta ainda está em construção, não nos é possível dar conta da sua *performance* nem, tão pouco, avaliar o seu real impacto pedagógico—esses assuntos serão abordados noutros documentos a produzir num futuro próximo. Contudo, também não era essa a nossa intenção; o que pretendemos realçar nesta comunicação foi:

- *os requisitos impostos pelo modelo de aprendizagem escolhido*, contrariando a ideia de que a aplicação é que impõe, pela tecnologia empregue, o método ou o ritmo de aprendizagem;
- *a abordagem ao problema, para desenvolver com rigor um processador que satisfizesse esses requisitos*, modelando em UML os cenários de utilização e as componentes da arquitectura do sistema;
- *a tecnologia adoptada para implementar com eficiência esse processador*, usando Java e a sua vasta colecção de bibliotecas para implementar directamente o modelo construído minimizando o esforço e o tempo dispendidos.

Referências

1. Kal Ahmed, Sudhir Ancha, Andrei Cioroianu, Jay Cousins, Jeremy Crosbie, John Davies, Kyle Gabhart, Steve Gould, Ramnivas Laddad, Sing Li, Brendan Macmillan, Daniel Rivers-Moore, Judy Skubal, Karli Watson, Scott Williams, and James Hart. *The JFC Swing Tutorial*. Wrox Press, 2001.
2. Altova. Altova Xml Spy 2005. `http://www.altova.com/products_ide.html`, 2005.
3. David P. Ausubel. *Educational Psychology, A Cognitive View*. New York: Holt, Rinehart and Winston, Inc, 1968.
4. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley, 1999.
5. Bert Bos. Cascading Style Sheets home page. `http://www.w3.org/Style/CSS`, 2002.
6. Neil Bradley. *The XML Companion*. Addison-Wesley, 3rd edition, 2002.
7. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML). World Wide Web Consortium, October, 2000. `http://www.w3.org/TR/REC-xml`.

8. Mary Campione and Kathy Walrath. *The Java Tutorial*. Amazon, 1995-2004.
9. Andy Clark. CyberNeko DTD Converter. `http://www.apache.org~andyc/neko/neko/doc/dtd/`, 2005.
10. Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Nikola Ozu, Ian Stokes-Rees, Jeni Tennison, Kevin Williams, and Kurt Cagle. *Professional XML Schemas*. Wrox Press, 2001.
11. Charles F. Goldfarb and Paul Prescod. *XML Handbook*. Prentice Hall, 4th edition, 2001.
12. G. Ken Holman. *Definitive XSLT and XPath*. Prentice Hall, 2001.
13. Eve Maler and Jeanne Andaloussi. *Developing SGML DTDs: From Text to Model to Markup*. Prentice-Hall, 1996.
14. Laurent Mignet, Denilson Barbosa, and Pierangelo Veltri. The XML Web: a first study. 2003.
15. Jorge Pinto. *Psicologia da Aprendizagem: concepção, teoria e processos*. Instituto do Emprego e Formação Profissional, 1996.
16. Ricardo Alexandre Peixoto Queirós. Edição estrutural de documentos XML sobre a Web. Master thesis, Faculdade de Ciências da Universidade do Porto, 2004.
17. José Carlos Ramalho and Pedro Henriques. *XML & XSL Da Teoria à Prática*. FCA Editora, 2002.
18. Edla Ramos. O Fundamental na Avaliação da Qualidade do Software Educacional, Ago, 2003. `http://www.hipernet.ufsc.br/foruns/ine/docentes/qualid.doc`.
19. Jacqueline Fátima Teixeira. Uma Discussão sobre a Classificação de Software Educacional, Jun, 2004. `http://www.revista.unicamp.br/infotec/artigos/jacqueline.html`.
20. Henry Thompson. The Extensible Stylesheet Language (XSL) Family. `http://www.w3.org/Style/XSL/`, 2003.
21. W3C. DOM – Document Object Model. `http://www.w3.org/DOM/`, 2005. World Wide Web Consortium.
22. Kathy Walrath, Mary Campione, Alison Huml, and Sharon Zakhour. *The JFC Swing Tutorial*. Amazon, 1995-2004.

A Exemplo do Tipo de Documento Memorando

Neste apêndice, incluem-se algumas informações que completam o exemplo usado no cenário descrito na secção 2, relativas à edição de um documento XML da família dos Memorandos.
Começa-se com uma instância dessa família, ou seja, com um memorando concreto; depois mostra-se o DTD que estabelece a anotação a usar para esse tipo; termina-se com a folha de estilos CSS que se usou para gerar uma vista do documento.

A.1 Documento XML

```
<Memo>
<Remetente>Marta</Remetente>
<Destinatario>João</Destinatario>
<Texto>
```

```
<Paragrafo>Domingo é os anos da Ana.</Paragrafo>
<Paragrafo>Leva o champanhe!</Paragrafo>
</Texto>
</Memo>
```

A.2 Documento DTD

```
<!ELEMENT Memo (Remetente, Destinatario, Texto)>
<!ELEMENT Remetente (#PCDATA)>
<!ELEMENT Destinatario (#PCDATA)>
<!ELEMENT Texto (Paragrafo)+>
<!ELEMENT Paragrafo (#PCDATA)>
```

A.3 Documento CSS

```
$DOCUMENT {
font-family: "Times New Roman";
font-size: 12pt;
margin-top: 5px;
margin-left: 5px;
}

Remetente, Destinatario {
display: block;
font-size: large;
}

Texto {
display: block;
font-size: small;
}

Paragrafo {
font-style: italic;
color: blue;
}
```

HotSpotter: a JavaML-based approach to discover Framework's HotSpots

Nuno Flores , Diana Soares, Helder Ferreira, Marco Rodrigues

Faculdade de Engenharia Universidade do Porto
R. Dr. Roberto Frias s/n,
4200-465 Porto, Portugal
{mei03009, mei03014, mei03001, mei03016}@fe.up.pt

Abstract. Object-oriented frameworks are designed with reusability as the main design concern. In order to develop concrete solutions by customizing existing frameworks, developers must be aware of its areas of flexibility, which are commonly called hot-spots. A common problem with framework reuse is the extension and detail of existing documentation, often resulting insufficient to communicate the level of knowledge the developer needs to be effective in reusing the framework. This paper addresses this problem by providing a tool that detects those hot-spots, called the HotSpotter, thus enabling the developer to rapidly identify where is possible to customize the framework in order to obtain the desired solution. The HotSpotter is a tool targeted for frameworks written in Java, and it follows a multi-phased process that starts from a JavaML base representation of the framework's source-code and evolves through a series of XSL transformations until reaching the desired results. This approach intends to uncover the hot-spots of frameworks or applications, by identifying templates and hooks and then grouping them using predefined heuristics. The results obtained with the XML-based version of the HotSpotter tool was compared and validated against those of a similar existing tool, using the JUnit framework as a case-study.

1 Introduction

Object-oriented application frameworks are intended to provide software for reuse [4]. Their generic design within a given domain and its reusable implementation spurs rapid development of software solutions. Despite this great potential for reuse, its practical feasibility can only be reached if the framework design becomes thoroughly understood and its extensibility parts are clearly recognizable. The complexity behind the design and implementation of the framework makes it difficult to grasp its flexibility, forcing a good documentation support vital. Most often, the existing documentation lacks in extent and detail and is insufficient to make the developer aware of its flexibility.

This paper addresses the need to such awareness by providing a tool that detects those areas of flexibility, called the hotspots, thus enabling the developer to rapidly identify where to act upon, configuring the framework to evolve to the desired solu-

tion. Due to its potentially large size and complexity, the ability to quickly understand and apply a framework is a critical issue [4] in software development.

The HotSpotter is a tool targeted for frameworks written in Java and which goal is to detect and show the framework's hot-spots. It follows a multi-phased process that starts from a JavaML base representation of the framework's source-code and evolves through a series of XSL transformations until reaching the desired results. This approach intends to uncover the hot-spots of frameworks or applications, by identifying templates and hooks and then grouping those using predefined heuristics.

The development of this approach was part of a dual-approach-based project using JavaML [6] and Eclipse [2] in parallel. The results obtained with the XML-based version of the HotSpotter tool were compared and validated against those of the similar Eclipse tool, using the JUnit [3] framework as a case-study.

Section 2 gives a brief explanation on the concepts of hotspots, template and hook methods and their relationships. In order to maintain expressiveness, a few considerations had to be made and specific domain definitions were adopted.

Section 3 describes the goals and the development methodology: a multi-step process aiming at detecting and recognizing the relevant reusability structures and the appropriate way of aggregating the results into two views for broader understanding.

In Section 4, all the process steps are explained in detail. Beginning at the JavaML source code representation, several XSL transformations are applied accordingly to achieve the desired outcome.

Section 5 illustrates some results after applying the process to the JUnit framework, whereas Section 6 presents the conclusions and the forthcoming work.

2 HotSpots, Template Methods and Hooks Relationships

As a concept, a hotspot has its foundations built upon a framework's Open-Closed Principle [15]. This attribute describes the framework's potential flexibility and proneness to reuse. The principle encompasses two definitions: the "closed" and the "open" parts. The latter represents the areas that are variant, configurable and redefinable, whereas the former describes those areas that remain immutable, yet group the variant parts together [14].

This combination of open and closed parts embodies the framework's reusability regions, the so called hotspots. The notions of *template methods* and *hook methods* support adherence to the Open-Closed Principle as they materialize the concepts of "open" and "closed". A template ("closed") method defines an invariant part of the framework which links ("calls") the hook ("open") methods together. These hook methods are the re-definable elements to where the user should aim at, thus adapting the framework to provide a particular solution.

Cardino [1] states that "a reuser should not have to worry […], since hotspots should be sufficient to customize the framework to new needs". That is to say that the awareness and understanding of hotspots is crucial to exploit the extensibility aspects of a framework. Identifying these extensibility regions heightens the development speed and improves the quality of the process [4].

According to [14], the relationship between templates and hooks translates into what is called *meta-patterns*. These relationships can be observed through the analysis of several design patterns [13]. Knowing the composing patterns of a framework's design enables the assertion of its degree of reusability. Starting from a lower level of abstraction, one must first identify the templates and hooks contained in a framework and then identify its relationships. Albeit seven known meta-patterns exist, as far as hotspots are concerned, template methods and hook methods can be combined by either inheritance or composition [12]:

1. Through Inheritance, abstract hook methods, defined in the same class as the template method, are overridden in descendant classes (Fig. 1).
2. Through Composition, hook methods are defined in interface classes which are subsequently implemented by one or more concrete classes. The template class calls the hook methods defined in the interface, not knowing which concrete class is implementing its behaviour. (Fig. 2). Many GoF design patterns base their flexibility on this concept, such as Builder, Strategy or Bridge [13].

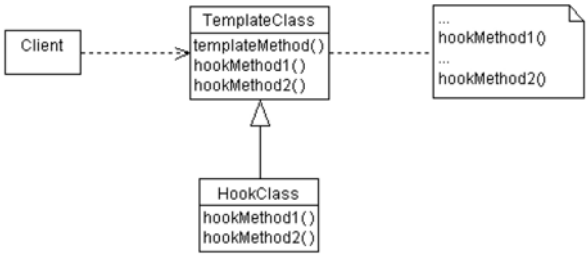


Fig. 1. Inheritance Template Method (IHS).

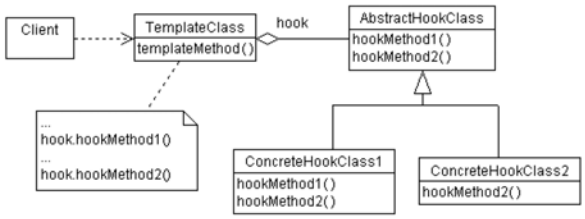


Fig. 2. Composition Template Method (CHS).

Upon template and hook detection and their relationship, one can group them into hotspots. This raises a concern: expressiveness. This issue is addressed by [12] which come up with a definition of hotspot, neither too fine-grained nor too course-grained:

[a hotspot is] a set of hook methods and their associated template methods, in which each hook method is invoked by exactly the same set of template methods.

From this definition of a hotspot (Fig. 3), Schauer [12] divides them into two sub-categories: IHS (Inheritance hotspots) and CHS (Composition hotspots). IHS stands for hotspots exclusively based on Inheritance Template Methods (Fig. 1), whereas CHS stands for hotspots exclusively based on Composition Template Methods (Fig. 2).

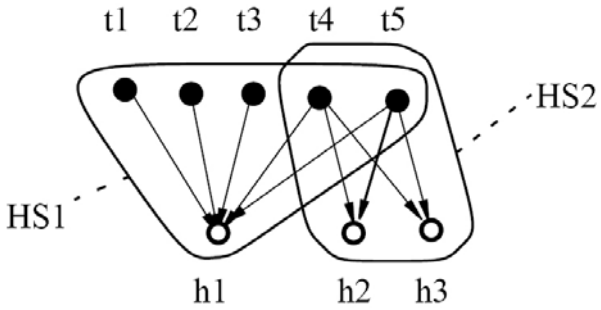


Fig. 3 Hot Spot definition, according to [12].

3 Goals and development methodology

The process of developing a tool for detecting and visualizing hotspots undertook three successive steps:

1. *Source Code Parsing*. The primary step would be to parse through the source code and identify the relevant elements, that is, candidate methods and their enclosing classes that might be classified as templates or hooks.

2. *Template/Hook detection heuristics*. Upon gathering the filtered source code elements, its structure and relationships would have to be confronted against the definitions of templates and hooks. Namely, methods that call other methods defined in descendant classes (IHS approach) or in interface classes (CHS approach) would be scrutinized.

3. *Aggregate into hotspots and generate results*. Coupling templates and hooks would provide candidate hotspots. In order to conform to the pre-defined hotspot definition, an aggregation step would have to take place to broaden the scope of the detected areas of flexibility. Without loss of information, the collapsing of contiguous candidate hotspots into a single element would define wider regions of reuse. Consequentially, granularity would decrease and a better vision of the overall flexibility would be easily acquired.

While detecting templates and hooks, two views emerged from the progressing development and occurring results. Amidst the process, it made sense to come up with two kinds of aggregation heuristics for candidate hotspots:

1. "Vista1" (View1). *Each template method calls exactly the same hook methods, grouping by the higher number of hook methods as possible*. That is, templates and hooks are joined together into a container-content structure. A template "contains" (calls) a certain number of hooks. Aggregation of the final hotspots is then made by

intersecting the template methods through their hook set. The new hotspot is generated by maximizing the number of hooks methods intersected.

2. "Vista2" (View2). *Each hook method is called by exactly the same template methods*. In this case, templates and hooks are joined together into a container-content structure, having the hook as the container. The hook "contains" the templates which call it. Aggregation of the final hotspots is made by intersecting the hooks through their templates set. The new hotspot is generated by grouping only those hooks that have exactly the same templates as callers.

In the next section, a further insight of the JavaML approach will enlighten the purpose behind these views.

4 JavaML Approach

The lack of a canonical structured representation of the java source code, and its convenient plain-text representation to the programmers induced the need for a universal format, able to directly represent the program structure and its contents. This transformation would supply other software tools with an easy platform for source code analysis and manipulation. XML [5] presented itself as a good solution and led to the creation of JavaML [6] [7]. The work presented in this paper followed a JavaML approach, as it easily allows the manipulation of Java source code, providing a simple and fast way of getting results.

4.1 System's Overview

As mentioned before (see abstract) the JUnit framework was used as a case-study for HotSpotter. The first step in the JavaML approach consisted in the transformation of the JUnit java source code into JavaML files.

Extract of junit-all.java.xsl (abstract method declaration)

```
  <method name="testEnded" id="Ljunit/runner/BaseTestRunner;testEnded(Ljava/lang/String;)V"
idkind="method">
    <modifiers>
      <modifier name="public"/>
      <modifier name="abstract"/>
    </modifiers>
    <type name="void" primitive="true"/>
    <formal-arguments>
      <formal-argument name="testName" id="Ljunit/runner/BaseTestRunner;arg391" idkind="formal">
        <type name="String" idref="Ljava/lang/String;" idkind="type"/>
      </formal-argument>
    </formal-arguments>
  </method>
```

Extract of junit-all.java.xsl (hook call by template method)

```
  <method name="endTest" id="Ljunit/runner/BaseTestRunner;endTest(Ljunit/framework/Test;)V"
idkind="method">
    <modifiers>
      <modifier name="public"/>
      <modifier name="synchronized"/>
    </modifiers>
    <type name="void" primitive="true"/>
    <formal-arguments>
      <formal-argument name="test" id="Ljunit/runner/BaseTestRunner;arg338" idkind="formal">
        <type name="Test" idref="Ljunit/framework/Test;" idkind="type"/>
      </formal-argument>
    </formal-arguments>
```

```
    <block>
      <send message="testEnded"
idref="Ljunit/runner/BaseTestRunner;testEnded(Ljava/lang/String;)V" idkind="method">
        <arguments>
          <send message="toString" idref="Ljava/lang/Object;toString()Ljava/lang/String;"
idkind="method">
            <target>
              <formal-ref name="test" idref="Ljunit/runner/BaseTestRunner;arg338"
idkind="formal"/>
            </target>
          <arguments/>
        </send>
      </arguments>
    </send>
    </block>
  </method>
```

The transformation was performed with the IBM Jikes Java Compiler [8] complemented with the JavaML Patch for Jikes [9]. Next, the XML files generated were packed into one single JavaML file with the root element `<javaml>`. Afterwards, several XSL transformations [10] for both IHS and CHS methods and both "Vista1" and "Vista2" views were applied, producing several XML outputs. Each of these last XML outputs were then transformed by a PHP [11] script that grouped the hooks and templates, producing a single final HTML presentation file containing the Hotspots found in the JUnit framework. Fig. 4 shows the overall structure of this process.

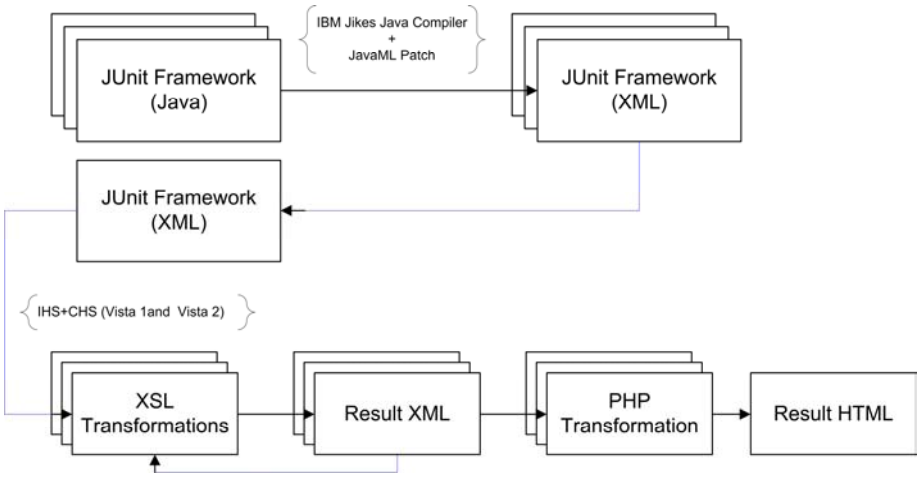


Fig. 4. System Overview of JavaML approach.

4.2 XSL Transformations

Several XSL files were developed with the purpose of extracting relevant information from the JavaML files. Each method (IHS and CHS) used the appropriate XSL.

For the IHS method the stylesheets do the following:

1. *Capture of Hooks and Templates*: templates are captured by searching the JavaML file for the `<method>` tag with the *abstract* attribute. Hooks are captured by searching for the `<send>` tag with a *message* attribute that refers the called method.

ihsa-vista1-getHooksandTemplates.xsl

```
<xsl:template match="javaml">
  <Java>
    <xsl:apply-templates select="//method/modifiers/modifier[@name='abstract']"/>
  </Java>
</xsl:template>

<xsl:template match="modifier">
  <xsl:param name="hookID" select="../../@id"/>
  <hook>
    <xsl:attribute name="name"><xsl:value-of select="../../@name"/></xsl:attribute>
    <xsl:attribute name="id"><xsl:value-of select="../../@id"/></xsl:attribute>
    <xsl:if test="ancestor::class/@name">
      <xsl:attribute name="class"><xsl:value-of select="ancestor::class/@name"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="ancestor::interface/@name">
      <xsl:attribute name="class"><xsl:value-of select="ancestor::interface/@name"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="//send[@idref=$hookID]"/>
  </hook>
</xsl:template>

<xsl:template match="send">
  <template>
    <xsl:attribute name="name"><xsl:value-of select="ancestor::method/@name"/></xsl:attribute>
    <xsl:attribute name="id"><xsl:value-of select="ancestor::method/@id"/></xsl:attribute>
    <xsl:attribute name="class"><xsl:value-of select="ancestor::class/@name"/></xsl:attribute>
  </template>
</xsl:template>
```

2. *Invert the position of Hooks and Templates*: this is only required for "Vista1" view and it concerns grouping hooks inside a template, and not the contrary.

For the CHS method:

1. *Capture of all abstract methods, all methods declared in interfaces and all variables used inside those methods*: the methods are captured in a similar way as in IHS; methods in interfaces are captured by searching the method declaration inside an `<interface>` tag and variables are captured by searching the tags `<field>`, `<local-variable-decl>` and `<formal-argument>` inside the captured methods.

chsa-vista1-getHooksandTemplates.xsl

```
<!-- Variables -->
<xsl:template match="field">
  <field>
    <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    <xsl:attribute name="class"><xsl:value-of select="type/@name"/></xsl:attribute>
  </field>
</xsl:template>
<xsl:template match="local-variable-decl">
  <field>
    <xsl:attribute name="id"><xsl:value-of select="local-variable/@id"/></xsl:attribute>
    <xsl:attribute name="class"><xsl:value-of select="type/@name"/></xsl:attribute>
  </field>
</xsl:template>
<xsl:template match="formal-argument">
  <field>
    <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    <xsl:attribute name="class"><xsl:value-of select="type/@name"/></xsl:attribute>
  </field>
</xsl:template>
<!-- Abstract methods -->
<xsl:template match="modifier">
  <xsl:param name="MethodId" select="../../@id"/>
  <xsl:for-each select="//send[@idref=$MethodId]">
    <hook>
      <xsl:attribute name="name"><xsl:value-of select="@message"/></xsl:attribute>
      <xsl:attribute name="id"><xsl:value-of select="@idref"/></xsl:attribute>
      <xsl:if test="descendant::field-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::field-ref/@idref"/>
        </xsl:attribute>
        <xsl:attribute name="classtype">varclass</xsl:attribute>
      </xsl:if>
      <xsl:if test="descendant::var-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::var-ref/@idref"/>
        </xsl:attribute>
```

```
        <xsl:attribute name="classtype">varlocal</xsl:attribute>
      </xsl:if>
      <xsl:if test="descendant::formal-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::formal-ref/@idref"/>
        </xsl:attribute>
        <xsl:attribute name="classtype">varlocal</xsl:attribute>
      </xsl:if>
      <template>
        <xsl:attribute name="name"><xsl:value-of select="ancestor::method/@name"/>
        </xsl:attribute>
        <xsl:attribute name="id"><xsl:value-of select="ancestor::method/@id"/></xsl:attribute>
        <xsl:attribute name="class"><xsl:value-of select="ancestor::class/@id"/>
        </xsl:attribute>
      </template>
    </hook>
  </xsl:for-each>
</xsl:template>
<!-- Interface methods -->
<xsl:template match="method">
  <xsl:param name="MethodId" select="@id"/>
  <xsl:for-each select="//send[@idref=$MethodId]">
    <hook>
      <xsl:attribute name="name"><xsl:value-of select="@message"/></xsl:attribute>
      <xsl:attribute name="id"><xsl:value-of select="@idref"/></xsl:attribute>
      <xsl:if test="descendant::field-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::field-ref/@idref"/>
        </xsl:attribute>
        <xsl:attribute name="classtype">varclass</xsl:attribute>
      </xsl:if>
      <xsl:if test="descendant::var-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::var-ref/@idref"/>
        </xsl:attribute>
        <xsl:attribute name="classtype">varlocal</xsl:attribute>
      </xsl:if>
      <xsl:if test="descendant::formal-ref/@idref">
        <xsl:attribute name="classid"><xsl:value-of select="descendant::formal-ref/@idref"/>
        </xsl:attribute>
        <xsl:attribute name="classtype">varlocal</xsl:attribute>
      </xsl:if>
      <template>
        <xsl:if test="ancestor::method/@name">
          <xsl:attribute name="name"><xsl:value-of select="ancestor::method/@name"/>
          </xsl:attribute>
          <xsl:attribute name="id"><xsl:value-of select="ancestor::method/@id"/>
          </xsl:attribute>
          <xsl:attribute name="class"><xsl:value-of select="ancestor::class/@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:if test="not(ancestor::method/@name)">
          <xsl:attribute name="name"><xsl:value-of select="ancestor::constructor/@name"/>
          </xsl:attribute>
          <xsl:attribute name="id"><xsl:value-of select="ancestor::constructor/@id"/>
          </xsl:attribute>
          <xsl:attribute name="class"><xsl:value-of select="ancestor::class/@id"/>
          </xsl:attribute>
        </xsl:if>
      </template>
    </hook>
  </xsl:for-each>
</xsl:template>
```

2. *Capture the Classes where methods are declared*: this is done by searching for the *idref* attribute of the called method tag which relates to the variables declaration *id* attribute.

3. *Invert the position of Hooks and Templates*: this is only required for "Vista1" view and its purpose is similar to the explained before in the IHS method.

4.3 Hotspot Detection

After applying all the XSL transformations, the XML output result file is processed by a PHP script that groups hooks or templates (depending whether its "Vista1" or "Vista2" views) and detects possible hotspots, outputting them into a HTML file.

In both views, the flow and set of activities are the same:

1. Read the file;
2. Parse the XML structure;
3. Find and group the hotspots;
4. Show the results.

Reading the file and showing the results is trivial. A brief description of activities 2 and 3 follows:

Vista1. The XML structure is parsed into a hash container. Whenever a new template is found, a new entry is inserted into the hash, with the template *id* being the key and an empty array as its value. The template "hook method" *id*s will then be added into this array.

At the end, a hash is available with all template *id*s as keys and the corresponding hook *id*s as its values.

Next, an attempt is made to find and group the hotspots. For each template method *id* in the hash, its hook methods are intersected with the remaining ones from the other templates in the hash. If the result of this intersection is not empty, it will be the key of a new hash with its value being an array with the corresponding templates. If that hash key already exists, the template *id*s are added to the already existing array of templates.

At the end, a new hash is available containing the candidate hotspots. Its keys are a set of hooks, and its values the templates who call them.

Vista2. In this view, the document is parsed with the same method as in "Vista1", with one slight difference: since the XML structure was swapped to a "hook to template" hierarchy, the resulting hash will contain all hook *id*s as keys and its values being the template methods which call them.

Finally, to find and group the hotspots, the template *id*s stored in the first hash are used as the key to a new hash of candidate hotspots. Its corresponding value is an array with the hook methods which are called by exactly those templates.

5 Some Results

Figure 5 illustrates the results of applying the "Vista2" view to the JUnit framework. In comparison with the similar tool developed for Eclipse, both "Vista1" and "Vista2" produced perfect matches, despite a few glitches in "Vista1", but of no relevancy to the overall goal. This comparison served to prove that the proposed solution produced accurate results and that the pre-defined heuristics for hotspot detection were indeed valid.

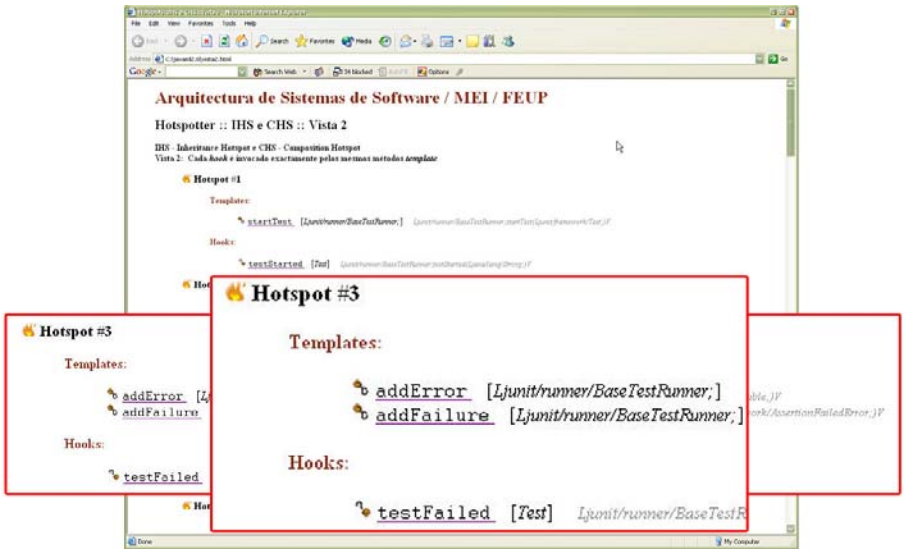


Fig. 5. Vista2 HTML generated results.

6 Conclusions and Future Work

This paper presented a JavaML-based approach to detect hotspots using successive XSL transformations over a multi-phased evaluation process. Using the JUnit framework as a case study, the results were confronted against a similar tool for Eclipse, proving the efficiency of the approach and validating its results. Despite the possibility of performing all the transformation steps using XSL, a final step was made using PHP. This decision had to do with project development schedule restrictions only, and not technological issues regarding XSL. At the time of development, PHP served as a quicker developing platform in order to uphold the project's deadline. At the writing of this article, the PHP step was being converted into XSL, with satisfying results. As future work, this approach could be extended to include the following features:

- Convert the PHP step to XSL (currently underway).
- Detect all seven relationships between templates and hooks (meta-patterns).
- Extend the JavaML notation to include "flexibility-oriented" tags, thus enriching the code representation.
- Infer what kind of design patterns can be found on the code with a degree of certainty.

References

1. Guido Cardino *et al. "The Evaluation of Framework Reusability".* ACM SIGAPP. Volume 5, Issue 2. September 1997
2. Eclipse website - *http://www.eclipse.org* .
3. JUnit website - *http://www.junit.org*.
4. Garry Froelich et al. *"Hooking into Object-Oriented Application Frameworks".* 19th International Conference on Software Engineering. 1997.
5. XML – eXtensible Markup Language (website). *http://www.w3.org/XML/*.
6. Greg Badros. "*JavaML: A Markup Language for Java Source Code*". 9th International World Wide Web Conference. 2000.
7. Ademar Aguiar et al. *"JavaML 2.0: Enriching the Markup Language for Java Source Code"*. XATA. 2004.
8. IBM Jikes Java Compiler - *http://www-124.ibm.com/developerworks/opensource/jikes/.*
9. JavaML Patch for Jikes website- *http://www.cs.washington.edu/homes/gjb/JavaML/.*
10. XSL website- *http://www.w3.org/Style/XSL/*.
11. PHP website - *http://www.php.net/*.
12. Reinhard Schauer. *"Hotspot Recovery in Object-Oriented Software with Inheritance and Composition Template Methods"*. ICSM. 1999.
13. Erich Gamma et al. *"Design Patterns: Elements of Reusable Object-Oriented Software"*. Addison-Wesley. 1995.
14. Wolfgang Pree. *"Design patterns for object-oriented software development"*. Addison-Wesley. 1995.
15. Martin, R. *"The Open Closed Principle"*, C++ Report, 1996

GerExa: Plataforma Integrada para a Organização, Geração e Avaliacão de Exercícios e Testes

Ângela Oliveira, Escola Superior de Tecnologias, Castelo Branco
email:angelaoliveira@est.ipcb.pt
Nelma Moreira, DCC-FC& LIACC, Universidade do Porto
email:nam@ncc.up.pt

Resumo Com o advento da sociedade digital aparecem novos paradigmas na área da educação. É fundamental conceber soluções de e-Learning que flexibilizem o acesso aos recursos de aprendizagem. Assim, neste trabalho, apresenta-se o desenho duma plataforma que assenta nos princípios básicos de desenvolvimento das aplicações de e-Learning, mas com agregação de funcionalidades que permitirão aos professores manter a sua informação organizada e disponível. Para tal definiu-se uma linguagem XML para a manipulação de exercícios e de testes, que serve de base para o desenvolvimento duma aplicação que permita aos professores a edição de exercícios de diferentes disciplinas, para posterior elaboração, aleatória ou assistida, de documentos de apoio às aulas e de avaliação, e sua resolução e correcção. Também os alunos poderão, gerar testes de treino e de avaliação. Pretende-se ainda que a maior parte dos exercícios sejam de correcção automática, para permitir uma avaliação interactiva e mais frequente.

Palavras-Chave: e-Learning, XML, geração de testes

1 Introdução

O XML (eXtensible Markup Language) é uma linguagem para representação de informação estruturada e uma das suas grandes vantagens é o facto de ser independente de ferramentas e plataformas [10,3]. Hoje em dia, e devido ao grande volume de informação e às inúmeras aplicações que manipulam documentos de texto, surge a necessidade de utilizar uma linguagem que de forma simples permita a sua manipulação e armazenamento uniformizados, tendo como base os seus conteúdos. Precisamente, o XML é um padrão aberto que fornece um conjunto de regras para descrever o conteúdo de documentos, bem como a sua estrutura lógica, de modo a que possam ser interpretados e/ou manipulados, quer por diferentes utilizadores, quer por diferentes aplicações [14].

Neste artigo descreve-se a modulação da informação em XML para uma plataforma integrada para gerar e organizar documentos de apoio ao ensino,

tais como fichas de trabalho e testes, que posteriormente, serão resolvidos e corrigidos. Na secção seguinte apresentamos a motivação do desenvolvimento de uma aplicação deste tipo. Na Secção 3 fazemos uma análise da informação que se pretende tratar. A Secção 4 é constituída pela descrição da linguagem XML que denominamos `GerExa` e que define a estrutura da informação. Na Secção 5 é apresentado o desenho da plataforma GerExa, actualmente em desenvolvimento. Alguns comentários finais são discutidos na Secção 6.

2 Motivação

Os professores para o exercício da sua função de ensino, manipulam uma vasta informação, fazendo uso de documentos estruturados, nomeadamente documentos para o apoio às aulas (por exemplo, fichas de exercícios) e documentos de avaliação. Para que os alunos possam ter acesso a uma diversidade considerável de exercícios, o professor necessita de reunir, muitas vezes durante anos, um vasto conjunto de material, que na maioria das vezes se encontra organizado em documentos distintos e dispersos.

Tendo em conta a evolução do ensino e a motivação dos alunos para o recurso a novas tecnologias, nomeadamente a tecnologia WEB, é fundamental o professor evoluir também nesse sentido, disponibilizando informação nesse tipo de ferramentas. Assim surge a proposta de organização da informação que diz respeito ao processo de elaboração de fichas de exercícios e testes de avaliação formativa e sumativa. Estes três tipos de documentos vão conter em comum exercícios, que se encontram organizados por conteúdo, num primeiro nível, estando os conteúdos organizados segundo módulos e estes segundo disciplinas. Os aspectos mais relevantes desta informação prendem-se com os exercícios e testes, uma vez que a ideia subjacente é o desenvolvimento de uma aplicação, que permita aos professores guardarem todos os exercícios de uma determinada disciplina, de forma organizada, para posteriormente poderem produzir os diversos elementos de apoio às aulas e de avaliação, através de geração aleatória ou assistida, desses mesmos documentos. De forma semelhante os alunos poderão, gerar testes de treino, bem como responder a testes de avaliação produzidos pelo professor. Para que seja possível uma avaliação interactiva e mais momentos de avaliação, pretende-se que a maior parte dos exercícios permitam correção automática. Por isso, e pelo menos numa primeira fase, serão construídos exercícios de resposta simples, p.e, de escolha múltipla ou resposta calculada.

Com o advento da sociedade digital aparecem novos paradigmas na área da educação e da formação, tal como a formação ao longo da vida, como factor fundamental na estabilidade do indivíduo no mercado de trabalho. É fundamental conceber soluções de e-Learning que flexibilizem o acesso aos recursos de aprendizagem implantando estratégias pedagógicas adequadas a uma aprendizagem mais eficaz [13]. A plataforma GerExa aqui apresentada assenta nos princípios básicos de desenvolvimento das aplicações de e-Learning, mas com uma visão alternativa mais vocacionada para as necessidades do ensino tradicional, ou seja, com a agregação de funcionalidades que permitirão aos docentes

manter a sua informação organizada e sempre disponível. Neste caso, organização da informação, prende-se com a implementação de uma estrutura que permita guardar os exercícios e os testes de forma simples e de fácil acesso, aspectos que se encontram no XML como alternativa às tradicionais bases de dados que são usadas na maior parte das plataformas de e-Learning, por exemplo, NetSupport School© [5], Advanced e-Learning Builder© [2] e FORMARE© [6] e plataformas que implementam geração de testes. Também existem já plataformas de geração de testes que usam XML, como é o caso da IMS Global Learning Consortium, Inc. © [4], cujo software proprietário implementa uma plataforma de e-Learning. Uma das características do GerExa é precisamente ser desenvolvido usando especificações abertas, independente da plataforma e software livre, e ainda, ir estar disponível sob uma licença GPL (GNU Public Licence).

3 Análise da Informação

A informação base a manipular será a dos exercícios. Os testes, fichas de trabalho e resoluções dos alunos serão constituídas por conjuntos de exercícios com diversos tipos de conteúdos (e/ou respostas dos alunos, métodos de resolução, cotações, etc). Os exercícios para além de informações referentes às disciplinas (módulos e conteúdos, docente, etc.), diferem essencialmente no tipo. Os tipos a considerar são os seguintes:

1. **Escolha múltipla**
   Exemplo de um exercício da disciplina de Programação:
   Escolha a resposta correcta:
   Em C um programa começa com a função:
   a) `return()`
   b) `#include`
   c) `void()`
   d) `main()`
   e) `#declare`
2. **Resposta múltipla**
   Exemplo de um exercício da disciplina de Geografia:
   Seleccione a(s) resposta(s) correctas:
   São cidades de Portugal:
   a) Porto
   b) Salamanca
   c) Madrid
   d) Paris
   e) Faro
   f) Guarda
   g) Vigo
   h) Guimarães.
3. **Correspondência de colunas**
   Exemplo de um exercício da disciplina Algoritmos:

Estabeleça a correspondência de colunas.
Qual a ordem de cada uma das funções:

Coluna 1	**Coluna 2**
1. $n \log n$	1. $O(n)$
2. $2n^2 + n$	2. $O(1)$
3. c, c constante	3. $O(n^2)$

Coluna1	Coluna2

4. **Verdadeiro ou Falso**
   Exemplo de um exercício de Matemática Discreta:
   Sendo A e B conjuntos, é verdade que $\overline{A \cap B} = \overline{A} \cup \overline{B}$?
5. **Resposta calculada**
   Exemplo de um exercício da disciplina Álgebra:
   Considere a matriz $M = \begin{pmatrix} 1\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \end{pmatrix}$
   Calcule o valor próprio dominante da matriz M.
6. **Desenvolvimento**
   Exemplo de um exercício da disciplina Análise Numérica:
   Considere a matriz $A = \begin{bmatrix} 4\,1\,2 \\ 0\,1\,0 \\ 8\,4\,5 \end{bmatrix}$ e o vector $b = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$
   Resolva o sistema $Ax = b$, utilizando o Método de Eliminação de Gauss e Factorizarão LU.

Como se pode observar pelos exemplos, pretende-se que nos enunciados e resoluções dos exercícios se possam incluir elementos como fórmulas matemáticas, imagens ou (fragmentos de) programas de linguagens de programação.

Para cada tipo de exercício é necessário ainda definir a informação que permitirá a correção automática, isto é, como é representada a sua solução, a resolução do aluno e como será (eventualmente) calculada a cotação. Os casos mais complexos, são obviamente para os exercícios de **resposta calculada**. Aqui, podem-se englobar exercícios mais simples em que a resposta é só um valor, até exercícios que envolvam a comparação de dois programas de computador. Para estes exercícios poder-se-á usar o tipo de abordagem proposta em [11]. Para os exercícios de **desenvolvimento** não está previsto qualquer tipo de correção automática. No entanto, no caso do exemplo acima apresentado para exercícios de desenvolvimento, pode-se prever a sua correção (semi)-automática, uma vez que se poderá implementar o método referido e comparar a solução com a resolução do aluno.

4 A Linguagem GerExa

Com base na informação referida na secção anterior, construiu-se uma linguagem XML, denominada GerExa. A linguagem será descrita usando DTDs ("Document Type Declaration") [3], sendo indicado um fragmentos dos DTDs para cada

uma das suas componentes. A escolha deste esquema de especificação, em detrimento de, por exemplo, o XML Schema [9] ou RelaxNG [7], prende-se com sua maior simplicidade e, também, ser o mais conhecido e implementado. A possibilidade de especificar tipos de estruturas de dados e o modo como se podem compor as diversas especificações, serão vantagens dos outros dois esquemas referidos. E, embora as expressividades associadas sejam diferentes [12], pretendemos fazer uma especificação da linguagem GerExa usando o RelaxNG. A sua simplicidade (especialmente na notação compacta) e clara formalização teórica, tornam-o um forte candidato de substituição dos DTDs. Uma das vantagens que se pretende explorar (para além das acima referidas) é a maior flexibilidade entre o uso de elementos e atributos.

4.1 Exercícios

O fragmento de DTD seguinte especifica o elemento `exercicio`:

```
<!ELEMENT exercicio ((enunciado)?,(questao)+)>
<!ATTLIST exercicio id_conteudo IDREF #REQUIRED
                    id_exercicio ID #REQUIRED >
<!ELEMENT enunciado (#PCDATA|formula|imagem|programa)*>
<!ELEMENT questao (pergunta,(escolha_multipla|resposta_multipla|
correspondecia|verdadeiro_falso|desenvolvimento|
resposta_calculada),valorizacao,(cotacao_questao)?)>
<!ATTLIST questao id ID #REQUIRED
                  grau_dificuldade (1|2|3|4) "2">
<!ELEMENT pergunta (#PCDATA|formula|imagem|programa)*>
<!ELEMENT cotacao_questao (#PCDATA)>
<!ELEMENT valorizacao (cotacao,(classificacao)?)>
<!ELEMENT cotacao (valor,valordesconto)>
<!ELEMENT classificacao (#PCDATA)>
<!ELEMENT valor (#PCDATA)>
<!ELEMENT valordesconto (#PCDATA)>
```

Cada elemento `exercicio` tem um elemento `enunciado`, que é opcional, e vários elementos `questao`. O enunciado pode conter texto, fórmulas matemáticas, fragmentos de programas ou imagens. As questões correspondem aos exercícios com alíneas. O enunciado não é obrigatório uma vez que cada questão tem um elemento `pergunta`, que pode dispensar o uso de enunciado. Por exemplo:

```
<enunciado> Considere o lançamento de duas moedas.</enunciado>
<questao id="1">
<pergunta> Qual a probabilidade de sair cara?</pergunta>
</questao>
```

Para este mesmo enunciado poderia haver mais questões, por exemplo:

```
<questao id="2">
   <pergunta> Qual a probabilidade de sair coroa? </pergunta>
</questao>
```

Mas pode-se optar por colocar tudo, dentro do elemento `pergunta`, prescindindo do `enunciado`:

```
<pergunta>No lançamento de duas moedas qual a probabilidade
 de sair coroa?
</pergunta>
```

As fórmulas matemáticas serão guardadas usando a linguagem MathML [8], mas aqui iremos apenas indicar que correspondem a um elemento `formula`. Para as imagens o conteúdo do elemento `imagem` será uma indicação da localização da imagem. Os programas poderão ser acedidos como as imagens ou incluídos no elemento, mas em qualquer dos casos deverá ser indicada a linguagem de programação, p.e:

```
<!ELEMENT programa (#PCDATA)>
<!ATTLIST programa codigo (CDATA) #IMPLIED
                   linguagem (CDATA) #REQUIRED
                   uri (0|1) ''0'' #REQUIRED>
```

Cada exercício foi hierarquizado, na estrutura referida anteriormente, ou seja o exercício pertence a um determinado conteúdo, que está incluído num módulo de uma determinada disciplina. Considera-se como atributo o identificador do conteúdo ao qual pertencem. Esta hierarquia poderá ser demasiado rígida. A nível de especificação XML seria simples alterar: bastava considerar que o exercício teria um elemento `disciplina`, que teria opcionalmente um elemento `modulo`, e este por sua vez um elemento `conteudo`. Cada um destes elementos teria apenas um atributo com o respectivo identificador.

Cada questão pode ser dum dos tipos seguintes, que por abuso de linguagem, iremos designar por *tipo de exercício*:

Escolha múltipla

Neste tipo de exercício existem várias opções das quais apenas uma está correcta. Estipulou-se a inserção mínima de três opções e máxima de cinco, sendo este número configurável na aplicação. Como existe uma única opção correcta, a resposta correcta tem o valor do atributo `resposta` igual a `1` e para as restantes esse valor será `0`.

A respectiva estrutura é a seguinte:

```
<!ELEMENT escolha_multipla (op,op,op,(op)?,(op)?)>
<!ELEMENT op (#PCDATA|formula|imagem|programa)*>
<!ATTLIST op
        op_id  (1|2|3|4|5) "1"
        resposta  (0|1) "0"
        resolucao  (0|1) #IMPLIED>
```

O atributo `resolucao` será usado para guardar a resolução do aluno.

Resposta múltipla

Neste tipo de exercícios, decidiu-se que poderiam ser inseridas no mínimo duas alíneas e no máximo oito, e tal como nos exercícios de escolha múltipla, este valor será configurável na aplicação. Cada uma das alíneas pode ser verdadeira ou falsa, a que corresponderá o valor do atributo `resposta`, `1` ou `0`, respectivamente.

```
<!ELEMENT resposta_multipla (alinea,alinea,alinea?,alinea?
alinea?,alinea?,alinea?,alinea?)>
<!ELEMENT alinea (#PCDATA|formula|imagem|programa)*>
<!ATTLIST alinea
        alinea_id  (1|2|3|4|5|6|7|8) "1"
        resposta  (0|1) "0"
        resolucao  (0|1) #IMPLIED>
```

Como no caso anterior, o atributo `resolucao` será usado para guardar a resolução do aluno. Note-se que a nível de especificação XML os elementos `escolha_multipla` e `resposta_multipla` são basicamente idênticos. E isto aconteceria se tivesse sido usado outro esquema, p.e, XML Schema ou RelaxNG. No entanto a sua separação permitirá um tratamento diferente pela aplicação de verificação e correção automática.
Nestes dois casos, o número de opções (ou de alíneas) podia também ter ficado por especificar usando-se, por exemplo, o operador `+`. Esta especificação, contudo, está de acordo com o interface usado para a geração dos exercícios. Caso seja necessário, poder-se-á fazer uma especificação mais genérica, que valide os documentos obtidos pela especificação aqui adoptada e outros mais gerais.

Correspondência

Nestes exercícios é necessário fazer a correspondência de linhas entre duas colunas com igual número de linhas. Considera-se um mínimo três linhas e um máximo cinco. Cada linha tem um atributo identificador. A coluna 2 tem uma numeração fixa. Para a coluna 1 o atributo `resposta`, indica, para cada linha, qual a linha correspondente na coluna 2. Também o atributo `resolucao` apenas é necessário na coluna 1.

```
<!ELEMENT correspondecia (col1,col1,col1,(col1)?,(col1)?,
        col2,col2,col2,(col2)?,(col2)?,(resolucao)?)>
<!ELEMENT col1 (#PCDATA|formula|imagem|programa)*>
<!ATTLIST col1
      col1_id  (1|2|3|4|5) "1"
      resposta  (1|2|3|4|5) "1"
      resolucao  (1|2|3|4|5) #IMPLIED>
<!ELEMENT col2 (#PCDATA|formula|imagem|programa)*>
<!ATTLIST col2 col2_id  (1|2|3|4|5) "1">
```

Mais uma vez, não é possível garantir, nesta especificação, que o número de elementos `col1` e `col2` é o mesmo. Para tal teria de se ter diversos elementos: `correspondencia3`, `correspondencia4`, etc.

Verdadeiro-Falso

Para este tipo de exercício apenas é necessário indicar se a pergunta é verdadeira ou falsa.

```
<!ELEMENT verdadeiro_falso (EMPTY)>
<!ATTLIST verdadeiro_falso
      resposta  (V|F) "F"
      resolução  (V|F) #IMPLIED>
```

Resposta Calculada

Nesta categoria englobam-se todos os exercícios cuja resposta pode ser obtida através de um cálculo efectuado por computador. No caso mais simples, apenas poderá ser definido no elemento `resultado`, o valor da resposta. Mas, no caso geral, deverá ser dado no elemento `dominio` o método e os dados necessários.

```
<!ELEMENT resposta_calculada (respostacal,resolucaocal?)>
<!ELEMENT respostacal (resultado|dominio)>
<!ELEMENT resolucaocal (#PCDATA|formula|programa)*>
<!ELEMENT resolucao_calculada ANY>
<!ELEMENT resultado  (#PCDATA|formula|programa)*>
<!ELEMENT dominio (aplicacao,metodo,testes)>
<!ELEMENT aplicacao (#PCDATA)>
<!ELEMENT metodo (#PCDATA)>
<!ELEMENT testes ANY>
```

Desenvolvimento

Como já se disse, para estes exercícios não se prevê nenhum tipo de correção automática. Contudo pode ser vantajoso que o professor indique qual seria a resposta correcta, para uma eventual indicação aos alunos, no caso de testes de treino, ou para uma possível correção depois de um teste de avaliação.

```
<!ELEMENT desenvolvimento (resposta_desen?,resolucao_desen?)>
<!ELEMENT resposta_desen (#PCDATA|formula|imagem|programa)*>
<!ELEMENT resolucao_desen (#PCDATA|formula|imagem|programa)*>
```

Para cada elemento `questao` consideram-se ainda elementos que permitam, num teste, o cálculo de uma classificação. Estes elementos referem-se à cotação que o professor pode atribuir a cada questão e a classificação obtida pelo aluno.

```
<!ELEMENT valorizacao (cotacao,classificacao?)>
<!ELEMENT cotacao (valor,valordesconto)>
<!ELEMENT classificacao (#PCDATA)>
<!ELEMENT valor (#PCDATA)>
<!ELEMENT valordesconto (#PCDATA)>
```

O elemento `cotacao` é constituído pelos elementos `valor` e `valordesconto`. O primeiro refere-se à percentagem de cada opção, no caso dos exercícios de escolha múltipla e resposta múltipla, nos restantes essa percentagem será 100%, uma vez que a pergunta é única; o segundo refere-se ao valor a descontar por cada resposta mal assinalada.

4.2 Disciplinas

Uma disciplina é constituída por módulos e cada módulo por conteúdos. A definição de uma disciplina pode ainda incluir alguma informação geral como o título ou designação, o curso ou cursos a que pertence, ano ou anos curriculares da disciplina, a escolaridade, etc. Temos os seguintes elementos:

```
<!ELEMENT disciplina (titulo,(curso)+,(anolectivo)?,
(escolaridade)?,(docente)*,(modulo)+)>
<!ATTLIST disciplina id_disciplina ID #REQUIRED
                     ref_docente IDREFS #IMPLIED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT curso (nome,ano)>
<!ATTLIST curso instituicao_id IDREF #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT ano (#PCDATA)>
<!ELEMENT anolectivo (#PCDATA)>
<!ELEMENT escolaridade (#PCDATA)>
<!ELEMENT modulo (conteudo)+>
<!ATTLIST modulo id_modulo ID #REQUIRED
                 tema CDATA #REQUIRED >
<!ELEMENT conteudo (#PCDATA)>
<!ATTLIST conteudo id_conteudo ID #REQUIRED>
```

Cada disciplina tem associado um ou mais docentes e uma referência à instituição, aos quais se associam os elementos seguintes:

```
<!ELEMENT docente (nome_doc)>
<!ATTLIST docente id_docente ID #REQUIRED>
<!ELEMENT nome_doc (#PCDATA)>
<!ELEMENT instituicao (#PCDATA)>
<!ATTLIST instituicao id_instituicao ID #REQUIRED>
```

Finalmente, é necessária alguma informação para identificar os alunos:

```
<!ELEMENT aluno (nome_aluno,numero_aluno)>
<!ATTLIST aluno id_aluno ID #REQUIRED>
<!ELEMENT nome_aluno (#PCDATA)>
<!ELEMENT numero_aluno (#PCDATA)>
```

A informação das disciplinas será agregada num elemento de topo da forma:

```
<!ELEMENT disciplinas (disciplina)+>
```

E, analogamente, para a informação dos docentes, dos alunos e das instituições.

4.3 Testes

Um teste é um conjunto de exercícios que poderá ter ou não uma identificação da data em que se realiza. O elemento `cotacao_teste` será usado para guardar a cotação da resolução de um aluno.

```
<!ELEMENT teste (data?,(exercicio)+,cotacao_teste?)>
<!ATTLIST teste id_teste ID #REQUIRED
                id_disciplina IDREFS #REQUIRED
                id_instituicao IDREF #IMPLIED
                aluno IDREF #IMPLIED>
                tipo_teste  (Treino|Ficha_Trabalho|Frequencia|
                     ExameN|ExameR|ExameEsp) "Frequencia">
<!ELEMENT data (#PCDATA)>
<!ELEMENT cotacao_teste (#PCDATA)>
```

5 A Aplicação GerExa

A especificação apresentada na secção anterior está a servir de base para o desenvolvimento da aplicação GerExa. A aplicação permitirá ao professor:

- Inserir, editar e apagar exercícios e disciplinas,
- Gerar testes de forma aleatoria ou assistida,
- Corrigir ou verificar testes,

e, permitirá ao aluno gerar e resolver testes de treino e resolver os testes de avaliação.

Para a inserção de um exercício será necessário seleccionar a disciplina, o respectivo módulo e o conteúdo a que o exercício pertence (que caso não existam, terão de ser introduzidos). Opcionalmente será introduzido o enunciado do exercício, e em seguida as várias questões. Para cada uma, será escolhido o tipo e mediante esse tipo, serão apresentados formulários para o preenchimento da informação correspondente.

Cada exercício é guardado num documento de tipo `exercicio`, i.e, num documento cujo elemento de topo é `exercicio`. A informação das disciplinas é guardada num documento de tipo `disciplinas`.

Os testes serão gerados pela aplicação, apartir dos exercícios anteriormente introduzidos (ver Figura 1).

Para o caso do aluno, a selecção dos exercícios é efectuada de forma completamente aleatória, dentro de um conteúdo, de um módulo de uma disciplina, ou vários conteúdos ou módulos de uma mesma disciplina.

O professor pode também optar pela geração automática ou por uma geração assistida. No primeiro caso, escolhe um conjunto de exercícios, a partir do qual pretende seleccionar um número menor. Depois de ter um conjunto de questões seleccionadas, pode alterar por exemplo, a ordem, retirar, acrescentar questões, etc.

A implementação está a ser considerada de modo a que, a geração dos testes seja o mais dinâmica possível, considerando-se futuramente a hipótese de alguns exercícios poderem ser gerados por aplicações externas à aplicação GerExa.

Cada teste é guardado num documento de tipo `teste`, sendo feita uma cópia dos vários exercícios que o compõem. Embora, possa aqui haver alguma redundância de informação, diminuí a complexidade da gestão, nomeadamente, no caso de se pretender apagar exercícios. Notar que a ênfase da aplicação é na organização dos exercícios, sendo os testes considerados documentos autónomos.

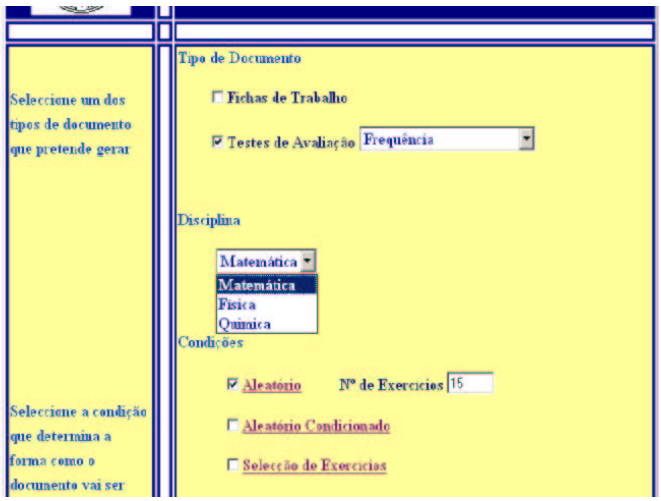


Figura 1. Interface para a geração de testes

Depois de gerado um teste, pode ser respondido por um ou mais alunos, e/ou transformado para ficheiros HTML, LaTeX, PDF, etc. Em particular, tanto o enunciado como as resoluções dos alunos poderão ser impressas em papel.

Os testes já resolvidos pelos alunos poderão ser consultados pelos professores para correção de perguntas de desenvolvimento (ou resposta calculada).

Cada teste resolvido por um aluno será guardado num documento XML. Durante as fases de resolução e correção esse documento conterá toda a informação do teste, podendo depois ser apenas armazenada a informação referente à resolução do aluno. Presentemente, não está definida qual a especificação destes últimos.

Para a introdução de fórmulas matemáticas, quer na inserção de exercícios quer na resolução dos alunos, está a ser usado actualmente uma aplicação javascript `ASCIIMathML` [1], a qual através de um mecanismo de importação permite converter para MathML. Na aplicação ao seleccionar o botão `Inserir Fórmulas Matemáticas`, aparece uma janela de texto que permite inserir as fórmulas, através de uma linguagem simbólica.

6 Conclusões

Uma aplicação como o GerExa, pretende tornar mais cómodo e eficiente o trabalho de produção de material didáctico e de avaliação. O recurso a uma linguagem como o XML, trás inovação e flexibilidade relativamente às bases de dados estruturadas tradicionais, mantendo a garantia de continuar a ter uma forma estruturada, mas mais próxima dos documentos usuais. A especificação apresentada pode contudo ser melhorada permitindo uma maior flexibilidade na construção dos exercícios, de que salientamos:

- mecanismos de geração de exercícios semelhantes (p.e, que difiram no valor de uma quantidade)
- os mecanismos de correção automática de exercícios de resposta calculada

Pretendemos ainda que o prótipo, agora em fase de desenvolvimento, seja testado e avaliado por um conjunto de professores e alunos.

Agradecimentos

Agradecemos aos revisores anónimos os comentários e sugestões que permitiram melhorar este trabalho.

Referências

1. Asciimathdemo. http://www1.chapman.edu/ jipsen/mathml/asciimathdemo.xml.
2. Edulq.com. http://www.eduiq.com/elearning.htm.
3. Especificação de xml. http://www.w3.org/TR/REC-xml.
4. Ims. http://www.imsglobal.org/question/qti_item_v2p0pd/xmlbinding.html.
5. Netsupport incorporated. http://www.netsupport-inc.com/nss/netsupport_school_overview.htm.
6. Pt inovação. http://www.ptinovacao.pt/produtos/plataformas/-fol_FORMARE.pdf.
7. Relax ng. http://www.relaxng.org/.
8. W3c math. http://www.w3.org/Math/implementations.html.
9. Xml schema. http://www.w3.org/XML/Schema.
10. José Carlos Ramalho e Pedro Henriques. *XML & XSL*. FCA, 2002.
11. José Paulo Leal and Nelma Moreira. Using matching for automatic assessment in computer science learning environments. In Francisco Restivo and Lígia Ribeiro, editors, *Web-Based Lerning Environments*. Merlin 2000 Project, FEUP, June 2000.
12. Dongwon Lee, Murali Mani, and Makoto Murata. Reasoning about xml schema languages using formal language theory. Technical report, IBM Almaden Research Center, 2000.
13. Jorge Reis Lima. *e-Learning e e-Conteúdos*. FCA, 2003.
14. Artur Afonso Sousa. *Bases de Dados, Web e XML*. FCA, 2000.

Integrando Web Services e Recursos Educacionais Através de Composição

Roseli Persson Hansen[1], Sérgio Crespo Coelho da Silva Pinto[2], Cristiano Roberto Hansen[3]

[1] Faculdade SEAMA, Coordenação de Trabalhos de Conclusão, Curso de Sistemas de Informação, Macapá – AP, Brasil
rhansen@seama.edu.br
[2] UNISINOS, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo – RS, Brasil
crespo@exatas.unisinos.br
[3] Faculdade SEAMA, Coordenação de Curso, Tecnologia em Redes de Computadores, Macapá – AP, Brasil
chansen@seama.edu.br

Resumo. Com o advento das novas tecnologias para desenvolvimento de ambientes na *Web*, diversas ferramentas, linguagens e protocolos foram surgindo com o objetivo de permitir a construção de aplicações de forma mais rápida e eficiente. O mercado de *e-commerce* iniciou suas tentativas se utilizando de protocolos inseridos dentro do http para permitir que aplicações pudessem enviar seus dados por meio de SOAP. Paralelamente, no contexto educacional, novas formas de garantir a interoperabilidade entre aplicações foram sendo construídas, tais como IMS, ARIADNE dentre outras. Nestas iniciativas, XML torna-se a linguagem comum para garantir a semântica dos dados e assim permitir seu reuso. Este trabalho apresenta uma arquitetura que permite utilizar as facilidades já existentes em *e-commerce*, não existentes no contexto educacional, para reuso de objetos de aprendizagem LOM por meio de uma linguagem de composição chamada GlueScript.

1 Introdução

Os *Web Services* são aplicações modulares que podem ser descritas, publicadas e invocadas sobre uma rede, geralmente a *Web*. Ou seja, é uma interface que descreve uma coleção de operações que são acessíveis pela rede através de mensagens em formato *eXtensible Markup Language* (XML) padronizadas. Permitem uma integração de serviços de maneira rápida e eficiente [7],[10].

Através da estrutura arquitetural dos *Web Services* é permitida a comunicação entre diferentes aplicações. Um serviço pode ser invocado remotamente ou ser utilizado para compor um novo serviço juntamente com outros serviços. Como a tecnologia de *Web Services* está baseada em XML, é possível invocar ou reutilizar um serviço sem a necessidade de conhecer a plataforma ou linguagem de programação usada na sua construção.

Alguns projetos como o IMS estão direcionando seus esforços na tentativa de obter a interoperabilidade dos recursos entre diferentes sistemas educacionais. Esta interoperabilidade é conseguida através da utilização de padrões como, por exemplo, o metadado *Learning Object Metadata* (LOM) [8].

As arquiteturas de *e-learning* existentes têm uma definição bastante precisa quanto à descrição de recursos educacionais. Já a arquitetura de *Web Services* apresenta uma estrutura bem definida para permitir busca, acesso e principalmente reuso de serviços na *Web*.

Com o foco na integração, foi desenvolvida uma arquitetura, que utiliza a linguagem de composição chamada GlueScript, para permitir que recursos disponibilizados por sistemas educacionais possam ser reutilizados juntamente com serviços acessados através da arquitetura de *Web Services*.

2 Web Services Como Forma de Garantir a Interoperabilidade Entre Aplicações

Um *Web Service* é um componente de *software* independente de implementação e plataforma. Pode ser descrito utilizando-se uma linguagem de descrição de serviços, publicado em um registro e descoberto através de um mecanismo de busca padrão. Por conseguinte, também ser invocado a partir de uma *Application Program Interface* (API) através da rede, e ser composto com outros serviços [7],[10].

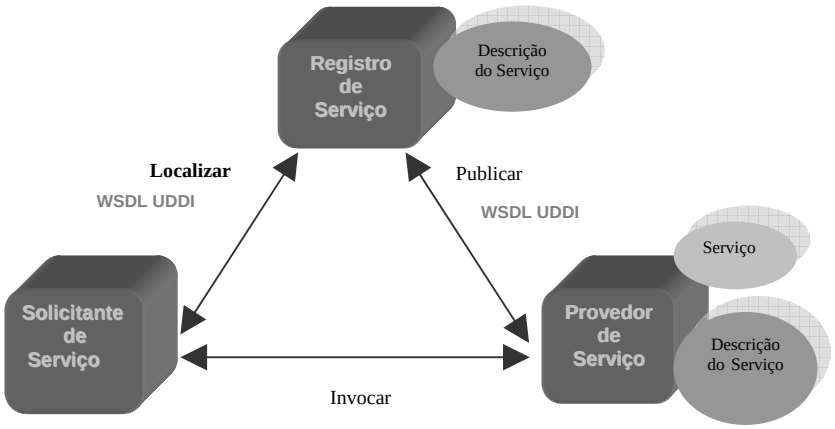


Fig. 1. Arquitetura Geral de *Web Services* [10].

A arquitetura de *Web Services* apresentada na Fig. 1, está baseada nas interações entre um Provedor de Serviço para disponibilizar um serviço, um Solicitante de Serviço que faz uso do serviço e um Registro de Serviço onde os provedores publicam as descrições dos serviços. A arquitetura de *Web Services* é composta por padrões XML assim como: a *Web Services Description Language* (WSDL) para

descrever os serviços; o *Simple Object Access Protocol* (SOAP) utilizado para publicar, localizar e invocar um *Web Service* em um registro UDDI; o *Universal Description, Discovery and Integration* (UDDI), um registro que é acessado por clientes para localizar os serviços de que necessitem.

A utilização de padrões como SOAP, UDDI e WSDL representam uma forma de permitir a comunicação de *Web Services* construídos em diferentes plataformas. O uso de padrões baseados em XML, permite que aplicações possam se comunicar utilizando tecnologia de mensagem padrão. Desta forma, a linguagem de programação na qual a aplicação foi construída torna-se transparente.

Outra característica da arquitetura de *Web Services*, é a possibilidade de construir novas aplicações utilizando-se a composição de serviços que estejam publicados em *Web Services*.

Por outro lado, os metadados educacionais possibilitam que se possa fazer descrição de recursos educacionais, de forma a permitir o compartilhamento dos mesmos. Portanto torna-se relevante conceituar brevemente os metadados educacionais na próxima seção.

3 Metadados de Objetos Educacionais - LOM

Os metadados são "dados sobre dados" ou "informações sobre informações". São um conjunto de palavras, frases ou sentenças que resumem ou descrevem o conteúdo de um site *Web* ou uma página *Web* individual, um recurso computacional com o objetivo de beneficiar o trabalho de agentes de busca [3], [4].

Os metadados são descritos em formato XML, formando uma estrutura de dados que descreve as características de um recurso. Desta maneira, podem ser utilizados para descrever conteúdos e estruturas, além de fornecer informação sobre acessibilidade, organização e relacionamento entre os dados [3].

Alguns padrões de metadados para *e-learning* estão emergindo, dentre eles pode-se citar o Dublin Core e o IEEE-LOM [4],[8].

No padrão LOM um objeto de aprendizagem é definido como uma entidade que pode ter um formato digital ou não. Este padrão pode ser utilizado em sistemas de aprendizado, educação e treinamento. Uma instância de um metadado descreve características relevantes do objeto de aprendizagem enfocando a sua semântica, uso e pressupostos técnicos e pedagógicos de como deve ser melhor utilizado [3],[8]. O LOM é caracterizado por permitir a definição de blocos independentes, compartilhamento e troca de conteúdo educacional. Estes blocos podem conter referências para outros objetos e podem ser combinados ou seqüenciados para formar grandes unidades educacionais [8].

A seguir, será apresentada uma proposta de arquitetura que visa promover a interoperabilidade por meio da integração de *Web Services* e LOM-XML.

4 Usando Objetos LOM Através de Web Services

A arquitetura de integração é composta pela arquitetura de *Web Services* apresentada na Seção 2 e por uma arquitetura genérica extraída a partir de estudos realizados em algumas arquiteturas de metadados educacionais, dentre elas ARIADNE e GESTALT [2],[5].

Esta arquitetura genérica é composta por um Cliente o qual busca por um recurso educacional através de um *Broker*. Este consulta um repositório de metadados educacionais em formato LOM-XML.O *Broker* referencia o Provedor de Conteúdo que é responsável por acessar o repositório de recursos educacionais e fazer a entrega do recurso solicitado ao cliente. Isto pode ser visto na Fig. 2.

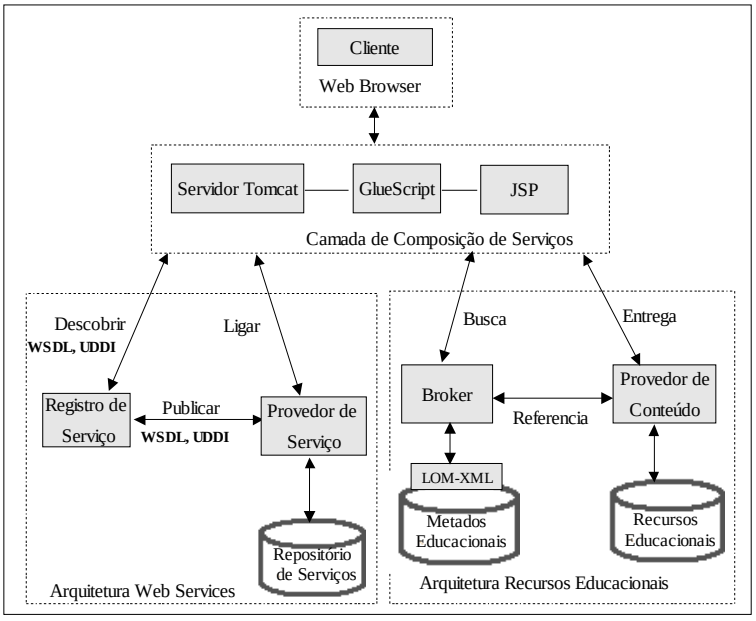


Fig. 2. Protótipo da Arquitetura da GlueScript [6].

A camada de composição é composta por um servidor *Web* Tomcat, pela linguagem JSP e pela GlueScript.

As características de algumas linguagens *scripts* foram observadas com o objetivo de encontrar uma para que a GlueScript pudesse estendê-la. Foi escolhida a linguagem *Java Server Pages* (JSP). Foi criada pela Sun, é multiplataforma e roda em servidores como Tomcat, BEA WebLogic, IBM WebSphere e outros que tenham *Java Virtual Machine* (JVM). Os códigos JSP são transformados em *servlets* em *background* [10].

‘A linguagem JSP é baseada em Java o que a torna portável. É utilizada através de *scriptlets* e *tags*. Os *scriptlets* são escritos em Java e exige do programador conhecimento da sintaxe Java. As *tags* apresentam uma sintaxe simples e permitem fácil integração no ambiente de desenvolvimento [10],[11].

É possível implementar um conjunto de *tags* que são chamadas de *Tag Libraries*. Estas JSP *Tag Libraries* permitem estender JSP [11].

A linguagem de composição GlueScript é uma extensão de JSP e será utilizada através de uma *Tag Library*. Desta forma, os desenvolvedores de ambientes Web poderão utilizar a GlueScript sem necessitar conhecer programação Java.

5 GlueScript Uma Linguagem de Composição de Web Services e Objetos LOM

A Fig. 3 apresenta um diagrama UML do relacionamento da *Tag Library* da GlueScript e JSP. O desenvolvedor *Web* poderá fazer uso dos recursos já disponíveis em JSP e fazer uso das *tags* da GlueScript.

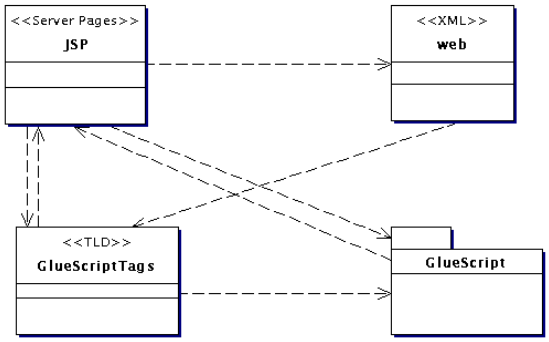


Fig. 3 Diagrama UML da GlueScript [6]

Quando o servidor for compilar os códigos JSP, e encontrar *tags* que não forem padrão de JSP, através de um arquivo web.xml irá localizar o arquivo que armazena as *tags* da GlueScript. Um arquivo GlueScriptTags irá referenciar as *tags* e os parâmetros possíveis de serem utilizados.

Foram desenvolvidas um total de oito *tags*. A Fig. 4 apresenta o relacionamento entre os arquivos Web.xml (A) e GlueScriptTags.tld (B) onde:

1. Define o endereço da localização de todos os arquivos relacionados para que possam ser encontrados pelo Tomcat.
2. Define a localização do arquivo descritor das *tags* (GlueScriptTags.tld). Desta forma o servidor pode localizar o descritor de *tags*, o qual apresenta a relação de todas as *tags* desenvolvidas, os parâmetros (quando houver) e o nome da classe Java relacionada.
3. Apresenta a definição de uma *tag* (GlueScript) e sua classe Java correspondente no pacote GlueScript. Esta é a *tag* principal do conjunto de *tags*. Em virtude do

arquivo descritor de *tags* ser de um tamanho considerável, ele apresenta apenas a *tag* principal (GlueScript).

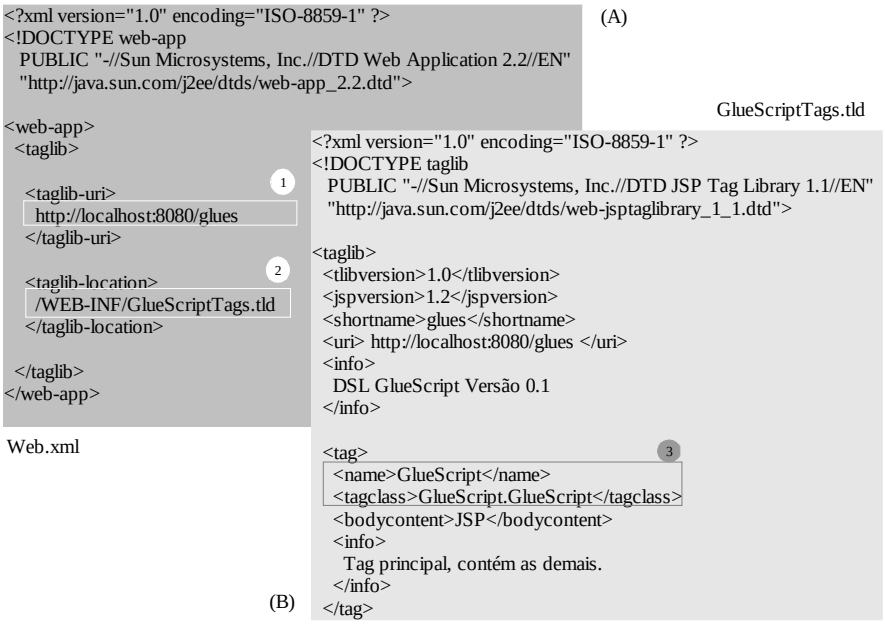


Fig. 4 Relacionamento entre os arquivos Web.xml e GlueScriptTags.tld

5.1 Tag GlueScript

A *tag* GlueScript é necessária para a utilização de todas as outras *tags* desenvolvidas. Esta *tag* é responsável por armazenar os parâmetros que são retornados pelas demais *tags* da linguagem. É a *tag* que determina o início e o final da utilização das *tags* da GlueScript. As demais *tags* são descritas nas subseções que seguem.

5.2 Tag LocateWS

A *tag* LocateWS faz a localização de *Web Services*. Os registros UDDI são acessados e então é possível saber a localização de um documento WSDL que descreve o *Web Service* definindo a sua localização, seus métodos e a forma de acesso. Na Fig. 5 o arquivo descritor de *tags* GlueScriptTags (A), define:

1. O nome da *tag* e a classe Java correspondente no pacote GlueScript.
2. O parâmetro que deve ser passado para a *tag*.
3. O atributo *address* indica o endereço de um registro UDDI.
4. Define se o atributo é obrigatório ou não.
5. Permite a inserção de scriptlets JSP, ou seja, o parâmetro *address* pode ser passado para a *tag* através de um *scriptlet* JSP.

5.3 Tag ParsingWS

A *tag* ParsingWS permite analisar os documentos de descrição de *Web Services* para obtenção de informações como a localização, o tipo de recurso ou serviço dentre outras. A Fig. 5 parte (A) descreve:

6. Define o nome da *tag* a ser utilizada dentro de uma página JSP.
7. Define a classe Java acessada pela *tag* no pacote da GlueScript.

GlueScriptTags.tld (A)

```
<tag>
  <name>LocateLOM</name> 1
  <tagclass>GlueScript.LocateLOM</tagclass> 2
  <bodycontent>JSP</bodycontent>
  <info>
   Localiza metadados educacionais no registro passado como parâmetro.
  </info>
  <attribute>
   <name>address</name> 3
   <required>true</required> 4
   <rtexprvalue>true</rtexprvalue> 5
  </attribute>
</tag>
<tag>
  <name>ParsingWS</name> 6
  <tagclass>GlueScript.ParsingWS</tagclass> 7
  <bodycontent>JSP</bodycontent>
  <info>
   Efetua o parsing de um WSDL passado como parâmetro.
  </info>
</tag>
```

(B)

```
<%@ taglib uri="http://localhost:8080/glues" prefix="gs" %>
<html>
 <title>Teste da GlueScript</title>
 <body>
   <gs:GlueScript>
     <gs:ParsingWS>
       <gs:LocateWS address="<%=registry%>"/>
     </gs:ParsingWS>
   </gs:GlueScript>
 </body>
</html>
```

Fig. 5 *Tags* LocateWS e ParsingWS

A *tag* ParsingWS é utilizada em conjunto com a *tag* LocateWS. Um exemplo de uso é apresentado na Fig. 5 (B).

5.4 Tag AllocateWS

A *tag* AllocateWS permitem a utilização de *Web Services.* A Fig. 6 apresenta um exemplo de uso da *tag* AllocateWS. O arquivo de descrição de *tags* GlueScriptTags (A) define o *tag* e seus atributos:

1. Atributo *endpoint*: é o endereço para acesso do *Web Service*.
2. Atributo *service*: é o nome do *Web Service* que será invocado.
3. Atributo *portn*: é a porta para a qual será enviado o pedido do serviço.
4. Atributo *tns*: é o *namespace* do *Web Service*.
5. Atributo *method*: é o nome do método que será invocado.

6. Atributo *parameters*: são os parâmetros que serão passados para o método.
7. Atributo valores: são os dados dos parâmetros especificados.
8. Atributo *pipe*: indica se a alocação alimentará a *tag* Pipe.

GlueScriptTags.tld

```
<tag>
  <name>AllocateWS</name>
  <tagclass>GlueScript.AllocateWS</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>
   Utiliza um método de um WS.
  </info>
  <attribute>
   <name>endpoint</name>
   <required>true</required>          1
  </attribute>
  <attribute>
   <name>service</name>
   <required>true</required>          2
  </attribute>
  <attribute>
   <name>portn</name>
   <required>true</required>          3
  </attribute>
  <attribute>
   <name>tns</name>
   <required>true</required>          4
  </attribute>
  <attribute>
   <name>method</name>
   <required>true</required>          5
  </attribute>
  <attribute>
   <name>parameters</name>
   <required>true</required>          6
  </attribute>
  <attribute>
   <name>valores</name>
   <required>true</required>          7
  </attribute>
  <attribute>
   <name>pipe</name>
   <required>true</required>          8
  </attribute>
</tag>
```

(A)

```
<%@ taglib uri="http://localhost:8080/glues" prefix="gs" %>
<html>
 <title>Teste da GlueScript</title>
 <body>
  <gs:GlueScript>
   <gs:AllocateWS
    endpoint="http://localhost:8000/DayService/DayService"
    tns="urn:DayService/wsdl"
    portn="DayServiceServantInterfacePort"
    service="DayService"
    method="getDayofWeek"
    parameters="year month day"
    valores="2002 12 20"
    pipe="no"/>
  </gs:GlueScript>
 </body>
```

(B)

Fig. 6 *Tag* AllocateWS

Na parte (B) da Fig. 6 é exemplificado o uso das *tags* para alocação de um *Web Services*.

5.5 Tag LocateLOM

Através do endereço de um registro de metadados LOM é possível saber onde estão os documentos de descrição de recursos educacionais.

5.6 Tag ParsingLOM

A *tag* ParsingLOM permite analisar os documentos de descrição recursos educacionais para obtenção de informações como a localização, o tipo de recurso ou serviço dentre outras. Na Fig. 7 (A) é apresentado o arquivo descritor da *tag*. Esta é utilizada em conjunto com a *tag* LocateLOM, conforme o exemplo da Fig. 7 (B).

GlueScriptTags.tld

(A)

```
<tag>
  <name>ParsingLOM</name>
  <tagclass>GlueScript.ParsingLOM</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>
   Efetua o parsing de um documento LOM-XML passado como parâmetro.
  </info>
</tag>
```

(B)

```
<%@ taglib uri="http://localhost:8080/glues" prefix="gs" %>
<html>
 <title>Teste da GlueScript</title>
 <body>
  <gs:GlueScript>
   <gs:ParsingLOM>
    <gs:LocateLOM address="http://localhost:8080/lom/metadata.index"/>
   </gs:ParsingLOM>
  </gs:GlueScript>
 </body>
</html>
```

Fig. 7 *Tags* ParsingLOM e LocateLOM

5.7 Tag AllocateLOM

Recursos educacionais podem ser acessados através da *tag* AllocateLOM. A Fig. 8 apresenta o arquivo descritor da *tag* (A) com seus atributos:

1. A definição do nome da *tag* e a classe Java correspondente no pacote GlueScript.
2. Atributo *name*: especifica o nome de um recurso educacional a ser acessado. O campo *required* define se é obrigatório ou não.
3. Atributo lomUrl: especifica o endereço de um recurso educacional.
4. Atributo pipe: indica se a alocação alimentará a *tag* Pipe.

A Fig. 8 (B) também apresenta um exemplo de uso da *tag* AllocateLOM.

GlueScriptTags.tld

```
<tag>                                                          (A)
  <name>AllocateLOM</name>                                  1
  <tagclass>GlueScript.AllocateLOM</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>
    Verifica se um determinado metadado educacional está disponível.
  </info>
  <attribute>
    <name>name</name>                                       2
    <required>true</required>
  </attribute>
  <attribute>
    <name>lomUrl</name>                                     3
    <required>true</required>
  </attribute>
  <attribute>
    <name>pipe</name>                                       4
    <required>true</required>
  </attribute>
</tag>
```

(B)

```
<%@ taglib uri="http://localhost:8080/glues" prefix="gs" %>
<%@ page language="java" import="java.util.*" %>
<html>
 <head>
   <title>GlueScript - Teste LOM (AllocateLOM)</title>
 </head>

 <body>
   <gs:GlueScript>
     <gs:AllocateLOM
        name="The Java Web Services Tutorial"
        lomUrl="http://localhost:8080/jwsdp/docs/tutorial/index.html"
        pipe="no"/>
   </gs:GlueScript>
 </body>
</html>
```

Fig. 8 *Tag* AllocateLOM

5.8 Tag Pipe

A *tag* Pipe tem como objetivo permitir que o resultado da invocação de um *Web Service* ou um recurso educacional possa servir de entrada para outro. Assim como *Web Services*, os recursos educacionais podem fazer uso da *tag* Pipe. Logo, os dados podem ser manipulados e modificados, adequando-os ao serviço que irá recebê-los como parâmetro de entrada. Caso o desenvolvedor necessite modificar ou converter os tipos de dados de saída para servirem como entrada por outro serviço ou recurso, é possível fazer uso dos recursos disponíveis pelo JSP como, por exemplo, *scriptlets*. O desenvolvedor pode ainda implementar filtros para manipulação destes dados, caso seja de seu interesse, e utilizá-los juntamente com a *tag* Pipe.

Para um melhor entendimento da relação entre as *tags*, a Fig. 9 apresenta um diagrama de seqüência. Este diagrama demonstra que a *tag* LocateWS é utilizada para localizar os *Web Services* disponíveis em um registro UDDI. Esta *tag* é utilizada em conjunto com a *tag* ParsingWS que realiza o *parsing* do registro para retornar os nomes dos *Web Services* e a localização do documento WSDL destes. Com a localização do documento WSDL é possível saber a localização do *Web Service* e quais métodos são necessários para sua utilização, bem como os parâmetros que devem ser passados. Através da *tag* AllocateWS é possível então fazer uso do *Web Service*.

A *tag* LocateLOM é utilizada para localizar recursos educacionais através de um endereço de um repositório de LOM-XML. É utilizada em conjunto com a *tag* ParsingLOM para extrair as informações de localização e como um recurso educacional pode ser utilizado. Estas informações são retornadas e então através da *tag* AllocateLOM é possível fazer uso deste recurso.

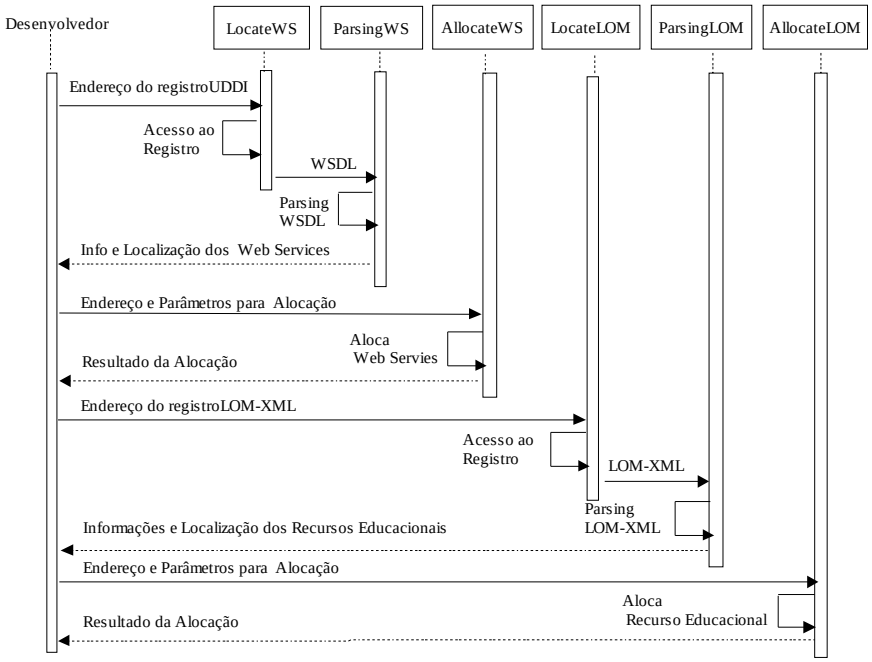


Fig. 9 Diagrama de Seqüência das *tags* da GlueScript

As *tags* foram implementadas em Java utilizando o pacote *Java Web Services Development Pack* (JWSDP) [1] , que apresenta um conjunto de APIs Java para XML o que facilita o uso do protocolo SOAP para o acesso aos registros UDDI, documentos WSDL e LOM-XML.

6 Conclusões e Trabalhos Futuros

A utilização da tecnologia baseada em *Web Services* permite que serviços possam ser disponibilizados na *Web* de forma padronizada. Esta padronização permite a localização e acesso a serviços de forma bastante eficiente, já que as tecnologias empregadas estão baseadas em XML

Este trabalho propôs uma arquitetura de integração das tecnologias de *Web Services* e recursos educacionais descritos através de LOM-XML. Esta integração se dá através da linguagem de composição GlueScript.

A linguagem GlueScript é uma *Tag Library* JSP, dando suporte à localização e acesso a *Web Services* e recursos educacionais. Através do uso das *tags* da GlueScript a construção de ambientes *Web* torna-se facilitada pois um desenvolvedor não necessita implementar as aplicações de que necessita. Ao utilizar a GlueScript um desenvolvedor pode também fazer uso da linguagem JSP para dar mais poder de processamento a linguagem. O uso de JSP em conjunto com a GlueScript permite que o desenvolvedor possa acrescentar funcionalidades à sua página. As *tags* da

GlueScript têm funcionalidades específicas para localização e acesso a *Web Services* e recursos educacionais. Caso o desenvolvedor deseje trabalhar com banco de dados, por exemplo, este deverá recorrer às *tags* específicas de JSP.

Cabe salientar que a linguagem GlueScript não tem suporte à manipulação de tipos de dados complexos. Para que isso seja possível é necessário aperfeiçoar o conjunto de *tags*. A *tag* Pipe também necessita ser aperfeiçoada de forma a permitir a transformação de dados (filtros) utilizados como entrada e saída para *Web Services* e recursos educacionais. Até o momento esta *tag* apenas direciona as saídas e entradas de dados para os serviços e recursos.

Referências Bibliográficas

1. Amrstrong, E.; Bodoff, S.; Carson, D.; Fisher, M.; Green, D. and Haase, K. The JavaWeb Services Tutorial (2002) http://java.sun.com/webservices/downloads/webservicespack.html Último acesso - Novembro 2004.
2. ARIADNE Educational Metadata Recommendation, Version 3.2 (2002). http://www.ariadne-eu.org/. Último acesso – Novembro 2004.
3. BECTa - British Educational Communications and Technology agency. Metadata in Education. (2001).
4. Dublin Core Metadata Element Set, Version 1.1: Reference Description. (2003). http://dublincore.org/documents/2003/02/04/dces/. Último acesso - Novembro 2004.
5. Foster, P.; Kraner, M.; Graziano, A.; Romano, S. P. GESTALT - Getting Educational Systems Talking Across Leading-edge Technologies. (2000).
6. Hansen, R. P. GlueScript: Uma linguagem Específica de Domínio para composição de Web Services. Dissertação de Mestrado - Programa Interdisciplinar de Pós-Graduação -PIPCA - Universidade do Vale do Rio dos Sinos (2003).
7. Hansen, R. P.; Santos, C.T.; Pinto, S. C. C. S.; Lanius, G. L and Massen, F. Web Services: An Architectural Overview. First International Seminar on Advanced Research in E-Business - EBR 2002. PUC-RIO (2002).
8. IEEE LTSC. Draft Standard for Learning Object Metadata. (2002) http://ltsc.ieee.org/. Último acesso. Novembro 2004.
9. IMS Global Learning Consortium, Inc. IMS Learning Resource MetaData Information Model. (2001). http://www.imsproject.org/. ltimo acesso – Novembro 2004.
10. Kreger, H. *Web Services Conceptual Architecture*. IBM Software Group, May 2001.
11. Schachor, G.; Chace, A. and Rydin M. JSP Tag Libraries. Manning Publications Co (2001) 623 pages.
12. Silva, W. L. S. JSP and Tag Libraries for Web Development. New Riders (2002) 442 pages.

Edukalibre - Ferramentas de Auxílio à Produção de Documentos para Educação Baseadas em XML.

Guilherme Dutra, Nuno Faria e Jaime E. Villate

Faculdade de Engenharia da Universidade do Porto
Departamento de Física

Resumo Apresentam-se neste artigo um conjunto de ferramentas que compõem a plataforma de auxílio à elaboração colaborativa de documentos para educação, Edukalibre. Este sistema permite não só facilitar a gestão do controlo de versões mas, sobretudo, facilitar o acesso e produção de informação pela disponibilização dum leque alargado de formatos de documentação.
A utilização do formato DocBook XML como base geradora de outros formatos facilita a utilização de ferramentas previamente desenvolvidas e disponibilisadas como software livre. A partilha de conhecimentos e o re-aproveitamento de recursos previamente desenvolvidos tornam-se assim o objectivo das ferramentas desenvolvidas e a metodologia para o desenvolvimento dessas mesmas ferramentas.

1 Introdução

O desenvolvimento das comunicações inter-pessoais, proporcionado pela generalização da utilização da Internet, tem sido o factor decisivo na génese de uma classe de projectos baseados na partilha de conhecimentos e talentos. São exemplo disso os inúmeros projectos de software livre que, a partir de colaborações mais ou menos altruístas de uma enorme comunidade de programadores e utilizadores, permitiu já produzir programas de computador capazes de desempenhar eficientemente as mais variadas tarefas.

Paralelamente, tem-se registado um interesse crescente nas possibilidades oferecidas por programas de apoio ao ensino à distância. Um exemplo deste tipo de programas, licenciado como software livre, é o Moodle (Projecto Moodle, 2004).

O projecto Edukalibre (Projecto Edukalibre, 2004), aqui apresentado, tem como objectivo o desenvolvimento de ferramentas de apoio à criação de documentos para educação, a partir da colaboração entre vários autores através da Internet. Estas ferramentas permitem a criação de interfaces de fácil utilização para que autores sem uma formação especializada em informática possam beneficiar das vantagens dos sistemas de controlo de versões e conversão entre formatos de documentos.

O formato XML desempenha um papel fulcral nas ferramentas desenvolvidas pelo projecto Edukalibre. Aos autores é dada a possibilidade de produzirem os

seus documentos em formato Latex, Wiki ou utilizando o editor de texto do OpenOffice. Estes documentos são, de seguida, convertidos em formato XML (DocBook) e a partir deste, nos restantes formatos disponíveis nesta ferramenta. Finalmente, todos os ficheiros gerados são arquivados, utilizando para tal, uma ferramenta de controlo de versões.

Neste artigo são descritas as ferramentas de conversão entre XML e os restantes formatos, já desenvolvidas, assim como os projectos e principais desafios que se colocam aos investigadores que participam no projecto Edukalibre. São igualmente descritas as interfaces do sistema, nomeadamente os módulos para o Moodle já desenvolvidos.

2 O projecto Edukalibre

2.1 Objectivos do projecto

Edukalibre é um projecto financiado pelo programa Socrates / Minerva (Projecto Edukalibre, 2004), no qual participam seis países europeus. O seu principal objectivo consiste em explorar novas formas de facilitar a produção de materiais educativos, baseadas nas práticas e procedimentos normalmente observados nas comunidades de desenvolvimento de software livre.

Alguns dos aspectos a serem explorados prendem-se com o desenvolvimento colaborativo, a interacção com os utilizadores e a aplicação de sistemas de controlo de versões de documentos, no contexto das comunidades educativas. Em resumo, pretende-se desenvolver ferramentas de auxílio à criação de materiais educativos. Nomeadamente, estão a ser desenvolvidas ferramentas que permitam aos utilizadores colaborar para a criação material educativo, através de funcionalidades tais como:

- Sistema de controlo de versões de documentos
- Conversão automática de documentos para diversos formatos
- Diferentes formas de editar documetos e registra-los num repositório directamente a partir da interface de um editor de texto, a partir de um formulário na internet, etc...

Todas as ferramentas são desenvolvidas a partir de programas licenciados como software livre e todo o código desenvolvido é disponibilizado sob uma licença de softwre livre. Os documentos produzidos serão distribuídos sob licenças de documentação livres.

2.2 Ferramentas em desenvolvimento

As ferramentas que estão a ser desenvolvidas no projecto Edukalibre podem ser divididas em três classes distintas, tal como está exemplificado na figura 1.

A conversão de ficheiros entre os vários formatos admitidos pelo sistema é baseada no formato DocBook XML. Um ficheiro submetido ao sistema, por exemplo, em formato OpenOffice XML é convertido para DocBook XML e a

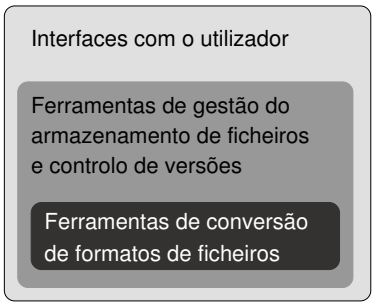


Figura 1. Esquema de classificação das ferramentas em desenvolvimento no projecto Edukalibre.

partir deste para um formato Wiki, HTML e PDF. No momento em que este artigo é escrito, não está ainda disponível a conversão entre o formato LaTex e DocBook, no entanto esse será um dos objectivos a explorar em trabalhos futuros. As ferramentas utilizadas para efectuar as conversões referidas são apresentadas na secção 3.

Um vez geradas as diferentes versões do documento, estas são arquivadas num repositório por um programa de controlo de versões. Esta ferramenta é apresentada na secção 3.1.

Paralelamente ao desenvolvimento do "motor" de gestão de ficheiros já apresentado, estão a ser desenvolvidas várias interfaces que simplificam a utilização do sistema através da Internet. Estas ferramentas são apresentadas na secção 4.

3 Sistema de gestão de documentos

As ferramentas de Gestão Electrónica de Documentos (GED) surgiram devido à necessidade de reduzir o volume papéis e permitir o acesso concorrente a documentos de diferentes unidades organizacionais. No seu sentido clássico as ferramentas GED devem permitir o armazenamento electrónico, a classificação e a recuperação de documentos (Vojciechovski, 2002). Funcionalidades como controlo de versão e workflow também são incorporadas nesse tipo de ferramenta.

O sistema Edukalibre enquadra-se na categoria de ferramentas GED, mas foram acrescentadas funcionalidades para servir propósitos educacionais.

O facto do sistema suportar diferentes formatos facilita a publicação de conteúdos e torna praticamente nula a necessidade de reescrever conteúdos já editados. Além disso os leitores podem ter acesso aos conteúdos no formato que mais lhes agrada, como pode ser observado na figura 2.

São autenticados dois grupos distintos de utilizadores. O primeiro grupo é entitulado Publicadores; cada membro desse grupo tem permissão de escrita para os seus próprios documentos. O segundo grupo, Contribuintes, é formado principalmente por leitores, mas englobam também publicadores potencias, os quais somente podem fazer modificações em documentos mediante autorização de um membro do grupo de Publicadores.

Ambos os grupos de utilizadores podem aceder a diferentes versões dos documentos via navegador Web. É possível também consultar as diferenças entre as diversas versões do documento.

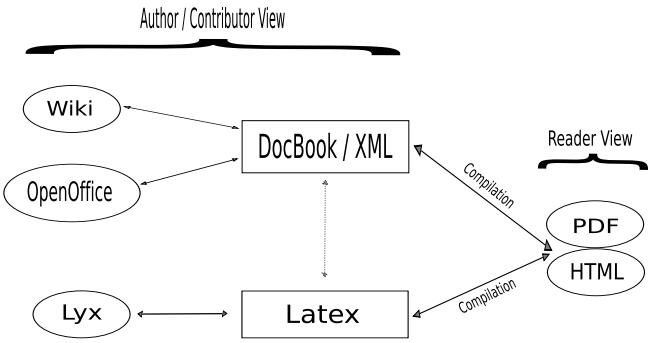


Figura 2. Conversões entre formatos e permissões de acesso.

3.1 Repositório de documentos e controlo de versões

Controlo de versões é a arte de gerir alterações na informação. Tem sido uma ferramenta crucial para programadores, os quais tipicamente gastam o seu tempo fazendo pequenas mudanças em programas e desfazendo-as no dia seguinte. Mas a utilidade do software de controlo de versões vai além dos limites do mundo de desenvolvimento de software (Collins-Sussman et al., 2004). No caso do sistema Edukalibre, o controlo de versões é utilizado na gestão de documentos.

O Sistema de Controlo de Versões (SVN) é parte fundamental da ferramenta Edukalibre pois gere a maneira como os documentos são armazenados, como novas ramificações são criadas e como essas ramificações são fundidas.

Cada documento tem uma ramificação principal que é criada e actualizada pelo autor. Quando um contribuinte actualiza o documento, uma nova ramificação é criada e o autor decidirá se a nova ramificação será incluída ou não na ramificação principal.

Na figura 3 podemos ver um exemplo de como as ramificações são tratadas. Algumas delas são incluídas na ramificação principal e outras não.

O acesso ao repositório SVN é possibilitado através de SSH (Secure Socket Shell) por linha de comando ou WebDav. Web-based Distributed Authoring and Versioning (WebDav) é um conjunto de extensões para HTTP independentes de plataforma que permitem aos utilizadores colaborativamente editar e gerir ficheiros em servidores Web remotos.

Recursos adicionais do SVN:

– Sistema de bloqueamento - impede autores de sobreescreverem alterações de outros autores;

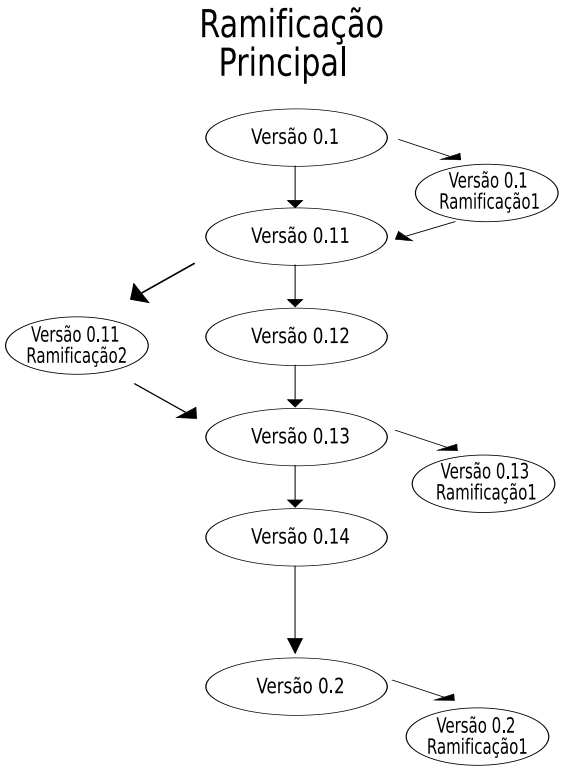


Figura 3. Esquema de como as ramificações são tratadas.

– Os documentos podem ser catalogados utilizando *NameSpaces*, assim pode-se criar colecções de documentos, mesmo que esses estejam em sítios distintos ou tenham sido movidos;
– É possível acrescentar Metadados, tal informação é armazenada no formato XML de forma transparente ao utilizador.

3.2 DocBook (XML) o formato base de todos os documentos

XML foi escolhido como formato base devido à facilidade de se escrever *scripts* de conversão para diversos outros formatos.

Decidiu-se pelo DocBook (Docbook.org, 2004) pois é uma DTD (Document Type Definition) amplamente utilizada e particularmente dirigida a livros e artigos. Ademais, utilizando-se OpenOffice é possível escrever documentos no formato DocBook tirando partido de uma interface WYSIWYG. É importante salientar que o formato nativo do OpenOffice também é XML, no entanto, é utilizado apenas pelo próprio OpenOffice.

Na conversão para os formatos PDF, PostScript e HTML utiliza-se o utilitário FOP (The Apache XML Project, 2004). O utilitário OOo2sDbk é utilizado na conversão para o formato OpenOffice. Ambos efectuam a conversão através de (Bellot, 2002) templates XSL (eXtensible Stylesheet Language).

3.3 Conversão entre Wiki e Docbook XML

O formato Wiki foi concebido para ser fácil de escrever e fácil de ler tanto quanto possível. A principal dificuldade na realização da tarefa de conversão está no facto de não haver um formato Wiki padrão. Ao invés disso, existem diversos formatos parecidos entre si mas com diferenças suficientes para tornar inconcebível uma ferramenta compatível com todos os formatos. Por isso, o grupo de desenvolvimento Edukalibre optou por focar apenas no formato Wiki Markdown (Gruber, 2004). Tal conversor foi implementado com a linguagem PHP.

Para fazer a conversão do formato Wiki Markdown para o formato DocBook foi utilizada a biblioteca Text_Wiki a qual pertence à framework PEAR (PHP Extension and Application Repository), (The PHP Group, 2004). A biblioteca Text_Wiki (Jones, 2004) é orientada a objectos e composta por pares de *Parsers* e *Renderers*. Os *parsers* analisam o texto e identificam a ocorrência de regras Markdown e os *renderers* transformam essas ocorrências nos seus equivalentes no formato de destino, nesse caso, o formato DocBook.

Utilizaram-se as funções XML Parser e SimpleXML da linguagem PHP para fazer a conversão do formato DocBook para o formato Wiki. Tendo em vista que o formato Wiki é mais simplificado do que o formato DocBook, muitas das formatações de texto são perdidas na conversão, no entanto, todo o corpo do texto é mantido.

3.4 Documentos em LaTex - um caso à parte

Devido à complexidade do formato LaTex, até o momento não há um utilitário que faça a conversão do formato Latex para o formato DocBook. Um utilitário

com esse propósito deverá ser implementado no futuro pelo grupo de desenvolvedores do projecto Edukalibre. A conversão do formato Wiki para o Latex é muito mais simples e já tem sido feito algum trabalho nessa área. De salientar que apesar de ter sido adoptado o DocBook XML como formato central, a aceitação do formato Latex é uma solução de compromisso para alguns casos em que são precisas as funcionalidades do Latex para criação de textos matemáticos complexos.

4 Interfaces de acesso em desenvolvimento

O acesso a um dado repositório de documentos pode ser feito a partir de várias interfaces distintas:

- Acesso directo via WebDAV;
- Interface *web* independente (PHP);
- *Macro* OpenOffice de acesso directo ao repositório;
- Módulo Moodle dedicado exclusivamente ao acesso ao repositório;
- Groupware - módulo integrado para o Moodle que fornece um conjunto de ferramentas para gestão de trabalhos em grupo, nomeadamente o acesso a repositórios Edukalibre.

As interfaces exclusivamente de acesso ao repositório permitem consultar, descarregar e publicar documentos ou actualizar documentos já existentes. A interface *web* independente permite ainda editar o ficheiro em linha, quer no seu formato original (excepto para ficheiros OpenOffice) quer em formato DocBook XML. Para o OpenOffice está em desenvolvimento uma *macro* que permitirá abrir ficheiros e publicar alterações directamente a partir do ambiente do programa, sem ser necessário recorrer a um programa externo para o fazer.

O módulo *Groupware*, está integrado na plataforma de ensino à distância Moodle, permitindo, por isso, ao utilizador o aceso a todas as outras ferramentas disponibilisadas por esta plataforma. Disponibilisa na mesma interface uma ferramenta de gestão de mensagens, um ambiente de gestão de discusões em tempo real e, claro está, o acesso ao repositório Edukalibre.

5 Conclusão

Apresentaram-se neste artigo um conjunto de ferramentas que compoêm o sistema Edukalibre. Estas ferramentas, apesar de estarem ainda em fase de desenvolvimento, permitem já proporcionar aos seus utilizadores um conjunto de facilidades que tornam mais simples o desenvolvimento partilhado de documentos para educação. Em desenvolvimentos futuros, serão privilegiados o desenvolvimento de ferramentas de conversão de Latex para DocBook e a implementação de um sistema de autenticação de utilizadores e gestão de permissões de acesso aos documentos, isto para além do aperfeiçoamento das ferramentas já desenvolvidas.

O formato DocBook XML ocupa um lugar de relevo em todas as ferramentas testadas. De facto, este é o formato para o qual todos os documentos são convertidos e a partir do qual todos os outros formatos são gerados. A principal vantagem da utilização deste formato reside na grande variedade de programas de conversão de e para este formato, licenciados como software livre. Assim, a filosofia que orientou o principal objectivo do projecto Edukalibre – facilitar a colaboração na elaboração de documentos para educação – acaba por guiar também as opções tomadas na escolha das tecnologias – utilisar, o mais possível, ferramentas e tecnologias previamente desenvolvidas e disponibilisar para a comunidade todas as ferramentas desenvolvidas no âmbito do projecto.

Bibliografia

Bellot, E. (2002), 'Ooo2sdbk', http://www.chez.com/ebellot/ooo2sdbk/.

Collins-Sussman, B., Fitzpatrick W., B. & Pilato, C. M. (2004), 'Version control with subversion', http://svnbook.red-bean.com/en/1.0/svn-book.pdf.

Docbook.org (2004), 'Docbook web site)', http://www.docbook.org/.

Gruber, J. (2004), 'Markdown', http://daringfireball.net/projects/markdown/.

Jones, P. (2004), 'Text_wiki', http://wiki.ciaweb.net/yawiki/index.php?area=Text_Wiki.

Projecto Edukalibre (2004), 'Página do projecto edukalibre na internet', http://www.edukalibre.org.

Projecto Moodle (2004), 'Página do projecto moodle na internet', http://moodle.org.

The Apache XML Project (2004), 'Fop (formating objects processor)', http://xml.apache.org/fop/.

The PHP Group (2004), 'Pear - php extension and application repository)', http://pear.php.net/.

Vojciechovski, M. (2002), 'O gerenciamento do conteúdo em projetos', http://www.pm21.com.br/pdf/artigo_001.pdf.

AudioMath – using MathML for speaking mathematics

Helder Ferreira, Diamantino Freitas

Laboratório de Sinais e Sistemas, Faculdade de Engenharia Universidade do Porto
R. Dr. Roberto Frias s/n,
4200-465 Porto, Portugal
{hfilipe,dfreitas}@fe.up.pt
http://lpf-esi.fe.up.pt

Abstract. How can blind people surpass the difficulty in reading online documents containing mathematical expressions? Usually they can't. Technical, scientific or even simple documents presented online that involve math expressions, issue a big accessibility problem. One possible solution is to create the contents using mark-up and in the retrieval phase to parse and interpret such contents and convert them into an audio format. This remains a complex, multidisciplinary problem, currently addressed by the AudioMath project, providing conversions from math expressions in W3Cs MathML format into marked-up text for text-to-speech engine understand and read-out. This paper reviews the state-of-the-art in publishing Web scientific documents, introduces the accessibility problems and reviews the problem of speaking mathematics. The main scientific and technical challenges, specifically at the levels of interpretation of the XML content of the math expressions and of conversion into spoken form are considered, as well as the current status of the work.

1 Introduction

Since the beginning of times that mankind feels the need to communicate. Language allows us to structure thinking, transmit feelings, record what we know and communicate with each others. The publication and distribution of scientific papers and articles, as well as e-learning websites, is a desirable way to spread knowledge, promote education and stimulate research and development.

Nowadays this knowledge is not only available as books, magazines or conference proceedings, but also as web pages or digital files on the Internet, reachable by "almost" everyone. The upcoming of Internet improved the availability and access to information, but introduced new problems. A web document can contain: text, images (graphics, diagrams or pictures), mathematical expressions, applets, scripts, multimedia (Flash, Shockwave…) or code blocks. Each one of these elements might present accessibility problems in the reading and interpretation of the document.

Although the existence of the Web Content Accessible Guidelines (WCAG) [1] from W3C, that provide a set of rules which can be applied to resolve these problems, at this time the accessibility of technical and scientific documents, especially on mathematical expressions, is extremely weak.

Most of the web documents containing mathematical materials utilize images (jpg, gif, png) created by TeX related software or Microsoft Word, to display the mathematical formulae. Some distribute the documents in PDF, TeX, Postscript, Word or RTF file types. The use of Java applets or plug-ins is also frequent.

This has lead to accessibility problems in the reading of documents containing mathematical expressions for visually impaired persons, and, in particular, for the blind. Their difficulty rises and accessibility diminishes, as the technical level of documents increases, even if the contents are already in a digital format and properly equipped computers are used [2].

The growing concern about accessibility to mathematical expressions on the Internet has taken research groups such as: W3C [3], OpenMath [4] and AMS [5], to create and develop initiatives in this area, promoting and regulating the accessible publication of mathematical contents over the Internet. One of those initiatives is the creation of the *MathML (Mathematical Mark-up Language)* [6] from W3C, which is used in this project.

The work presented in this paper refers to the problem of online accessibility to mathematical expressions and presents the tool *AudioMath*, which uses MathML and text-to-speech (TTS) technology to create audio versions of the mathematical contents. This technical solution has been adopted taking into consideration two principles: first, the audio medium is accessible and general purpose TTS engines are available; and second, it's assumed that math can be spoken out.

This paper is structured in the following sections: - Section 1 introduces the problem; - Section 2 reviews the current state of art; - Section 3 presents some considerations about MathML audio rendering; - Section 4 makes a brief overview analysis on 2 related tools; - Section 5 describes the tool AudioMath; and Section 6 presents the conclusions and the forthcoming work.

2 State of the art

This section reviews the current state of art in publishing mathematical contents online, and the current related projects in mathematical audio rendering.

2.1 Publishing technical documents in the Internet

The publication of scientific documents containing mathematical formulae is extremely demanding. The appearance of the TeX [7] system developed by Donald Knuth solved the majority of problems with printed documents. Then WYSIWYG (*What You See Is What You Get*) editor's appeared and subsequently mark-up languages for Internet, such as HTML (*Hypertext Mark-up Language*). However, HTML *per se* doesn't allow the use of a mathematical description language directly into the document. Being so, alternatives were developed, such as the use of:

- **(X)HTML** [8] **+ Images (JPG, GIF, PNG, SVG)**: the use of math expressions as images (vector or raster) is non accessible because usually don´t provide a text equivalent.
- **HTML + Symbol Fonts / (X)HTML + CSS** [9]: usually they use tables to structure information. No semantic meaning is provided. Ex: *translator TtH*. [10].
- **Applets**: these are used to generate mathematical expressions; it's a slow and non accessible process. Ex: *WebEQ*. [11].
- **Word / RTF / PDF / Postscript / TeX / Latex**: the HTML documents produced by these materials usually represent math expressions in the form of images. Ex: *TeX4tht* [12] (latest versions can now create MathML).
- **Mathematical Markup Language (MathML)**: one of the most accessible solutions.

MathML is becoming one of the most adopted formats for publishing math online since it is:

- Being developed by W3C and is becoming a standard.
- Rapidly growing and being used by several relevant organizations associated with the teaching and learning of mathematical contents, as well as the increasing involvement of software houses.
- Supported by several editors and applications able to create and manipulate MathML documents.
- The existence of conversion tools for the main publishing formats.
- The fact that it is a mark-up language allows its parsing, interpretation and conversion to other formats, and consequently a higher accessibility, portability and independence – it's XML [13]!

For all this reasons, MathML is one of the best supporting technologies for publishing online, manipulating formulae and speaking mathematics.

2.2 Audio rendering of mathematical expressions

It is known that most people read text letter-by-letter beginning at or near the leftmost symbol in a text line and often engage in backward scans to retrieve information that was previously read. But reading mathematics is very different than reading a plain text, the first one is a lot more complex. Reading math like a normal book doesn't allow us imagine the elements and the relationships between them. Formal structures and rules are required.

To start with, mathematics is a two-dimensional written language. This is quite different from almost all the other ordinary languages that are primarily spoken and later on written, and both in a one-dimensional form (in a clearly defined sequence).

For a non-visual person, understanding a mathematical formula requires a repeated scan and jumping over secondary portions. However, this can be a very complex task to blind people [14]; therefore studies should be done to understand how we should provide a correct access to math expressions. Unfortunately there is an almost com-

plete lack of studies on how to read mathematical expressions, mainly on mathematical prosody [15] [16].

The *MathSpeak Project* [17] is one of the proposed methods, consisting of a group of rules to dictate mathematical contents. However it is not a standard and it is intended to serve blind people that want to transcribe their documents into *Nemeth Code* [18], and later on into Braille.

Another method comes from the work of *T.V. Raman* [19], the most impressive and recognized study in this area (Aster system), consisting of a mathematical notation that takes into consideration several audio dimensions that make up the various pieces of the notation. However this only works for TeX related documents and has limitations, such as no provision for intra-formula navigation in the mathematical expression. Nevertheless, this work points out several generally useful clues for audio rendering of mathematics.

Arthur Karshmer is another influent researcher working in the area that is trying to study how to read mathematical expressions to blind students [20].

Design Science [21] is also working with audio rendering of MathML. Its product MathPlayer version 2.0 [22] is an example of it. However this plug-in has ambiguous rendering in some mathematical expressions. It's therefore a work in progress.

3 Audio rendering of MathML

In section 2.1 it was stated that MathML is one of the best supporting technologies for publishing online, manipulating formulae and speaking mathematics. Therefore, it is the authors' intention to make some considerations about using MathML for mathematical audio rendering.

3.1 Which MathML markup to use?

W3C's MathML Specification [23] states that "*MathML allows authors to encode both the notation which represents a mathematical object and the mathematical structure of the object itself.*" In other words, representation of a math expression is perceived by two distinct but associated concepts:

- *Visual structure or notation* (ex: a/b, a^{-1}) – *Presentation Markup*: since it concerns the notation it is ambiguous on the semantics. Adaptation for audio rendering is possible.
- *Meaning it represents* (ex: a divided by b) – *Content Markup*: since it is used for semantic meaning, it is ambiguous on the notation. Because it encloses the semantics, it is in theory the best to use for audio rendering.

The relationship between notation (Presentation) and meaning (Content) is not univocal: one meaning, more than one notation.

Also, it seems that Content Markup would suit better for audio rendering, however there are 2 major open issues: Content markup only covers basic math; and the majority of published MathML online is using Presentation markup.

Therefore, MathML Presentation Markup is for the moment the best choice. The downside is that requires a relatively much higher effort in the interpretation of given mathematical expressions, since it is prepared for the enclosure of visual information and not of semantics.

3.2 Interpretation of MathML tag set elements and attributes

Not all the elements and attributes need to be processed for audio rendering. For instance, styles and visual attributes not always provide extra information about the expression, and rarely enhance the audio description. However if Presentation markup is used some style attributes might be important to disambiguate meaning. For example, consider the following expression:

$$\left(\frac{2}{3}\right) \neq \left(\begin{matrix}2\\3\end{matrix}\right). \quad \textbf{(1)}$$

The left side of the expression is using the `mfrac` element with the attribute `linethickness` different from zero (becoming a fraction), but the right side of the expression uses `linethickness` equal to zero (changing the meaning of the expression to combinatorial number).

Other elements and attributes are important in the Presentation markup, for example: `mathvariant` (because different types of variants have different meanings, for example: *italic* might indicate a function, *bold* might indicate a vector, *fraktur* might indicate lie algebra) and `mrow` (which is usually used to enclose mathematical sub expressions, giving precious clues to audio rendering).

When using Content markup, almost every elements are useful once they all enclose semantic meaning about the mathematical expression.

MathML also has some special characters that are extremely important for audio rendering: `&ApplyFunction;` (allows an audio render to know it's a function of something) and `&InvisibleTimes;` (allows an audio render to know it is a product even if the symbol is not there). For example:

```
xy - <mi>x</mi><mo>&InvisibleTimes;</mo><mi>y</mi>
```

Without the `&InvisibleTimes;` it will render "*x y*". With the special character it will render "*x times y*".

4 Mathematical Audio Rendering Tools Overview

This section gives a brief overview of the only tools, besides AudioMath, that are able to produce mathematical audio rendering: ASTER and MathPlayer.

4.1 Audio system for technical readings - ASTER

ASTER stands for *Audio System For Technical Readings* [19] and is an application that accepts TeX notation (LaTeX [24]) as input and produces audio rendering as output. It has been developed by T.V. Raman in 1994 during his PhD and uses an Emacs [25] front-end (Linux platform). ASTER has 3 main components: *Latex parser* that creates an internal representation easier for the program to manipulate; AFL *(Audio Formatting Language)* which is used to render the parsed text using speech and other sounds; and *Browser* used to help the audio rendering.

ASTER was a breakthrough. TV Raman's work it's considered a bible in mathematics audio rendering and it supports a large number of mathematical formulae in Latex. However no math formulae navigation is supported. To suppress this need, ASTER uses a variable substitution process: complex expressions can be divided in sub expressions on user's request.

4.2 MathPlayer 2.0

MathPlayer is a mathematics display engine for Microsoft's Internet Explorer 6.0, developed by Design Science. It uses MathML Presentation markup as input and visual rendering (version 1.0 and 2.0) and audio rendering (only in version 2.0 out in 2004) as output.

Since MathPlayer is too recent, it has a few problems yet to be solved: system of equations and matrices are detected both as tables (it is not capable to detect a more detailed semantic meaning), and there is a lack of some `begin <operator>` and `end <operator>` keywords, making the audio result sometimes ambiguous. Also, no math formulae navigation is provided, so it gets complicated with complex math expressions; and there are no *usermodes* (the way people like to hear something).

5 AudioMath

Since the first appearance of MathML (1999), and the emerging support to it by several browsers, that the future of publishing mathematics online seems to have assured a certain degree of quality (better browsing, presentation, manipulation, flexibility and accessibility). Even the previous works in LateX are now being converted into XHTML+MathML.

For these reasons, the need to build an accessibility tool that would be able to deal with MathML was justified. AudioMath was the first tool, as far as the authors know, to be able to speak MathML contents. Only months later, Design Science included the feature "Speak Math" on MathPlayer 2.0.

This section describes the tool, *AudioMath* [26], that the Laboratory of Signals and Systems from the Faculty Engineering University of Porto is developing to enable MathML audio rendering.

5.1 The AudioMath project

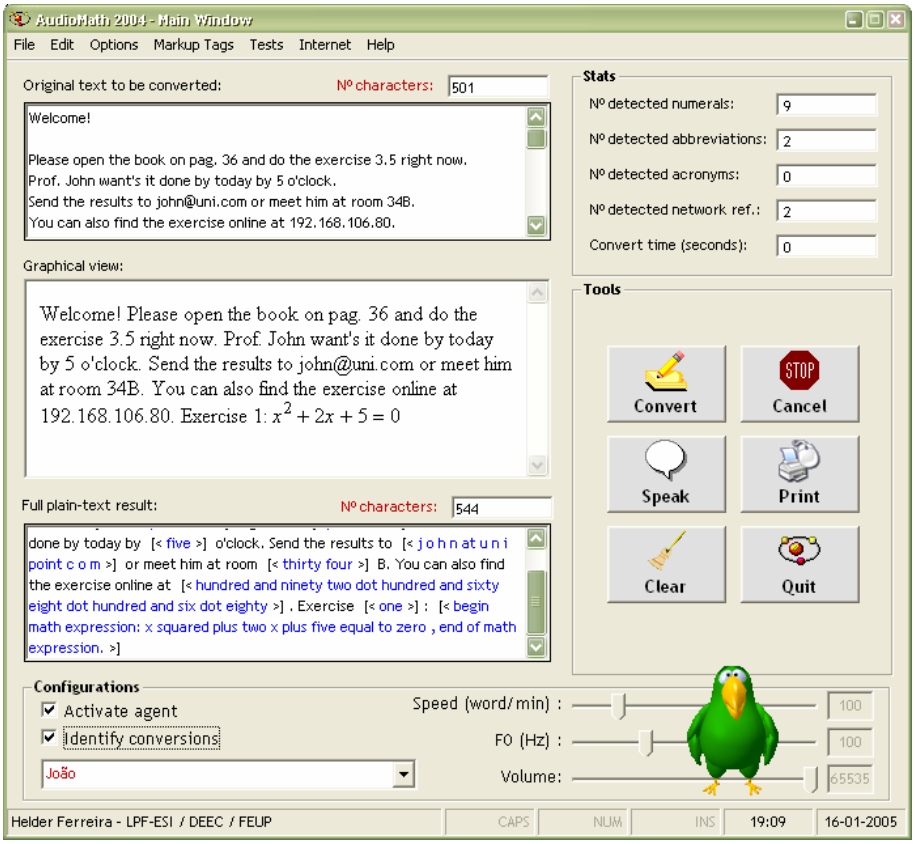


Fig. 1. AudioMath Screenshot – English version.

The AudioMath project aims is to build a tool to operate either as standalone or integrated in a speech interface (TTS text-to-speech), capable of:

– *Mathematics Audio Rendering*: parsing, interpretation and conversion of MathML into plain text format; generation of the appropriate prosodic contour for reading of the formula's text; and development of an intra-formula browsing device (navigation).
– Recognition and conversion of any other text or markup elements not directly "understandable" by the TTS Engine, such as: numerals, abbreviations, acronyms and network references.

AudioMath is an ActiveX dynamic link library (dll) that can be used by any program through internal calls. It is currently developed in PERL [27] and for Windows environment. However its main modules can also be used in Linux/Unix due to PERL's portability.

Its main applications are: reading of technical and scientific documents online in an accessible way; teaching and learning how to read math; and enhancing general accessibility to computer-based applications, when applied to a TTS engine.

5.2 AudioMath architecture

AudioMath has been built in a modular, extensible and configurable architecture. Currently it contains 6 major modules: *Numerals* (conversion of several types of numeric forms), *Abbreviations* (conversion of abbreviations in a text), Acronyms (conversion of acronyms in the document), *Network References* (conversion of IPs, emails and URI/URLs), *Mathematical* (conversion of MathML expressions) and *Auto-Discovery* (the "brain" of the operation that recognizes or identifies elements in the document and calls the respective conversion modules).

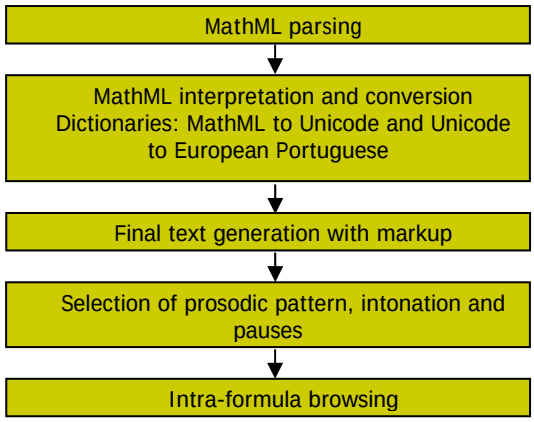


Fig. 2. AudioMath Architecture Overview.

5.3 The AudioMath process

The AudioMath's process can be divided into 3 functional parts: *Text Analysis*, *Parsing and Interpreting MathML* and *Converting and Speaking Mathematical Contents*.

There are several types of text elements in a document (acronyms, abbreviations, numerals, web references, math expressions or even special Unicode characters or math glyphs [28]), justifying the reason for text analysis algorithms.

To speed up the process the document should be divided into blocks of text, splitting MathML markup from the rest of the text. Text processing is strongly based on regular expressions [29] [30] and databases (for acronyms and abbreviations). Dictionaries and databases were included inside the code as hash tables (more speed but less flexibility in updates). All the conversion algorithms support *user modes*, i.e., preferences on audio rendering. For example: 1.25 can be spoken: *one point two five*, or *one point twenty-five*.

Currently, in AudioMath, interpreting Presentation markup is a process of raising flags each time a starting and ending tag is detected (SAX parser – event based), which allows to know the history of the markup and to retrieve information, enabling to understand the structure of the math expression and do its conversion. The identification of sub-expressions and its conversion also allows the composition of a bigger and complex math formula. As the expression is becoming discovered, the conversion process takes place by calling several algorithms as well as Unicode and MathML dictionaries. AudioMath uses 2 kinds of dictionaries: MathML entities to Unicode (`&InvisibleTimes;` is converted to `U+02062`) and Unicode to European Portuguese full plaintext form (`U+02062` is converted to `vezes(=times)`). The conversion to the full plaintext form is done according to a database of vocabulary and speech rules.

5.4 Database of vocabulary and speech rules

One of the AudioMath's current tasks is the definition of a database of vocabulary and speech rules, for several subsets of math formulae. These rules and intonation will be implemented at conversion time by tagging the text (using SSML [31]) with prosodic marks, to command the TTS engine in order to produce the required pauses and f0 modulations.

Math corpus is being defined and categorized by different semantic types: basic algebra, integrals, roots, powers, derivatives, complex numbers, matrices, systems of equations, trigonometry, series, theorems and definitions, Greek alphabet, *etc*.

Each type has its structure/concept analysed and defined in a formal plaintext way. Different readings of the same formula are spoken and analyzed for prosody and perceiveness (user evaluation). Pitch patterns and pauses are inferred from speech. For example, consider the following expression:

$$\sqrt{a^2 + b^2}\ . \tag{2}$$

Can be spoken as: "*square root of (pause) a squared (mini-pause) plus b squared (pause) end of radicand*".

We can see that there are 2 types of pauses: *large pause* - precede lower levels in hierarchy; *short pause* – optional, but used before some operators and between the arguments.

Also, there are rising and falling movements of f0 in the speaker's intonation intended to provide classification of the boundaries introduced by the pauses: - *rising tone*: used when lower hierarchical level is starting (ex: root of); - *falling tone*: used when level is ended (ex: b squared); - *falling and rising tone*: used to clarify the smaller separating pause (ex: a squared); and *emphatic falling tone*: used at the end of the expression (ex: end of radicand).

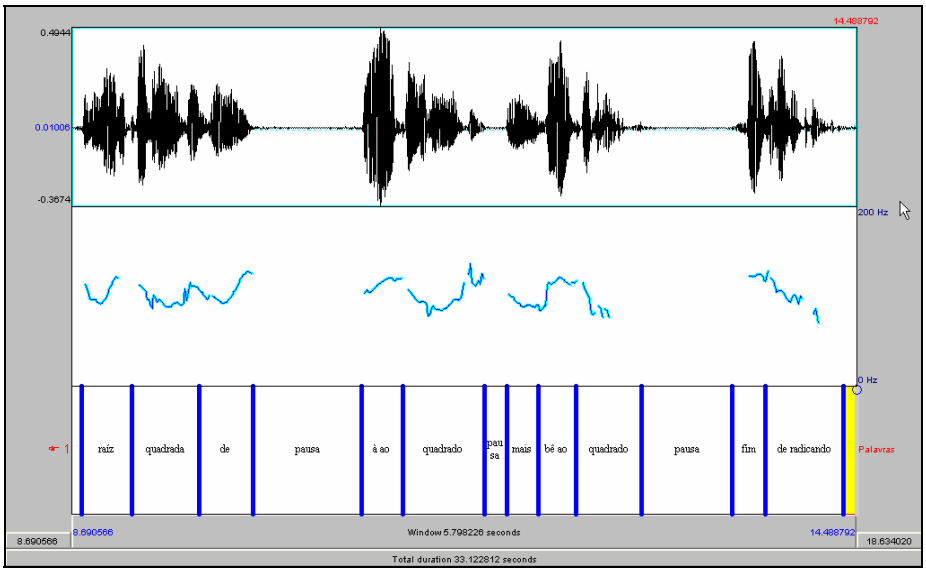


Fig. 3. Speech waveform of the previous mathematical formula example (2).

5.5 Current status and future work

In spite of the vast research this project has done in the field of speaking math using MathML, AudioMath is still a work in progress.

The tool currently supports several tags of MathML Presentation Markup and Unicode, and it is also able to detect and convert several types of numerals (cardinals, ordinals, roman numbers, dates…), abbreviations, acronyms and network references (emails, ip addresses …). Each module, used for detection and conversion, supports user preferences on the MathML audio rendering.

A browser has been built for functionality test and TTS integration. User evaluation is also performed in several iterations of the AudioMath's development, especially in the building of its mathematical corpus and rules for reading math. A few studies on mathematical prosody have also been made.

AudioMath's future work includes adding support for MathML Content Markup; completing the support to the MathML Presentation Markup; further learning and surveying on how to read mathematical formulae; develop modules that support HTML, XHTML, SSML and others; providing mechanisms for navigating inside mathematical formulae (eventually a special audio browser); adding support for new languages (English, French…); and finally, to develop the study on the prosody of reading mathematics.

6 Conclusions

The research developed along this project allowed us to conclude that, reading mathematical contents is very different than reading a text. The full verbal understanding of a math structure and its meaning requires, on the majority of cases, the presence of some kind of navigational mechanism; otherwise users will experience serious difficulties due to the human short-term memory. Also, the use of prosody should be combined with those mechanisms, to ensure a more natural and perceivable experience to the users. However, there is currently lack of studies on math prosody, which means there is yet no standard on how to read math!

The increase of applications and online materials that deal with MathML motivate the use of this markup language on tools that have in mind the audio rendering of math. The choice of the markup set that one should adopt, for MathML audio rendering, is not an easy one. In theory, Content Markup should be better, once it encloses the semantic meaning of the math expression; however Presentation Markup is more widely used, becoming the first choice for the moment. Also, the developer should take into consideration that not all the elements and attributes of the 2 markup sets are needed for the audio rendering (attributes that convey colour information, for instance).

The solution presented here, by AudioMath, concerns a work in progress of an accessibility tool that creates audio versions of the MathML Presentation Markup mathematical contents, with user modes support, i.e., it allows users to configure the MathML rendering to their preferences. AudioMath is built in a modular, extensible and configurable architecture that allows it to be upgraded easily. The use of special dictionaries, one for MathML entities and other for Unicode entities, allows this tool to adapt to several online realities. This project also studies the mathematical prosody and it's committed to the building of a speech rules database.

In conclusion, the almost lack of references, studies and standards on "how to use MathML for speaking mathematics" and "how to read math", prove the need and importance of this project and its work in progress to build the tool AudioMath [32][33][34].

References

1. W3C: Web Content Accessible Guidelines 1.0 (1999) - http://www.w3.org/TR/WCAG10/ .
2. Monaghan, Alex. Fitzpatrick, Donal. *Browsing Technical Documents: Document Modelling And User Interface Design*. BULAG99 Proceedings. 1999.
3. World Wide Web Consortium (W3C) – http://www.w3.org .
4. The OpenMath Group - http://www.openmath.org/cocoon/openmath//index.html .
5. American Mathematical Society (AMS) – http://www.ams.org .
6. Mathematical Markup Language (MathML) – http://www.w3.org/Math .
7. TeX Users Group Homepage – http://www.tug.org .
8. XHTML 1.1 - http://www.w3.org/TR/xhtml11/ .
9. CSS 2.1 - http://www.w3.org/TR/CSS21/ .
10. Translator Tth Homepage – http://hutchinson.belmont.ma.us/tth .

11. Design Science WebEQ Homepage - http://www.dessci.com/en/products/webeq/ .
12. TeX4ht - http://www.cse.ohio-state.edu/~gurari/TeX4ht/mn.html .
13. Extensible Markup Language Specification - http://www.w3.org/TR/REC-xml/ .
14. Stöger Bernhard, et al. *Mathematical Working Environment for the Blind Motivation and Basic Ideas*. ICCHP04 Proceedings. 2004.
15. Fitzpatrick D. and Karshmer A.I. *Multi-modal Mathematics: Conveying Math Using Synthetic Speech and Speech Recognition*. ICCHP04 Proceedings. 2004.
16. Stevens R., *Principles for the Design of Auditory Interfaces to Present Complex Information to Blind People*. PhD thesis. Department of Computer Science, January 1996.
17. MathSpeak - http://www.rit.edu/~easi/easisem/talkmath.htm .
18. Learn Nemeth Code – http://www.tsbvi.edu/math/math-nemeth.htm .
19. T.V. Raman. *Audio System for Technical Readings*. Dissertation to the Faculty of the Graduate School of Cornell University. 1994. Home Page: http://www.cs.cornell.edu/Info/People/raman/raman.html .
20. Karshmer, Arthur. *How Well Can We Read Equations to Blind Mathematic Students?* HCII2003 Proceedings Volume 4. 2003.
21. DesignScience Home Page - http://www.dessci.com/ .
22. MathPlayer 2.0 - http://www.dessci.com/en/products/mathplayer/ .
23. MathML W3C's Recommendation 2.0 - http://www.w3.org/TR/2003/REC-MathML2-20031021/ .
24. LateX - http://www.latex-project.org/ .
25. GNU Emacs - http://www.gnu.org/software/emacs/emacs.html .
26. AudioMath Home Page – http://lpf-esi.fe.up.pt/~audiomath .
27. Practical Extraction and Report Language - http://www.perl.org/ .
28. Unicode characters and math glyphs, MathML, W3C - http://www.w3.org/TR/2003/REC-MathML2-20031021/chapter6.html#chars.mathmlchars .
29. Friedl, Jeffrey. *Mastering Regular Expressions 2nd Edition*. O'Reilly. 2002. ISBN: 0-596-00289-0.
30. Jurafsky, Daniel and Martin, James. *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice-Hall. 2000. ISBN: 0-13-095069-6.
31. Speech Synthesis Markup Language (SSML) - http://www.w3.org/TR/speech-synthesis/ .
32. Ferreira, Helder. *AudioMath: Speaking Mathematics with MathML*. 2nd European Workshop on MathML & Scientific e-Contents. 2004.
33. Ferreira, Helder and Freitas, Diamantino. *Audio Rendering of Mathematical Formulae using MathML and AudioMath*. UI4ALL Proceedings. 2004.
34. Ferreira, Helder and Freitas, Diamantino. *Enhancing the Accessibility of Mathematics for Blind People: The AudioMath Project*. ICCHP04 Proceedings. 2004.

*p*GML – estudo de um subconjunto ”Preciso”do GML2.12

Mário Ricardo de Novais Henriques

INESC-Porto, Rua Dr. Roberto Frias, n.378 4200-465 PORTO, Portugal,
ricardo.henriques@inescporto.pt,
http://www.inescporto.pt

Resumo A área dos Sistemas de Informação Geográfica (GIS) é das áreas da produção de *software* onde a necessidade de obter soluções sem falhas, dentro dos prazos e orçamentos é mais imperativa – não só pelo número de pessoas que directamente podem ser afectadas, como pelos custos e meios envolvidos. A par deste objectivo, há percepção de que estes sistemas dificilmente operam em máquinas modestas, dadas as descrições dos elementos do mundo real serem representadas por enormes colecções de entidades gráficas de baixo nível, e tal motiva a questão de saber se as ferramentas comerciais de GIS operam no nível correcto de abstracção.
É entre estes dois objectivos que surgiu o estudo de um subconjunto do GML, como forma de obter, por engenharia reversa, um modelo formal descrito numa notação conceituada e *standard*, o VDM-SL. Obtido o modelo, conseguiu-se uma representação abstracta e ao mesmo tempo capaz de ser utilizada como protótipo,[1] com o núcleo da funcionalidade para representação de um Sistema de Informação Geográfica, sobre o qual se pode raciocinar e derivar a implementação. A capacidade de raciocinar nesta fase é essencial de forma a evitar a propagação de erros e inconsistências do modelo que, a ser detectado só durante implementação final, provocariam atrasos e custos muito acrescidos.

Palavras-chave: Métodos formais, GML, GIS, VDM-SL.

1 Introdução

Há um quarto de século o eminente cientista da computação, John Backus escreveu [1] um artigo revolucionário que se tornaria um *ex-libris* da moderna ciência de computação. Na essência, o artigo proclamava obsoletas as linguagens de programação orientadas ao comando devida à sua ineficiência, por não serem independentes a alto nível, do padrão do modelo computacional subjacente – a arquitectura de Von Neumann. Alternativamente, o já na época existente estilo de programação funcional devia ser a opção, por duas razões: em primeiro lugar por potenciar a computação paralela e concorrente, e em segundo por ser fortemente baseada em estruturas algébricas. Backus salientou que apenas teriam sucesso as linguagens que possuíssem uma álgebra para raciocínio sobre os objectos que semanticamente descrevem. O impacto do primeiro argumento de Backus nas comunidades da ciência da computação e da computação paralela foi significativo.

[1] Através da utilização das VDM-Tools da IFAD, por exemplo.

Por contraste, o seu segundo argumento para alterar a computação foi em larga medida ignorado. De facto, apenas a minoria dos investigadores em *program calculi* seguiu esta direcção.

Num trabalho publicado em 1998 ficou demonstrado [12], na área do *software* de CAD, ser possível introduzir estruturas algébricas de alto nível capazes de abstrair entidades gráficas de baixo-nível como pontos, linhas e segmentos. Um CAD é um bom exemplo de uma tecnologia suportada pela computação, onde não se tinha evoluído muito na direcção do artigo de Backus, apesar de toda a evolução a nível de gráficos 3D, realidade virtual e multimédia.

Para além da representação da realidade, através de colecções de entidades gráficas de baixo nível, os GIS dispõem da capacidade de registo e análise topológicas. É natural transpor para o domínio dos GIS as mesmas preocupações:

- As ferramentas comerciais de GIS estão a trabalhar no nível correcto de abstracção?
- As descrições de elementos reais estão condenadas a serem representadas por imensas colecções de entidades gráficas (geométricas) de baixo nível?
- Ou possuem uma estrutura algébrica intrínseca passível de raciocínio?

No sentido de conseguir uma resposta a estas preocupações/questões, foi investigada a possibilidade de obter um modelo com o nível de abstracção considerado adequado, baseado no rigor dos métodos formais.

2 Contexto e objectivos

A propriedade mais importante de um sistema de modelação é a sua vocação para permitir a análise e raciocínio, aliando o rigor à abstracção. Nesse intuito foi utilizado o VDM-SL (*specification language*)[5], o qual resulta da evolução do VDM concebido nos laboratórios da IBM em Viena em 1973, cuja origem remonta a 1968, onde os participantes da *NATO Conference on Software Engineering* anteviram uma disciplina de desenvolvimento de *software* suportada por uma forte base científica.

Reconhecendo o GML (*Geography Markup Language*) como uma plataforma rica, aberta e que serve de denominador comum aos formatos proprietários para representação de informação geográfica, foi natural procurar por engenharia reversa obter um modelo na notação formal do VDM-SL e com isso investigar a possibilidade de obter um modelo capaz de responder às preocupações enunciadas na secção anterior. O resultado foi a obtenção do *p*GML (*precise* GML), mais conciso e enriquecido com invariantes topológicos que o GML por si só não retém.

A seguinte figura apresenta em (a) e (b) os elementos contextuais relevantes. (a) representa num sentido a **formalização** do GML, na qual se obtém o enriquecimento semântico do GML através da introdução de invariantes. No outro sentido há a exportação do modelo formal para o domínio do XML. Em (b) consegue-se a acomodação de "*domain specific languages*"de MF (Métodos Formais) aos GIS (*Geography Information System*). Tal traduz-se, num sentido, na formalização de um subconjunto do

domínio dos GIS numa notação formal e obtenção de um modelo – neste caso o *p*GML (*precise*GML). No sentido inverso abre-se o caminho para a obtenção de novos paradigmas GIS mediante a obtenção de novas derivações do modelo formal.

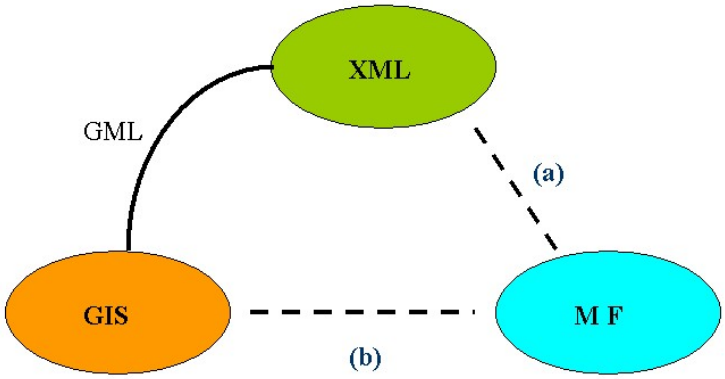


Figura 1. Domínios de desenvolvimento – objectivos

3 Obtenção do *p*GML

3.1 Engenharia reversa sobre o GML

O modelo geográfico proposto pelo OGC (`Open Geospatial Consortium`) apresenta-se sob a forma de uma especificação abstracta, o GML (*Geography Markup Language*), no qual se define uma entidade geográfica como uma "*abstracção de um fenómeno do mundo real; é uma entidade geográfica se estiver associada a uma localização relativa à Terra*". Desta forma, o mundo real pode ser representado através de um conjunto de entidades. O estado de cada entidade é definido por um conjunto de propriedades, as quais se definem por um triplo – nome, tipo e valor. A definição do tipo de entidade restringe-a quanto ao número de propriedades e tipos de propriedades que uma entidade desse tipo pode ter. O OGC indica também que as entidades geográficas se caracterizam por possuírem propriedades geométricas. E acrescenta que um conjunto de entidades pode ser visto como uma entidade em si só; por isso, uma colecção de entidades tem um tipo e propriedades particulares para além das que contém nas entidades contidas.

A W3C[14] revela que os XML *Schemas* expressam um vocabulário comum que permite as máquinas interpretarem regras e normas humanas. O OGC[3], ao especificar o GML através de XML *Schemas*, faculta a interpretação automática da norma que define, por exemplo através da utilização do mecanismo de transformação associado ao XML, XSLT (*eXtensible Stylesheet Language for Transformations*)[6][13], uma vez que um XML *Schema* é também ele próprio um documento XML.

O primeiro passo, do processo de engenharia reversa, traduziu-se na definição de uma função de transformação, um documento XSLT, no qual se aproxima a definição dos *feature.xsd*, *geometry.xsd*, *xlinks.xsd* que compõem o GML, numa primeira definição dos respectivos tipos em VDM-SL.

3.2 Obtenção de um subconjunto preciso do GML 2.12

Obtida a definição dos tipos em VDM-SL impunha-se a necessidade de analisar a sua definição formal e com isso investigar a possibilidade de obter um modelo compatível com o anterior, mais simplificado e, portanto, com um grau de abstracção superior, o *p*GML. Para o conseguir procedeu-se ao refinamento da especificação obtida, no passo anterior, pela ordem *natural* que o GML induz, ou seja, partindo da noção abstracta de entidade e da sua caracterização, até obter as suas propriedades geométricas e a respectiva definição de tipos (ou classes) dessas propriedades.

O resultado dessa análise traduz-se numa especificação em VDM-SL que aqui se apresenta em XML, após automática conversão da definição de tipos VDM-SL para XSD. O processo de construção do *p*GML, a sua sintaxe, semântica e o mecanismo de extracção do XSD e XML a partir de uma especificação VDM-SL, podem ser aprofundados em [4].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:gml="http://www.opengis.net/gml" xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:import namespace="http://www.opengis.net/gml" schemaLocation="feature.xsd"/>
<xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlinks.xsd"/>
<xs:complexType name="AbstractFeatureType">
   <xs:sequence><xs:element name="d" type="description" minOccurs="0" maxOccurs="1"/>
    <xs:element name="n" type="name" minOccurs="0" maxOccurs="1"/>
    <xs:element name="bb" type="boundedBy" minOccurs="0" maxOccurs="1"/>
    <xs:element name="fid" type="ID" minOccurs="0" maxOccurs="1"/></xs:sequence>
</xs:complexType>
<xs:complexType name="AbstractFeatureCollectionType">
   <xs:sequence><xs:element name="d" type="description" minOccurs="0" maxOccurs="1"/>
    <xs:element name="n" type="name" minOccurs="0" maxOccurs="1"/>
    <xs:element name="bb" type="boundedBy"/>
    <xs:element name="fid" type="ID" minOccurs="0" maxOccurs="1"/></xs:sequence>
</xs:complexType>
<xs:complexType name="PointPropertyType">
 <xs:choice><xs:element name="selector0" type="Point"/>
  <xs:element name="selector1" type="LinkPoint"/></xs:choice>
</xs:complexType>
<xs:complexType name="Point">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI" minOccurs="0" maxOccurs="1"/>
    <xs:element name="p" type="coord"/></xs:sequence>
</xs:complexType>
<xs:complexType name="PolygonPropertyType">
 <xs:choice><xs:element name="selector0" type="Polygon"/>
  <xs:element name="selector1" type="LinkPolygon"/></xs:choice>
</xs:complexType>
<xs:complexType name="Polygon">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI" minOccurs="0" maxOccurs="1"/>
    <xs:element name="outerBoundaryIs" type="LinearRing"/>
    <xs:element name="innerBoundaryIs" type="LinearRing" minOccurs="0"
        maxOccurs="unbounded" /></xs:sequence>
</xs:complexType>
<xs:complexType name="LinearRing">
 <xs:sequence><xs:element name="selector" type="coord"  minOccurs="4"
    maxOccurs="unbounded"/></xs:sequence>
</xs:complexType>
<xs:complexType name="LineStringPropertyType">
 <xs:choice><xs:element name="selector0" type="LineString"/>
  <xs:element name="selector1" type="LinkLineString"/></xs:choice>
</xs:complexType>
<xs:complexType name="LineString">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
```

```
    <xs:element name="srsName" type="anyURI" minOccurs="0" maxOccurs="1"/>
    <xs:element name="l" type="coord"  minOccurs="2"  maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name="MultiPointPropertyType">
 <xs:choice><xs:element name="selector0" type="MultiPoint"/>
  <xs:element name="selector1" type="LinkMultiPoint"/></xs:choice>
</xs:complexType>
<xs:complexType name="MultiPoint">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI"/>
    <xs:element name="mp" type="Point" minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name="MultiLineStringPropertyType">
 <xs:choice><xs:element name="selector0" type="MultiLineString"/>
  <xs:element name="selector1" type="LinkMultiLineString"/></xs:choice>
</xs:complexType>
<xs:complexType name="MultiLineString">
   <xs:sequence>
    <xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI"/>
    <xs:element name="mls" type="LineString" minOccurs="0"
        maxOccurs="unbounded" /></xs:sequence>
</xs:complexType>
<xs:complexType name="MultiPolygonPropertyType">
 <xs:choice>
  <xs:element name="selector0" type="MultiPolygon"/>
  <xs:element name="selector1" type="LinkMultiPolygon"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="MultiPolygon">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI"/>
    <xs:element name="mpol" type="Polygon" minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name="MultiGeometryPropertyType">
 <xs:choice><xs:element name="selector0" type="GeometryCollection"/>
  <xs:element name="selector1" type="LinkMultiGeometry"/></xs:choice>
</xs:complexType>
<xs:complexType name="GeometryCollection">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI"/>
    <xs:element name="collection" type="MultiGeometry" minOccurs="0"
        maxOccurs="unbounded" /></xs:sequence>
</xs:complexType>
<xs:complexType name="MultiGeometry">
 <xs:choice><xs:element name="selector0" type="Point"/>
  <xs:element name="selector1" type="LineString"/>
  <xs:element name="selector2" type="Polygon"/>
  <xs:element name="selector3" type="MultiPoint"/>
  <xs:element name="selector4" type="MultiLineString"/>
  <xs:element name="selector5" type="MultiPolygon"/>
  <xs:element name="selector6" type="GeometryCollection"/></xs:choice>
</xs:complexType>
<xs:element name="LinkPoint" type="AssociationAttributeGroup"/>
<xs:element name="LinkPolygon" type="AssociationAttributeGroup"/>
<xs:element name="LinkLineString" type="AssociationAttributeGroup"/>
<xs:element name="LinkMultiPoint" type="AssociationAttributeGroup"/>
<xs:element name="LinkMultiLineString" type="AssociationAttributeGroup"/>
<xs:element name="LinkMultiPolygon" type="AssociationAttributeGroup"/>
<xs:element name="LinkMultiGeometry" type="AssociationAttributeGroup"/>
<xs:complexType name="AssociationAttributeGroup">
   <xs:sequence><xs:element name="xlink" type="simpleLink"/>
    <xs:element name="rs" type="remoteSchema" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="coord">
   <xs:sequence><xs:element name="X" type="xs:long"/>
    <xs:element name="Y" type="xs:long" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Z" type="xs:long" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
```

```
 </xs:complexType> <xs:simpleType name="anyURI">
 <xs:restriction base="string"/>
</xs:simpleType>
<xs:simpleType name="remoteSchema">
 <xs:restriction base="anyURI"/>
</xs:simpleType>
<xs:simpleType name="description">
 <xs:restriction base="string"/>
</xs:simpleType>
<xs:simpleType name="name">
 <xs:restriction base="string"/>
</xs:simpleType>
<xs:simpleType name="ID">
 <xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
<xs:simpleType name="string">
 <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="href">
 <xs:restriction base="anyURI"/>
</xs:simpleType>
<xs:simpleType name="role">
 <xs:restriction base="anyURI"/>
</xs:simpleType>
<xs:simpleType name="arcrole">
 <xs:restriction base="anyURI"/>
</xs:simpleType>
<xs:simpleType name="title">
```

```
 <xs:restriction base="string"/>
</xs:simpleType>
<xs:simpleType name="show">
 <xs:restriction base="xs:string">
  <xs:enumeration value="new"/>
  <xs:enumeration value="replace"/>
  <xs:enumeration value="embed"/>
  <xs:enumeration value="other"/>
  <xs:enumeration value="none"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="actuate">
 <xs:restriction base="xs:string">
  <xs:enumeration value="onLoad"/>
  <xs:enumeration value="onRequest"/>
  <xs:enumeration value="other"/>
  <xs:enumeration value="none"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="label">
 <xs:restriction base="string"/>
</xs:simpleType>
<xs:simpleType name="NullType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="inapplicable"/>
  <xs:enumeration value="unknown"/>
  <xs:enumeration value="unavailable"/>
  <xs:enumeration value="missing"/>
 </xs:restriction>
</xs:simpleType>


<xs:complexType name="simpleLink">
   <xs:sequence><xs:element name="type" type="xs:string" fixed="simple"/>
    <xs:element name="attr2" type="href" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attr3" type="role" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attr4" type="arcrole" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attr5" type="title" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attr6" type="show" minOccurs="0" maxOccurs="1"/>
    <xs:element name="attr7" type="actuate" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="boundedBy">
 <xs:choice><xs:element name="selector0" type="Box"/>
  <xs:element name="selector1" type="NullType"/></xs:choice>
</xs:complexType>
<xs:complexType name="Box">
   <xs:sequence><xs:element name="gid" type="ID" minOccurs="0" maxOccurs="1"/>
    <xs:element name="srsName" type="anyURI" minOccurs="0" maxOccurs="1"/>
    <xs:element name="b" type="coord" minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name="featureMember">
   <xs:sequence><xs:element name="feature" type="AbstractFeatureType"/>
    <xs:element name="geoproperties" type="GeometryProperty"
        minOccurs="0" maxOccurs="unbounded" /></xs:sequence>
</xs:complexType>
<xs:complexType name="GeometryProperty">
 <xs:choice>
  <xs:element name="selector0" type="PointPropertyType"/>
  <xs:element name="selector1" type="LineStringPropertyType"/>
  <xs:element name="selector2" type="PolygonPropertyType"/>
  <xs:element name="selector3" type="MultiPointPropertyType"/>
  <xs:element name="selector4" type="MultiLineStringPropertyType"/>
  <xs:element name="selector5" type="MultiPolygonPropertyType"/>
  <xs:element name="selector6" type="MultiGeometryPropertyType"/>
 </xs:choice>
</xs:complexType>
</xs:schema>
```

4 Resultados

Do estudo e do trabalho realizado advém um conjunto de resultados que se podem dividir em duas categorias. O primeiro resultado é de âmbito mais teórico e visava comprovar a capacidade expressiva do *p*GML, ao passo que o segundo conjunto de resultados permite ao engenheiro de *software* utilizar a *framework* construída, para desenvolver modelos específicos sobre o *p*GML da mesma forma que utilizaria uma outra metodologia que lhe permitisse cumprir as fases da estratégia de desenvolvimento de *software* GIS.

4.1 Resultados da obtenção do *p*GML

A obtenção do *p*GML em VDM-SL ganha valor por através dela ser possível cumprir a estratégia de desenvolvimento de *software* na qual a representação do modelo é capaz de ser derivada num documento GML, o que completa o ciclo:
Assim:

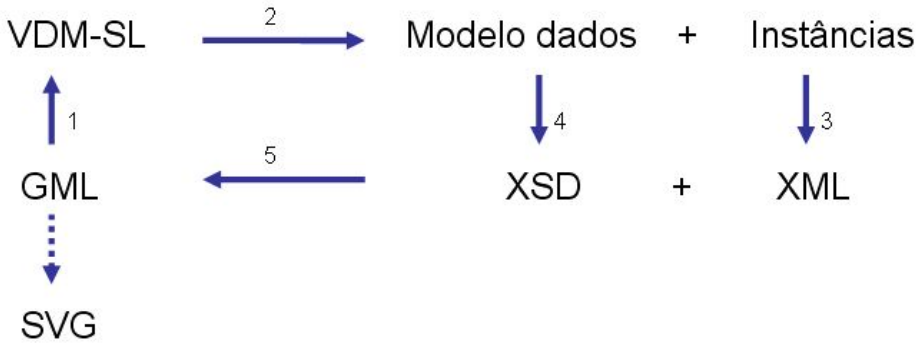


Figura 2. Ciclo: do GML ao GML

1. por engenharia reversa obteve-se um modelo preciso do GML em VDM-SL;
2. capaz de reter um modelo de dados e instâncias;
3. as instâncias são mapeadas em XML;
4. o modelo de dados permite definir o XSD;
5. o que nos retorna de novo ao GML.

Ganhou-se com isto:

- a transposição para o domínio dos Métodos Formais de um modelo de GIS;
- um mecanismo rápido de modelação e prototipagem;
- uma função de derivação da implementação a partir do modelo;
- um meio de comunicação com o cliente, utilizando por exemplo uma função de conversão de GML para SVG.

4.2 Utilização do *p*GML e da *framework* desenvolvida

O engenheiro de *software* passa a ter ao seu dispor um conjunto de facilidades que lhe permitem registar, sob forma de um modelo, os requisitos impostos para o seu Sistema de Informação Geográfica, capaz de ser animado (ie. utilizado como protótipo) de forma a obter uma correcta especificação dos requisitos. Deverá ainda utilizá-lo como *proof of concept* e meio de comunicação com o cliente. A obtenção do modelo final é obtida não de forma *ad hoc*, na qual vale ou não a experiência do engenheiro de *software*, mas por derivação passando da *abstract syntax* neste caso representada em VDM-SL para *concrete syntax*, neste caso o GML. Cada *função* de transformação (numa outra *concrete syntax*) corresponderá a uma nova implementação do mesmo modelo.

A utilização desta *framework* pode ser mais facilmente compreendida através de um pequeno *running example*:

> ”Uma instituição com várias unidades, projectos e edifícios pretende desenvolver um sistema de informação com os seus colaboradores, unidades, infra-estruturas e equipamentos, entre eles a própria localização física do posto de trabalho, contactos...”

O engenheiro de *software* irá utilizar a definição *p*GML da mesma forma que utilizaria o GML como base sobre a qual definiria o seu XSD específico para a sua representação aplicacional. Assim a definição do seu modelo deverá começar por importar todas as definições do *p*GML.

Sendo a sintaxe do VDM-SL bastante intuitiva apresenta-se de imediato o modelo de dados simplificado, ie. apenas com a indicação dos invariantes geométricos ou definidos nos requisitos.

```
module firststep
    imports
       from pgml all
    exports all
definitions
    EdificioType :: feature : pgml`AbstractFeatureCollectionType
                    Computadores : [pgml`MultiPointPropertyType]
                    Mobiliario : [pgml`MultiPolygonPropertyType]
                    Salas : SalaType-set
    inv mk-EdificioType (f, c, m, s) ≜
        EdiContainSala (s, f.bb) ∧
        EdiContainCompu (c, f.bb) ∧
        EdiContainMob (m, f.bb);
```

Onde se define que um edifício é um tipo de entidades que se traduz numa extensão a um *AbstractFeatureCollectionType*, extensão essa composta por três sub-entidades: *Computadores*, *Mobiliario* e um conjunto de *Salas*. Tem ainda um invariante associado, que irá garantir a conformidade geométrica deste tipo e das suas sub-entidades.

Por sua vez, *SalaType* define-se como:

```
    SalaType :: feature : pgml`AbstractFeatureType
                extentOf : pgml`PolygonPropertyType
                Pessoa : PessoaType-set
    inv sala ≜ card (sala.Pessoa) ≤ 10;
```

onde *SalaType* é uma entidade que estende *AbstractFeature*; é composta pela informação geométrica da sua área externa e da informação dos seus colaboradores. A título de exemplo, é imposto um invariante que limita a 10 o número máximo de colaboradores por sala.

Por fim, *PessoaType* é uma entidade com um atributo geométrico, a sua *location*, e herda, de acordo com o modelo, os atributos de *AbstractFeatureType* e tem ainda associada a informação da *idade* do colaborador:

$$
\begin{array}{l}
PessoaType :: feature : pgml`AbstractFeatureType \\
\qquad\qquad\quad location : pgml`PointPropertyType \\
\qquad\qquad\quad idade : \mathbb{N}
\end{array}
$$

A especificação do invariante de `EdificioType` é composta pela conjunção de três condições, que impõem a conformidade geométrica dos elementos contidos no edifício.

$$
\begin{array}{l}
EdiContainSala : SalaType\text{-set} \times pgml`Box \to \mathbb{B} \\
EdiContainSala\,(s, b) \triangleq \\
\quad (\forall\, sala \in s \cdot EdiContainSala2\,(sala.extentOf, b)); \\
EdiContainSala2 : pgml`Polygon \times pgml`Box \to \mathbb{B} \\
EdiContainSala2\,(pol, b) \triangleq \\
\quad (\forall\, crd \in \mathsf{elems}\ (pol.outerBoundaryIs) \cdot EdiContainSala3\,(crd, b)); \\
EdiContainSala3 : pgml`coord \times pgml`Box \to \mathbb{B} \\
EdiContainSala3\,(crd, b) \triangleq \\
\quad (crd.X \geq (\mathsf{hd}\ (b.b)).X) \wedge \\
\quad (crd.X \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).X) \wedge \\
\quad (crd.Y \geq (\mathsf{hd}\ (b.b)).Y) \wedge \\
\quad (crd.Y \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).Y) \wedge \\
\quad (crd.Z \geq (\mathsf{hd}\ (b.b)).Z) \wedge \\
\quad (crd.Z \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).Z); \\
EdiContainCompu : pgml`MultiPointPropertyType \times pgml`Box \to \mathbb{B} \\
EdiContainCompu\,(c, b) \triangleq \\
\quad (\forall\, e \in c.mp \cdot EdiContainCompu2\,(e, b)); \\
EdiContainCompu2 : pgml`Point \times pgml`Box \to \mathbb{B} \\
EdiContainCompu2\,(p, b) \triangleq \\
\quad (p.p.X \geq (\mathsf{hd}\ (b.b)).X) \wedge \\
\quad (p.p.X \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).X) \wedge \\
\quad (p.p.Y \geq (\mathsf{hd}\ (b.b)).Y) \wedge \\
\quad (p.p.Y \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).Y) \wedge \\
\quad (p.p.Z \geq (\mathsf{hd}\ (b.b)).Z) \wedge \\
\quad (p.p.Z \leq (\mathsf{hd}\ (\mathsf{tl}\ (b.b))).Z); \\
EdiContainMob : pgml`MultiPolygonPropertyType \times pgml`Box \to \mathbb{B} \\
EdiContainMob\,(m, b) \triangleq \\
\quad (\forall\, e \in m.mpol \cdot EdiContainMob2\,(e, b)); \\
EdiContainMob2 : pgml`Polygon \times pgml`Box \to \mathbb{B} \\
EdiContainMob2\,(pol, b) \triangleq \\
\quad EdiContainSala2\,(pol, b) \\
\mathsf{end}\ \textit{firststep}
\end{array}
$$

Construído o modelo o engenheiro de *software* pode utilizar o mecanismo de geração do XML e do XSD a partir do módulo VDM-SL[4][9][15] e com isso obter uma

implementação completa e automaticamente derivada a partir do modelo. A imagem seguinte ilustra uma instância XML de *EdificioType*.

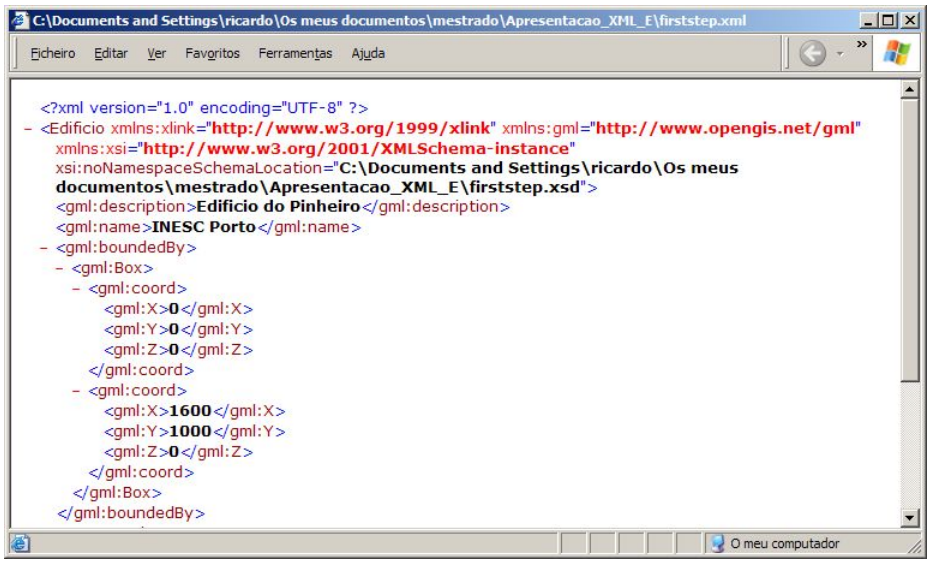


Figura 3. XML gerado a partir da *abstract syntax*

Através da ferramenta definida em [11] pode-se obter uma apresentação em SVG o que pode facilitar a comunicação com o cliente, como se ilustra na figura 4.

5 Conclusões e trabalho futuro

5.1 Conclusões

O GML, como modelo de armazenamento e como veículo de troca de informação geográfica é robusto e coerente. Estabelece um *standard* e um marco importante para a interoperabilidade que tanto em voga está. Contudo, como modelo de representação de um sistema de informação geográfica, não é ainda a mais valia que se procurava, no sentido em que apresenta algumas limitações, nomeadamente, por não introduzir estruturas algébricas de alto nível capazes de abstrair as entidades gráficas de baixo nível.

A sua forma de modelação *object-oriented* configura uma simplicidade e uma capacidade de derivação que, a ser estendida para outro nível poderá, então, garantir a conformidade semântica entre o elemento representado e a sua definição geométrica, e com isso a forma de como os elementos de representação do mundo real se articulam, para captarem, fielmente, as características e invariantes das entidades reais que representam. De facto o GML[2] deve ser encarado como uma plataforma genérica e

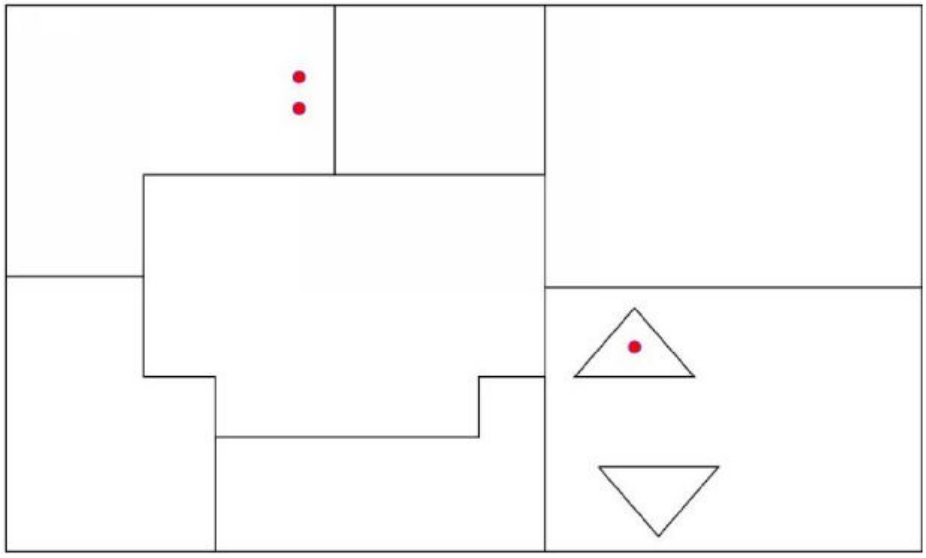


Figura 4. Exemplo do SVG obtido

rica que define apenas um patamar de abstracção para o armazenamento e partilha de informação geográfica.

O trabalho sucintamente descrito permitiu obter uma definição formal de um subconjunto do GML, traduzido por uma modelação sob o eixo dos dados. Uma especificação formal neste domínio tão vasto é apenas um primeiro passo e um exemplo sobre o qual se podem derivar novos modelos e implementações, mais ou menos ricas, consoante a necessidade e a capacidade de computação disponível. Conseguiu-se importar para o domínio dos Sistemas de Informação Geográfica o rigor, simplicidade, capacidade de raciocínio e prototipagem dos métodos formais e dualmente alargar o domínio dos métodos formais.

A capacidade de obter uma representação de um micro Sistema de Informação Geográfica, que respeite o GML, a partir de um modelo formal indica que é possível obter a mesma representação para qualquer implementação, bastando para isso mudar a função de derivação. Neste sentido é também possível derivar o *p*GML para paradigmas específicos utilizados em *softwares* comerciais.

5.2 Trabalho futuro

Na perspectiva histórica, sob o ponto de vista da interoperabilidade, ao olhar o XML, ou especificamente o GML, como uma espécie de denominador comum para a intercomunicação de diversos sistemas não compatíveis e com formatos proprietários, é compreensível que se tenha deixado o ónus da conformidade geométrica recair sobre

esses sistemas. Contudo a riqueza semântica do VDM-SL, no que respeita aos invariantes geométricos (geográficos), necessita também de ser expressa. De acordo com [8] o XSCL (*XML Constraint Specification Language*) é o candidato *natural* a representar os invariantes semânticos (que resultam da necessidade de representação de elementos reais, os quais obedecem a constrangimentos reais[10]), sobre outras abordagens como o *Schematron* e *XML-Schema*. O XCSL é ainda ele próprio um formato XML[7].

Referências

[1] J. Backus. Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. *CACM*, 21(8):613–639, August 1978.

[2] Open Gis Consortium. Gml implementation specification. Technical Report 02-023r4.

[3] Open GIS Consortium. *OpenGIS Geography Markup Language Implementation Specification*. OGC, 2002.

[4] Mário Ricardo de Novais Henriques. pgml-estudo de um subconjunto preciso do gml2.12. Technical report, Univ. Minho, 2004.

[5] The VDM Tool Group. Vdmtools - the ifad vdm-sl language. Technical report, IFAD, Forskerparken 10, DK-5230 Odense M, Denmark, 2000.

[6] Kal Ahmed; Sudhir Ancha; Andrei Cioroianu; Jay Cousins; JereMy Crosbie; John Davies; Kyle Gabhart; Steve Gould; Ramnivas Laddad; Sing Li; Brendan Macmillan; Daniel Rivers-Moore; Judy Skubal; Karli Watson; Scott Williams; James Hart. *Professional Java XML*. WROX Press Inc., 2001.

[7] Marta Jacinto; Giovani Librelotto; José Ramalho; Pedro Henriques. Xcsl tutorial. Technical report, Universidade do Minho, 2002.

[8] Marta Jacinto; Giovani Librelotto; José Ramalho; Pedro Henriques. Xcsl: Xml constraint specification language. Technical report, Universidade do Minho, 2003.

[9] IFAD. Vdmtools - the dynamic link facility. Technical report, IFAD, Forskerparken 10, DK-5230 Odense M, Denmark, 2000.

[10] Martien Molenaar. *An Introduction to the Theory of Spatial Object Modelling for GIS*. Taylor and Francis, 1998.

[11] Anabela Soares Pinto Monteiro. Servidor de mapas vectoriais. relatório do projecto. Technical report, 2002.

[12] J. N. Oliveira. CAD Tool Extension for Formal Building Description Language . *Advances in Engineering Software http://www.elsevier.nl/inca/publications/store/4/2/2/9/1/1/index.htt*, 29(7-9):571–586, 1998.

[13] Pedro Rangel Henriques; José Carlos Ramalho. *XML & XSD - da teoria à prática*. FCA, 2002.

[14] W3C. W3c architecture domain, xml schema. (http://www.w3c.org/XML/Schema).

[15] Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc, 1992.

Suporte à Elaboração e Manutenção de PMOT usando tecnologia XML

José Lino Oliveira

INESC Porto, R. Dr. Roberto Frias, 4200-465 Porto
http://www.inescporto.pt/
jose.lino@inescporto.pt

Resumo Os *PMOT* (Planos Municipais de Ordenamento do Território) são instrumentos de planeamento territorial de natureza regulamentar, que traduzem as acções a desenvolver pela Administração Local, visando assegurar uma adequada organização e gestão do território municipal. Sendo os *PMOT* Instrumentos de Gestão Territorial (IGT), cuja competência está atribuída às Autarquias, estas são as responsáveis pela sua elaboração, manutenção e divulgação.
Este artigo apresenta uma ferramenta baseada em XML que se destina a satisfazer as necessidades concretas dos técnicos que actuam nas actividades de gestão e planeamento urbanístico, designadamente no suporte à disponibilização e divulgação de conteúdos de um *PMOT*, peças gráficas e regulamento.
A aplicação foi desenvolvida usando a tecnologia VB.NET da *Microsoft[8]* e Mapguide da *Autodesk[2]* tendo como base ficheiros XML. A criação e edição de um regulamento é guiada pela aplicação que armazena o seu conteúdo, permitindo ainda a associação com as peças gráficas na construção e edição da informação do Plano. O resultado é um conjunto de ficheiros em dialectos XML que são usados pela plataforma de disseminação.
A utilização desta aplicação contribuiu, por um lado, para acelerar e simplificar o processo de disponibilização online de *PMOT*, e por outro para fomentar a utilização de uma estrutura reutilizável nas diferentes fases da sua elaboração.

1 Introdução

A Política de Ordenamento do Território e de Urbanismo define e integra as acções promovidas pela Administração Pública[1], visando assegurar uma adequada organização e utilização do território, num quadro de interacção coordenada, em três âmbitos distintos: o nacional, o regional e o municipal.

Os Instrumentos de Gestão Territorial, de acordo com as funções diferenciadas que desempenham, integram vários instrumentos de planeamento e desenvolvimento territorial de natureza estratégica, regulamentar e de política sectorial. Estes instrumentos têm o seu ciclo de vida e passam por diversas fases no seu processo: fase de acompanhamento da elaboração, que é feita por uma comissão mista de coordenação; fase de concertação em que estão envolvidos

vários técnicos com especificidades diferentes onde são analisados, avaliados e discutidos os vários cenários, com base na informação existente; fase de participação e discussão pública; fase de aprovação do plano. Este processo tem actualmente muito pouco suporte digital, mas um dos resultados finais é a sua disponibilização online.

Neste sentido, surge a necessidade de ferramentas de apoio que permitam reunir e processar informação, elaborar planos, construir modelos de avaliação de planos, confrontar cenários alternativos, dar fundamentação aos processos de tomada de decisão e publicação online.

O objectivo deste trabalho consiste na criação de uma estrutura de apoio à elaboração e manutenção de *PMOT* que reúna a informação gráfica e regulamentar, que os constituem, permitindo a sua gestão e rápida disseminação. A acção centra-se essencialmente nas fases de discussão pública e disponibilização online de uma forma transparente e automatizada, contribuindo para preencher uma lacuna que existe nos Sistemas de Informação Geográfica Municipais ao nível do Planeamento Urbanístico.

No passado recente o projecto SIMAT [10], teve no seu contexto a definição de especificações para aplicações técnicas de âmbito municipal suportadas em SIG[1]. No entanto dada a diversidade deste domínio de aplicações, a área de planeamento urbanístico teve apenas uma abordagem inicial.

As tecnologias associadas ao *Extensible Markup Language(XML)[14]* aparecem como o meio ideal para servir de base a esta ferramenta. Por excelência são ideais para representação de informação semi-estruturada e fornecem uma plataforma integradora que aqui se justifica na medida em que o trabalho está orientado para dar apoio às fases finais do processo nomeadamente a disponibilização online. Como existe ainda um vazio no que diz respeito ao suporte prestado nas fases iniciais, foi necessário criar um estrutura que permitisse uma futura integração. Houve ainda a necessidade de criar automatismos necessários à disponibilização online e que utilizam tecnologias proprietárias.

O resto do artigo está estruturado da seguinte forma: a secção 2 apresenta uma contextualização do problema a tratar bem como a arquitectura seguida; a secção 3 faz uma apresentação de alguns detalhes da implementação assim como uma avaliação de resultados alcançados; na secção 4 são apresentadas algumas conclusões que se podem retirar e feitas algumas considerações acerca de trabalho futuro.

2 Descrição do Problema e Requisitos

A elaboração de planos como instrumentos de planeamento urbanístico, de que são exemplo os *PMOT* é uma tarefa complexa que envolve muitas fases e também grandes volumes de informação de natureza distinta, quase sempre oriundas de diversas fontes: peças gráficas que correspondem à área de incidência; peças escritas que correspondem à informação regulamentar da área geográfica definida pelas peças gráficas.

[1] Sistema de Informação Geográfica

A principal motivação deste projecto foi desenvolver uma ferramenta que desse suporte à elaboração e manutenção de *PMOT* integrada numa estrutura que permitisse a sua rápida disponibilização e disseminação nas diferentes fases do processo, conjugando os diversos tipos de informação. Uma das faces visíveis seria a disponibilização imediata online do Plano.

Inicialmente deparou-se com o problema de termos uma tecnologia proprietária que iria ser usada para disponibilizar online os *PMOT*, e por outro lado as entradas da aplicação serem oriundos de diferentes fontes e existirem em diferentes formatos.

Pretendia-se que a aplicação conseguisse no futuro articular-se com as diferentes fases de elaboração do processo, visto ela dar suporte imediato apenas às fases de participação e disponibilização online, prevendo-se num futuro próximo o aparecimento de outras aplicações que suportem as fases iniciais.

A solução passou pela utilização das tecnologias baseadas em *Extensible Markup Language (XML)*. Foi criada uma estrutura que suportasse o regulamento afecto a cada *PMOT* e para isso foi criado um dialecto XML[3] próprio. Por outro lado foi criado um outro dialecto que suportaria a articulação entre o regulamento e as peças gráficas cuja fonte seria informação contida em ficheiros SDF[2]. Recorreu-se a algumas implementações com XSLT[13] para efectuar as transformações da informação nomeadamente ao que respeita a construção da informação geográfica a ser disponibilizada. No que diz respeito à disponibilização online, foi construído um módulo de visualização de planos que utiliza a informação gerada e a reproduz. A arquitectura do sistema é apresentada na figura 1.

A totalidade do regulamento do Plano é gerido e armazenado num documento XML, que irá ser usado posteriormente na fase de interligação com as peças gráficas. O regulamento pode ser exportado para formato HTML e DOC (*Microsoft Word*) recorrendo a folhas de transformação XSLT. Após o carregamento dos documentos contendo as peças gráficas, estamos em condições de podermos definir o Plano, que irá ser construído usando o documento do regulamento e as peças gráficas. No final deste processo temos então um novo documento XML que conterá a informação a constar do Plano. Seguidamente e recorrendo a uma folha de transformação XSLT, a informação é transformada para MWX[3] com os estilos, construindo um mapa. Através de um visualizador de planos, o mapa é disponibilizado online, bem como a informação regulamentar.

3 Implementação e Avaliação do Projecto

A aplicação foi desenvolvida em VB.NET e tem por base a composição de dois dialectos XML[4] criados para o tratamento específico do regulamento e para a estrutura da informação do Plano: *pmx.xsd* e *pcx.xsd*(ver a figura 2). A informação gráfica é carregada em ficheiros SDF e os estilos gráficos a aplicar são definidos num outro dialecto criado para o efeito, *styles.xsd*. As transformações

[2] *Spatial Data Files - Ficheiros gráficos nativos do Autodesk Mapguide*
[3] Dialecto XML do *Autodesk Mapguide*

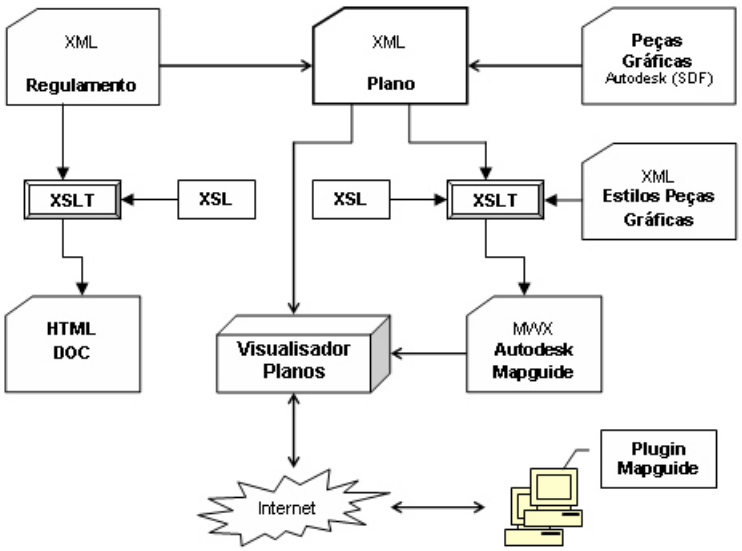


Figura 1. Arquitectura do Sistema

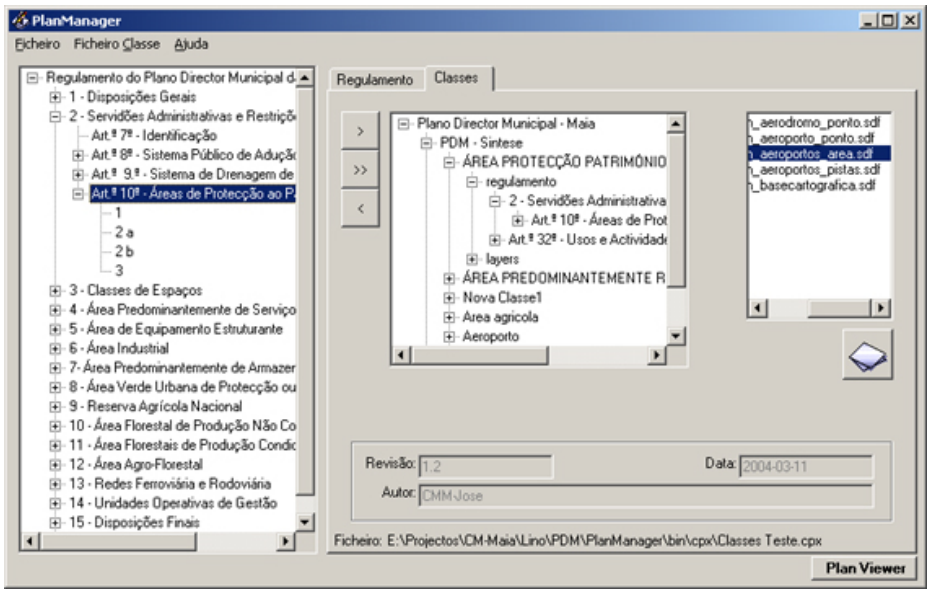


Figura 2. Aplicação do Plano

quer do regulamento para HTML e DOC, quer da informação do Plano para MWX recorrem a folhas de transformação XSLT respectivamente *pmx2html.xslt*, *pmx2doc.xslt* e *cpx2mwx.xslt*. A disponibilização online é feita utilizando um módulo de visualização de planos desenvolvido em ASP.NET que disponibiliza o mapa recorrendo ao *Autodesk Mapguide*. A visualização do regulamento é feita em HTML após ser aplicada uma folha de transformação XSLT ao XML que contêm a informação do Plano (ver a figura 3).

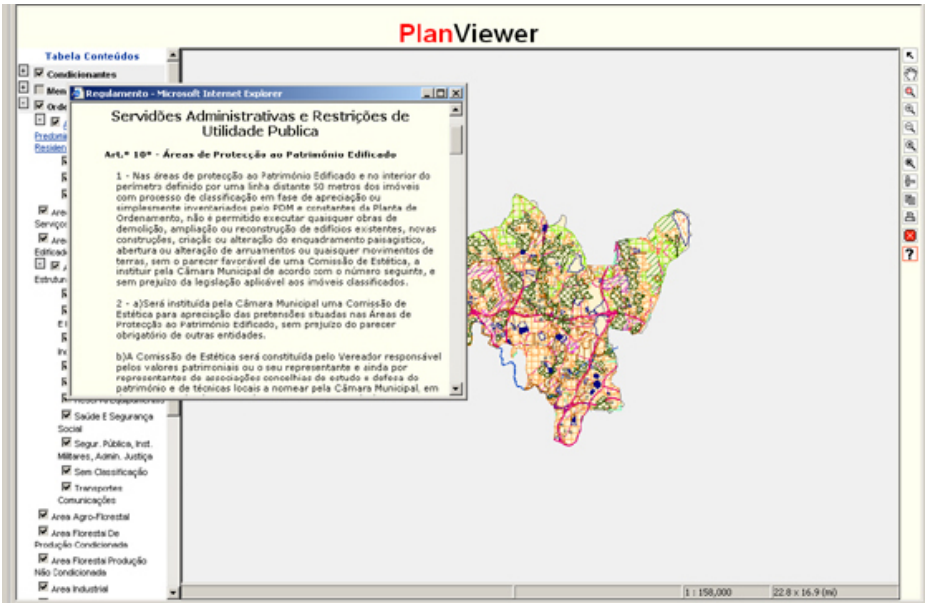


Figura 3. Visualizador de Planos

Dialectos XML desenvolvidos

No regulamento do Plano foi criada uma estrutura que conseguisse de uma forma não limitativa e semi-estruturada, albergar o seu conteúdo. Para tal foi criado um elemento genérico: *entrada* que contém dois atributos, o nome e tipo em que o primeiro será um título e o segundo a definição da sua classe (Capítulo, Secção, Artigo, etc). Deste elemento fazem ainda parte outros dois elementos *titulo* e *descricao* que irão conter o conteúdo do regulamento propriamente dito. Como um dos requisitos era não existir inicialmente restrições na construção do documento em termos de estrutura e apenas em termos de conteúdo, um elemento *entrada* pode estar contido dentro de outro elemento *entrada* (ver a figura 4).

O documento que contém a informação do Plano, seguiu as regras que regulam a sua constituição. Assim, existem elementos *planta* que dizem respeito

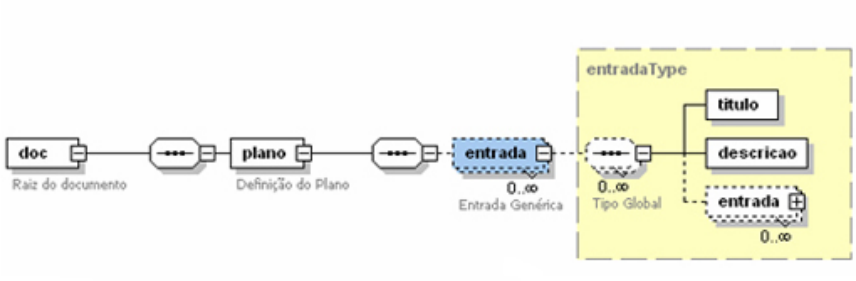


Figura 4. Estrutura do documento do regulamento

às diversas plantas que podem fazer parte do Plano. Estas podem conter vários elementos *classe* que estipulam as directivas e que são compostos por um elemento *regulamento* e por um elemento *layers*. O primeiro reúne a informação regulamentar e o segundo a informação gráfica (ver a figura 5).

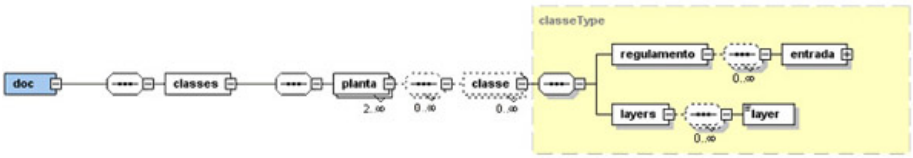


Figura 5. Estrutura da informação do plano

Implementação com XSLT

Foram implementadas algumas transformações nomeadamente a partir dos dialectos XML desenvolvidos, no caso do regulamento do plano, para HTML e DOC e no caso da informação do Plano para MWX de modo a esta poder ser posteriormente visualizada online. A folha de transformação para MWX foi desenvolvida obedecendo ao *schema* disponibilizado pela *Autodesk*.

As transformações foram executadas usando as classes disponibilizadas pela plataforma .NET

Avaliação dos Resultados

A primeira avaliação feita à aplicação, foi bastante positiva, visto que em termos da sua aplicabilidade na resolução do problema para que foi desenvolvida atingiu os objectivos a que se tinha proposto. Consegue-se implementar uma

estrutura que permite resolver um problema concreto e para o qual não havia grandes alternativas. A escolha das tecnologias associadas ao *Extensible Markup Language(XML)* vieram a provar-se acertadas, pois permitiram uma grande margem de manobra no que diz respeito às integrações que eram necessárias fazer nomeadamente com o Autodesk Mapguide.

A aplicação foi testada em ambiente real de utilização na disponibilização online do PDM[4] por parte de uma Câmara Municipal. Através desta forma expedita de colocar o Plano online, é possível não só agilizar a sua disponibilização para o público, mas também internamente (para a própria equipa do plano), fornecendo um meio para melhorar a comunicação interna durante o seu processo de preparação.

4 Conclusões e Trabalho Futuro

O objectivo principal deste trabalho era a criação de uma aplicação de apoio à elaboração e manutenção de *PMOT* e que desse suporte às fases finais do processo, nomeadamente a fase de discussão pública e disponibilização online.

É proposta uma arquitectura baseada em XML que permite integrar com tecnologias proprietárias e deixar em aberto a integração com outras ferramentas que possam vir a aparecer, e que se prevê que surjam visto não existir até ao momento nada que dê suporte às fases iniciais do processo de elaboração. Os resultados alcançados deixam antever a utilidade e ganho na produtividade que a utilização de uma ferramenta deste tipo fornece.

Ao ser criada uma ferramenta com uma estrutura de apoio flexível e escalável, foi feita uma tentativa de implementação de regras no processo de elaboração de *PMOT*, por forma a que este tirasse partido de alguns automatismos que fossem criados. Essa tentativa veio-se a provar uma boa solução, pois permitiu no imediato a disponibilização online dos mesmos.

A inovação que esta ferramenta trouxe foi ao nível da conjugação de tecnologias existentes e da colocação destas ao serviço das pessoas numa área muito pouco informatizada.

No que respeita a esta ferramenta existem ainda vários melhoramentos e funcionalidades que se podem desenvolver: na estruturação do regulamento podia-se implementar uma importação de um documento *Microsoft Word* directamente para a estrutura; no tratamento das peças gráficas, podia-se pensar em implementar um visualizador gráfico; no que respeita à construção e manutenção da informação do plano podia-se implementar um mecanismo de validação sobre o regulamento e sobre as peças gráficas no sentido de saber se todo o regulamento e todas as peças gráficas estão referidas no documento.

No que diz respeito ao suporte de elaboração de *PMOTS* ainda há muito para se fazer, nomeadamente no apoio às fases iniciais. A construção de uma ferramenta colaborativa que permitisse dar suporte à fase de concertação, e que

[4] *Plano Director Municipal, Plano de Urbanização e Plano de Pormenor são exemplos de PMOT*

reunisse toda a informação de uma forma centralizada, seria uma mais valia acrescida no processo de tomada de decisão e de fluidez da informação.

Referências

[1] Decreto-Lei 380/99 de 22 de Setembro com redacção conferida pelo Decreto-Lei nº310/2003 de 10 de Dezembro In: Diário da Républica I Série-A nr.222, Imprensa Nacional-Casa da Moeda, 1999.
[2] AUTODESK. Autodesk Worldwide, 2004. `http://www.autodesk.com`.
[3] Neil Bradley. *The XML Companion*. Addison-Wesley, 2002.
[4] José Carlos Ramalho e Pedro Henriques. *XML e XSL*. FCA.
[5] Object Management Group. XMI: XML Metadata Interchange. `http://www.omg.org`.
[6] Artur Rocha e João Correia Lopes Jorge Cardoso. M-GIS - Sistema Móvel Interoperável de Informação Geográfica, 2004. `http://www.inescporto.pt`.
[7] Luís Bártolo Marco Amaro Oliveira, Alexandre Carvalho. Public Discussion of Oporto's Municipal Master Plan: An e-Democracy Service Supported by a Geographical Information System, 2004. INESC-Porto, Rua Dr. Roberto Frias, Nº 378 4200-465 PORTO, Portugal `http://www.inescporto.pt`.
[8] MICROSOFT. Microsoft Corporation, 2004. `http://www.microsoft.com`.
[9] OGC. Consórcio Open GIS, 2004. `http://www.opengis.org`.
[10] Consórcio SIMAT. SIMAT: relatório de especificação de requisitos `http://simat.inescporto.pt`. Relatório, INESC-Porto, Rua Dr. Roberto Frias, Nº 378 4200-465 PORTO, Portugal `http://www.inescporto.pt`, 1998.
[11] J. Sperberg-McQueen T. Bray, C.M. Paoli and E. Maler. Extinsible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, 2000. `http://www.w3.org/TR/2000/REC-xml-20001006`.
[12] W3C. XML Schema. `http://www.w3.org/XML/Schema`.
[13] W3C. XSL Transformations, 1999. `http://www.w3.org/TR/xslt`.
[14] W3C. Extensible Markup Language (XML), 2003. `http://www.w3.org/XML/`.

Curriculum@UA - online xml based personal curriculum

C. Teixeira, J. S. Pinto, J. Santos

IEETA – Instituto de Engenharia Electrónica e Telemática de Aveiro
Universidade de Aveiro, Portugal
{claudio, jsp, johnny }@ieeta.pt

Abstract. This paper describes a project in development at Universidade de Aveiro concerning online filling and central storage of researchers' curriculum vitae (CV). This system enables anyone to have an up to date CV always available for use anywhere. The major features in this site are its use of XML as the storage format for each individual CV and PDF CV generation.

1. Introduction

Giving its flexible structure, XML [1] has been used not only as a standard for data interchange, but also for data storage. XML databases are being studied [2] in order to understand its advantages and limitations when matched against relational database systems. The choice to use XML technology instead of relational database is related with the need to present the same information in several ways. XML seems to be a more flexible approach to deal with very wide - but small - information sets. Each CV may have several sets of information, but each person will not have many elements in each set, and some of them may even be empty.

Online CVs are a good idea, but if the CV is only for use inside the institution where it was filled, with no possibility of being used in other organizations, then those users tend to insert their CV only when they need it, and after that, there is no updating of the their information (why would they?). We propose an online CV system whose goal is the standardization of CVs at Universidade de Aveiro, but that users can use to present their CV anywhere. With this system, users are invited to maintain an updated CV, since they may also benefit from this.

This paper describes an online tool that uses XML to store CV's according to international standards described later on this paper. The remainder of this paper is organized as follows: Section 2 presents related work, Section 3 presents the data model used in the project and Section 4 explains the overall architecture. Finally, experimental results, conclusions and future work are presented in Section 5.

2. Related Work

The majority of online services that allow users to fill their CVs are paid services. [3] lists several examples of such services available in the UK.

There are some research organizations and public institutions that also use online CV's: two of these are the Brazilian *Sistema CV-Lattes* [4] and the Portuguese *Fundação para a Ciência e Tecnologia* (FCT) [6].).

CV-Lattes was developed by CNPq [5] in 1999, and is used by several Brazilian institutions and organizations, including the Brazilian Ministry of Science and Technology. In order to scholars and researchers to apply to appropriate funding, they are obliged to have an updated CV in this system. With this policy since 2002, the CV-Lattes has grown to over 460.000 CVs.

Portuguese FCT [6] promotes national scientific research and technological development by sponsoring scholarships, projects and scientific research institutions. Again, in order to apply to these funding, online CVs must be filled into the system by researchers and their coordinators. There is no public information about the amount of CVs that are in the system at this moment, but as in CNPq, they are very important in the evaluation phase of funding applications.

3. Architecture

Fig. 1 represents the functional layers present in Curriculum@UA. The top level is where the user interaction takes place.

At the bottom of the system there are three store systems: XML, XSL-FO [8] and PDF [7]. Each one is responsible for storing a specific set of files: "XML Store" keeps the XML CVs, "XSL-FO Store" keeps the transformation rules for presenting a PDF of the CV and finally, "PDF Store" keeps all the generated PDFs.

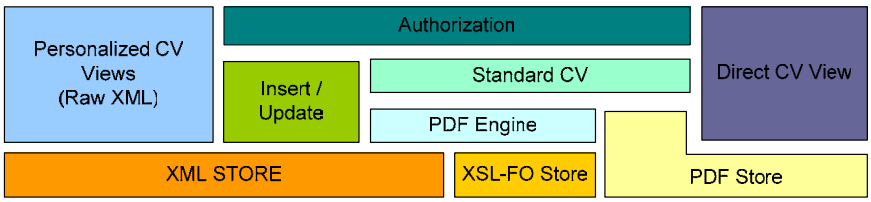


Fig. 1. Application Functional Layers

The "Authorization" module is responsible for validating the user and for granting access to the "Insert / Update" and "Standard CV (generator)" modules. "Insert / Update" module interacts directly with the "XML Store" to perform the desired operations. The "Standard CV" module creates the PDF CV at the "PDF Engine" and stores it in the "PDF Store" module. "PDF Engine" selects the XML CV and the XSL-FO to use to transform the XML.

As shown in Fig. 1, there are three main user interaction modules with the system: "Authorization" (already described), "Personalized CV Views" and "Direct CV

View". "Direct CV View" grants direct access to the "PDF store" (by using a specific URL), allowing universal access to the stored CVs. This feature can save paper and unnecessary printouts of CVs, if the users can give an URL (for a given application, job interview …) instead of a paper copy. The "Personalized CV View" allows everyone to access to the XML CV file. Since this application is built using public XSDs [12], anyone can use the updated XML to produce their own visual presentation for a given CV. This feature can be used by expert users to link a customized presentation of their always updated CV in their home pages. Organizations using HR-XML Resume schema can also use this to look at these CVs as if they were looking at "*house produced*" CVs.

Fig. 2, Fig. 3 and Fig. 4 represent different operations that can be performed in the system and the workflow involved.

Fig. 2 represents a user that needs his CV to file for application. The process starts whit the user's request of a PDF version of his CV to the system's Web Server. The system fetches it from the "PDF Store" and sends back both file and URL to the user. At this point, the user can choose between two scenarios depending on the application bureaucracy: either he prints out a copy and delivers it, or he simply delivers the URL for the services to see.

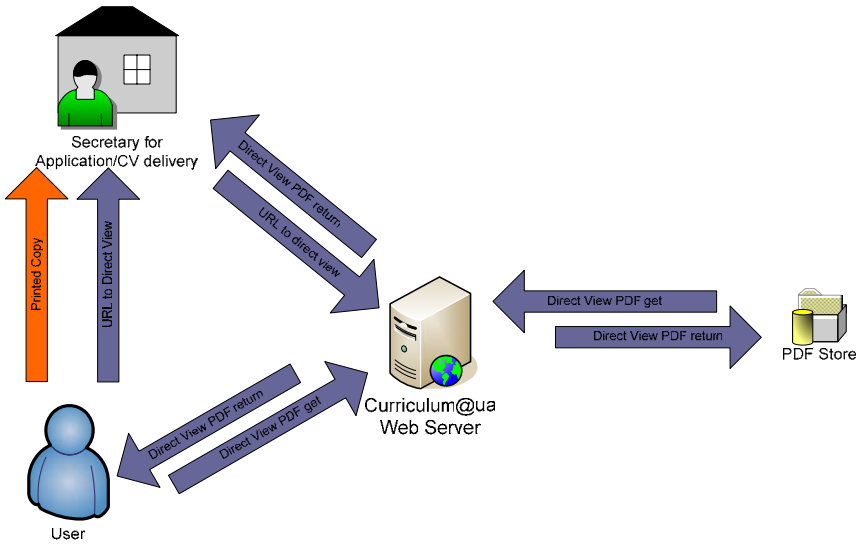


Fig. 2. Message flow for a PDF version of a CV to be shown

Fig. 3 represents the internal processing needed to complete an order of creating a PDF version of the user's CV. The messages flow as follows: the user requests the PDF generation; then the Web Server verifies the user's authorization on the "Authentication Server". If the user is allowed to generate the CV, then the Web Server requires from the "XSL-FO PDF Transformer" (XSL-FOT) the generation of the PDF file. The XSL-FOT gathers the required information - the CV in XML from the "XML Store" and the XSL-FO rules' file from the "XSL-FO Store" - and generates

the PDF CV and stores it in the PDF Store. Finally, the PDF Store sends both URL and PDF to the Web Server as referred before. For the sake of completeness, there must be made two remarks about this process: first, the PDF generation is a very expensive operation for the CPU. Therefore, it is only carried out at the user's request and not by a Direct View request. Second remark: as mentioned before, the goal of this system is standardization in information storage and presentation. This may seem inconsistent with the "XSL-FO Store" available, however, at the moment this Store has one single presentation format for CVs.

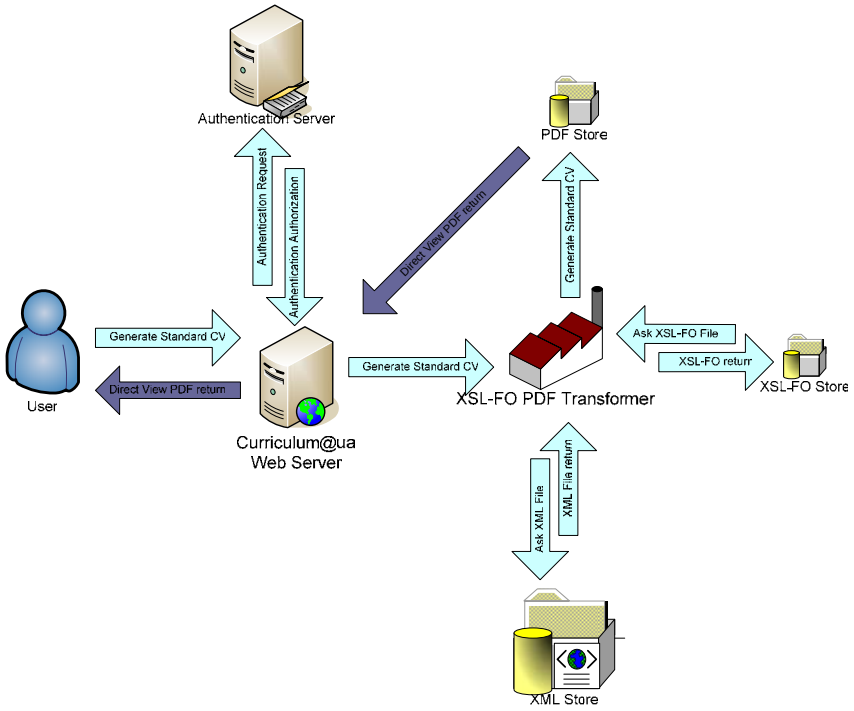


Fig. 3. Message flow to generate a PDF version of a CV

Fig. 4 represents an "Insert/Update" operation. The operation starts with the user requesting the operation to the "Web Server". Again, the "Web Server" verifies the user's authorization on the "Authentication Server". If the user is allowed to perform such operation, then the "Web Server" requests the XML CV from the "XML Store", makes the necessary changes and finally returns the updated XML CV to the Store.

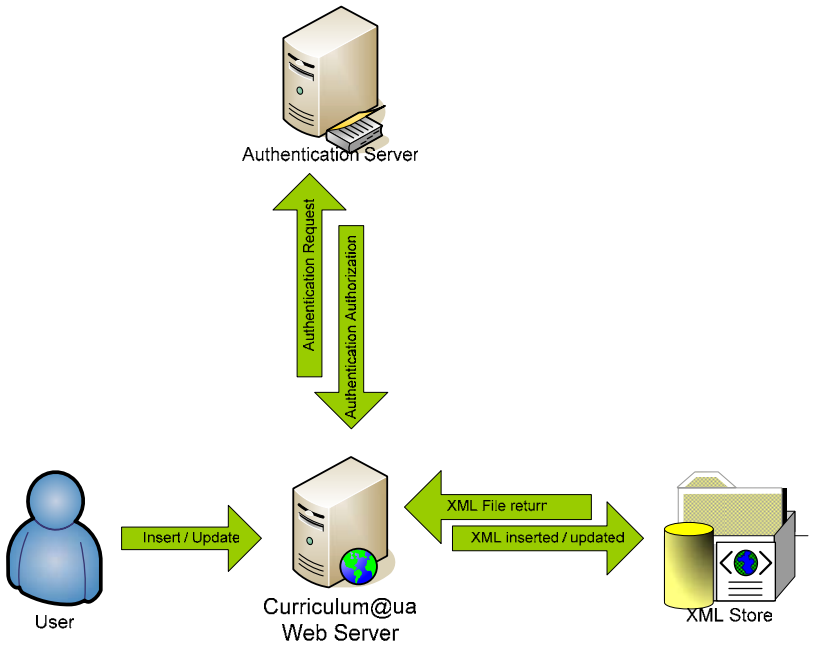


Fig. 4. Message flow for an insert/update to a given CV

4. Data Model

One of the major tasks on this project was the definition of which CV model to use. At the moment, there is no standard that states which information must or must not be placed in a CV or even in which place in the CV. After the analysis of three CV models (European [11], CNPq and HR-XML), the choice felt on the HR-XML Resume. It's a well documented model (that modelled more than enough information for our purposes) and is supported by a non governmental, international organization. Fig. 5 shows the XML-HR Resume schema adopted.

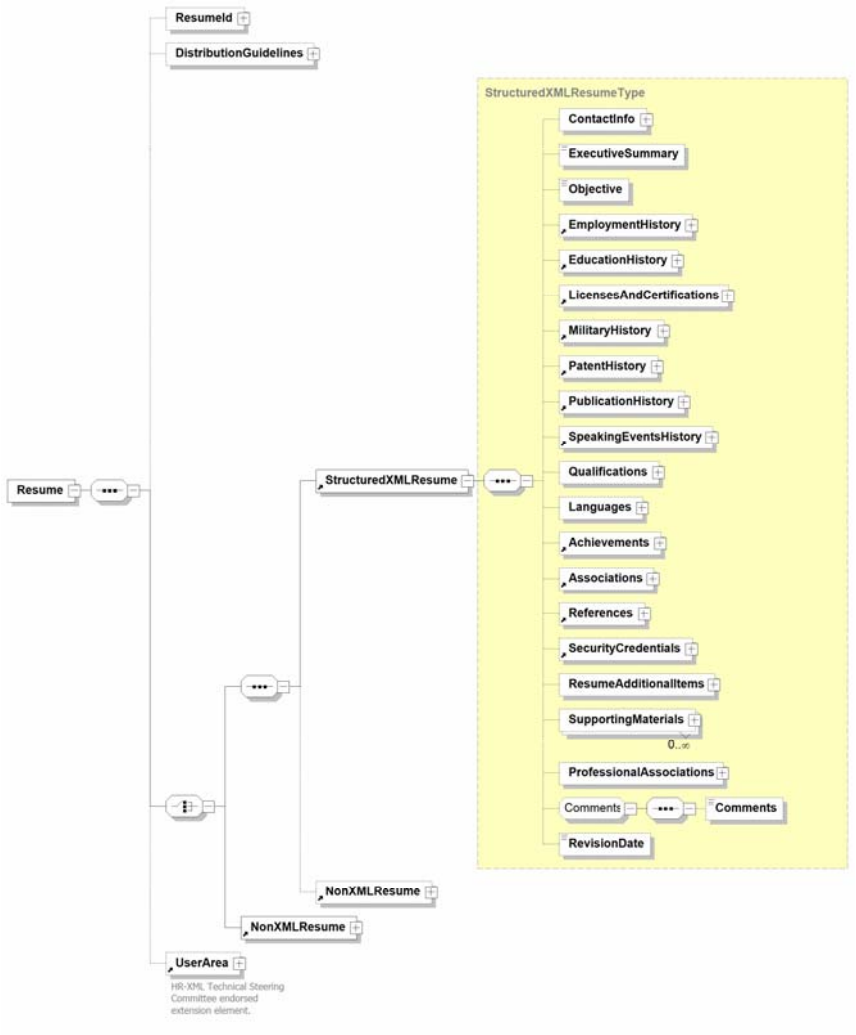


Fig. 5. Full XSD for Resume, as specified by HR-XML

As shown, the Resume is a very comprehensive list of structured information. For logistic reasons, not all this schema has been implemented. Only the subset that seemed to be fundamental for researchers and scholarship applications was implemented. This includes Contact Info, Executive Summary, Objective, Employment History, Licenses and Certifications, Patent History and Publication History.

5. Conclusions

As mentioned, this is still an on going project, but has already been used for PhD scholarship application held at Universidade de Aveiro during last summer. As not all of the possible candidates could understand Portuguese, the site developed is multilingual. XML technology was also used to implement such feature. This application also required two more features: an online work plan and a way to finalize each user's process. These features were developed according to the system's architecture and are also XML based.

There were about fifty candidates applying for scholarships and they had to register, fill in their CV and work plans in the system, print them, finalize the process and delivery the documents in the academic services in charge of the scholarship application.

An online management interface that enabled administrators to monitor the documents in the system was also developed. After the application period expired, the administrators could email a selected collection of PDFs to designated juries to evaluate the candidates.

According to the people in charge of the scholarship applications, this project has saved time in the evaluation phase, since the document layout was always the same, and every subject was exactly in the same place for all the candidates, enabling a quicker cross-checking analysis and evaluation. Therefore, the standardization has proven to be profitable.

From the user's point of view, he can continue to keep an updated record of his CV on the site, in order to use it whenever and wherever he wants to.

The results achieved by the scholarship application have encouraged the development team to increase the supported subset of information available.

With the increase of interoperability opportunities among applications developed at Universidade de Aveiro, this project will focus some attention in the development of interfaces to enable the automatic update of bibliographic references by existing specialized services, in a seamlessly way for the user.

As the HR-XML has a certification site and logo [10], tests are being made in order to make this site eligible for the certification logo.

References

[1] Bray, T., et al (editors), "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation 6 October 2000, W3C, online, last accessed in 10th November 2004, http://www.w3.org/TR/2000/REC-xml-20001006

[2] K. M. Win, W. K. Ng, and E. P. Lim. An Architectural Framework for Native XML Data Management. 2nd International Conference on Cyberworlds (CW 2003), 3-5 December 2003, Singapore. IEEE Computer Society 2003, 302-209 pp, ISBN 0-7695-1922-9

[3] "All About CVs", Regard-It, online, last accessed 16th November 2004, http://www.regard-it.com/cv.htm

[4] "CNPq – Plataforma Lattes", Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brazil, online, last accessed in 16th November 2004, http://lattes.cnpq.br/curriculo/

[5] "CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico", Conselho Nacional de Desenvolvimento Científico e Tecnológico, online, last accessed in 16th November 2004, http://www.cnpq.br/english/aboutcnpq/index.htm

[6] "Home Page FCT", Fundação para a Ciência e Tecnologia - Ministério da Ciência e Ensino Superior, online, last accessed in 16th November 2004, http://www.fct.mces.pt/

[7] "Adobe PDF: Secure, reliable electronic document distribution and exchange", Adobe Systems Incorporated, online, last accessed in 17th November 2004, http://www.adobe.com/products/acrobat/adobepdf.html

[8] "Formatting Objects", In "Extensible Stylesheet Language (XSL) – Version 1.0 – W3C Recommendation 15 October 2001", W3C, online, last accessed in 17th November 2004, http://www.w3.org/TR/xsl/slice6.html#fo-section

[9] "hr-xml.org -- Home of the HR-XML Consortium, Inc.", HR-XML Consortium Inc., online, last accessed in 28th October 2004, http://www.hr-xml.org/channels/home.htm

[10] "HRcertify.org: A Service of the HR-XML Consortium, Inc.", HR-XML Consortium Inc, online, last accessed in 17th November 2004, http://www.hrcertify.org/

[11] "CEDEFOP: The European Centre for the Development of Vocational Training - Cedefop in Brief", CEDEFOP, online, last accessed in 17th November 2004, http://www.cedefop.eu.int/transparency/cv.asp

[12] Priscilla Walmsley, "Definitive XML Schema", Prentice Hall PTR, 07 December, 2001. 560 pages. ISBN: 0130655678

Representação em XML da Floresta Sintáctica

Rui Vilela[1], Alberto Simões[1,2], Eckhard Bick[3], and José João Almeida[2]

[1]Linguateca, pólo de Braga
{ruivilela|ambs}@di.uminho.pt

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

[3]Institute of Language and Communication, University of Southern Denmark
eckhard.bick@mail.dk

Resumo A Floresta Sintáctica é um recurso linguístico mantido e distribuído livremente ao público pela Linguateca. Face à necessidade de abranger uma maior comunidade de linguistas e de programadores na área do processamento da linguagem natural, procura-se distribuir este recurso em outros formatos alem do existente, tais como formatos baseados em XML.
Pretende-se neste documento descrever o processo de equipar a Floresta Sintáctica com mecanismos que facilitem a sua utilização. Para isso pretende-se converter o seu formato actual para formatos baseados em XML. Serão analisados os problemas e questões da definição XML deste tipo de recurso.
Apresenta-se ainda um módulo experimental construído especificamente para facilitar a construção de processadores da Floresta.

1 Introdução

Pretende-se representar em XML uma estrutura organizada de textos anotados sintacticamente em forma de árvore, referenciada como *treebank*. De uma forma simplicista, um *treebank* pode ser comparado a um conjunto de árvores sintácticas de diferentes frases.

A necessidade de um recurso semelhante a um*treebank* na área do processamento computacional da língua portuguesa, que contenha uma base de dados estruturada de textos, previamente analisados e classificados gramaticalmente, levou ao surgimento da Floresta Sintáctica [1], um recurso disponibilizado pela Linguateca [13].

1.1 A Linguateca

A Linguateca[1] foi iniciada por iniciativa de Diana Santos [10], trata-se de um Centro de Recursos distribuído para a Língua Portuguesa. O objectivo da Linguateca é promover a existência, discussão, avaliação, distribuição e manutenção

[1] http://www.linguateca.pt

de recursos relacionados com o processamento linguístico do Português a uma comunidade que os utiliza. A Linguateca é parcialmente financiada pela Fundação para a Ciência e Tecnologia de Portugal através do registo POSI/PLP/43931/2001, e co-financiada pelo projecto POSI.

1.2 A Floresta Sintáctica

A Floresta Sintáctica nasceu de um projecto de colaboração entre a Linguateca, pólo de Oslo, e o projecto VISL[2], cujos responsáveis são Diana Santos e Eckhard Bick respectivamente. Actualmente é constituída por texto jornalístico português e brasileiro, cedidos pelo Jornal Público e Folha de São Paulo.

A Floresta Sintáctica está publicamente disponível na Linguateca[3]. Sendo um corpus analisado sintacticamente (um conjunto de árvores sintácticas), a Floresta (ou um *treebank* em geral) constitui um relevante contributo para a comunidade já que [11]:

- foi revista e corrigida manualmente;
- está bem documentada, mantida, e com ferramentas associadas;
- reflecte um consenso entre a comunidade linguística;
- constitui um recurso para avaliar ou para avaliação, já que é facilmente transformável em algo que sirva de padrão para comparação;
- pode ser usada em processos de aprendizagem automática ou assistida de ferramentas de *parsing* ou *tagging*;
- permite a realização de estudos linguísticos, nomeadamente no campo sintáctico;

A Floresta Sintáctica é desenvolvida de forma a ser um recurso computacional para aplicações mais complexas, contendo a maior quantidade possível de informação sintáctica útil [12].

As árvores que constituem a Floresta Sintáctica são geradas automaticamente pelo programa PALAVRAS [4]. Posteriormente, cada árvore é revista por linguistas. A revisão manual de cada frase é exaustiva e demorada, mas garante uma qualidade superior na classificação sintáctica das árvores.

O formato base usado na Floresta Sintáctica é chamado árvores deitadas (ad). A nomenclatura deste formato é descrita ao pormenor em [2]. Segue-se um pequeno exemplo.

```
1 SOURCE: CETENFolha n=22 cad="Cotidiano" sec="soc" sem="94a"
2 CF22-3 Escuto Stones desde os 13 anos de idade.
3 A1
4 STA:fcl
5 =P:v-fin('escutar' PR 1S IND)   Escuto
6 =ACC:prop('Stones' M P) Stones
7 =ADVL:pp
8 ==H:prp('desde')        desde
```

[2] http://visl.sdu.dk/visl/pt
[3] http://www.linguateca.pt/Floresta

```
==P<:np
===>N:art('o' <artd> M P)       os
===>N:num('13' <card> M P)      13
===H:n('ano' M P)       anos
===N<:pp
====H:prp('de') de
====P<:n('idade' F S)   idade
=.
```

Seguindo o exemplo, a primeira linha contém informação relativa à secção de texto de onde foi retirada a frase, incluindo o contexto.

A segunda linha contém o identificador único da frase, com a respectiva frase (CF para CETENFolha da Folha de São Paulo, CP para CETEMPúblico do Público).

`A1` define uma secção onde foi elaborada uma análise sintáctica. É colocada antes do nó raiz da árvore. Uma árvore pode conter várias análises sintácticas distintas sendo estas numeradas consequentemente por A2, A3, A4. Cada secção analisada sintacticamente é separada por &&.

O nó raiz, que está definido na 4ª linha, inicia a construção da árvore. Para identificar os níveis da árvores é usado o símbolo '='. Cada símbolo adicional representa mais um nível de profundidade na árvore (esquematizada na figura 1. Cada nó que tenha um incremento de nível na linha seguinte é um nó não terminal.

Todos os nós contêm informação morfossintáctica, constituída por duas etiquetas separadas por ':'. As etiquetas do lado esquerdo (STA,ADVL,H,N<) são etiquetas de *função*, enquanto que as do lado direito (fcl, pp, prp) são etiquetas de *forma*. Os nós podem conter informação morfológica e gramatical associada às palavras e/ou à frase em que se inserem, como por exemplo, informação morfológica, lema da palavra, e etiquetas secundárias.

1.3 Processamento da Floresta Sintáctica

O formato árvores deitadas não é fácil de ser processado computacionalmente. Actualmente, a floresta é composta por aproximadamente 7500 árvores. As dificuldades subjacentes são:

- uma quantidade extensa e complexa de etiquetas de informação morfossintáctica que deve ser devidamente validade;
- o facto de ser produto de uma revisão manual, está sujeito a conter erros na sua sintaxe;
- a actualização periódica da Floresta Sintáctica obriga a re-processamentos.

O formato base da floresta, não permite facilmente uma extracção de informação eficiente. Para facilitar o acesso a este recurso, é necessário exportar para outros formatos a partir do formato base. Estes novos formatos devem suportar as características do original, oferecendo uma sintaxe mais simples e/ou uma API de programação.

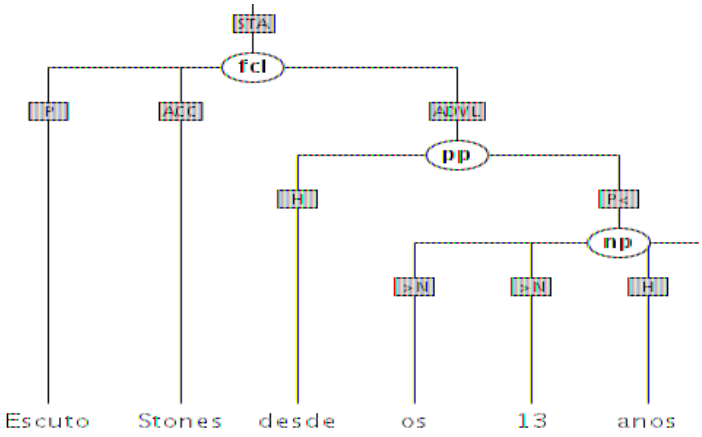


Figura 1. Representação gráfica de uma árvore.

2 Exportação da Floresta Sintáctica

Já existem diversos formatos, não baseados no XML, semelhantes ao formato base da floresta, desenvolvidos noutros países por projectos internacionais semelhantes, tais como: SUSANNE [9], Peen Treebank [8] e Tiger [6].

Estes formatos foram inicialmente desenvolvidos tendo em conta a morfologia gramatical das línguas a que se aplicavam, e o nível de detalhe exigido pelos linguistas e programadores para classificação sintáctica. Tendo em conta o último aspecto, não é fácil a exportação para outros formatos devido principalmente a:

- eliminação de informação morfológica originalmente contida pelo formato original. Resulta na perda de informação considerada importante em termos linguísticos;
- alteração da notação para indexar a floresta. Se possível, é desejável manter à parte a notação original;
- o novo formato contêm regras definidas rigidamente pelos autores;
- dificuldade em adaptar a informação morfológica original. Alguns dos formatos existentes não estão preparados para certos tipos de classificação morfológica;
- ferramentas inadequadas ou insuficientes para explorar o formato.

Qualquer um destes pontos inviabiliza a exportação correcta da Floresta Sintáctica nos formatos mencionados.

Dada a facilidade de estruturação do XML, pretende-se definir um formato que consiga reter toda a informação da floresta sintáctica.

3 A Floresta Sintáctica em XML

Nesta secção apresentamos duas abordagens à representação da floresta sintáctica em XML.

3.1 Tiger-XML

O formato Tiger-XML foi desenhado como uma linguagem de interface para o formato Tiger [6]. Este formato possui uma ferramenta para pesquisa, designada por TIGERSearch [7].

O TIGERSearch permite explorar sintacticamente corpora anotado, mediante a visualização do *treebank*. Pode-se obter fenómenos relevantes sintacticamente a partir de frases, propriedades lexicais e contexto da palavra na frase. Actualmente pode servir de interface a alguns *treebanks* existentes [7].

Para converter o formato base da Floresta Sintáctica para o formato Tiger-XML é usado um programa em Perl, originalmente desenvolvido pelo Eckhard Bick. Este programa foi modificado sucessivamente de forma a adaptar-se à Floresta Sintáctica.

Segue-se a descrição do processo de conversão:

- conversão do formato árvores deitadas (ad) para Tiger-XML em *bruto*;
- análise do código XML gerado, eliminando elimina secções inválidas de XML. Estas secções inválidas ocorrem geralmente devido a erros existentes no formato base;
- geração automática da *header* do XML, contendo a nomenclatura das etiquetas de função e forma das palavras. O *script* identifica e recolhe os valores dos atributos existentes no código XML. Todos os valores destes atributos estão descritos num ficheiro de texto. Se os valores dos atributos já possuírem uma descrição definida, são adicionados ao header do XML. Este processo ajuda a identificar árvores que possuam atributos errados. Este processo também ajuda a identificar árvores com erros no formato base;
- O processo é finalizado pela indentação do código XML usando a ferramenta de domínio público *xmllint*.

Durante o processo é adquirida informação estatística e identificados erros, o que permite o desenvolvimento de ferramentas de validação sobre o formato das árvores deitadas. Segue-se um exemplo de uma árvore no formato Tiger-XML:

```
<s id="s85" ref="CF22-3"
      source="CETENFolha n=22 cad=Cotidiano sec=soc sem=94a"
      forest="1" text="Escuto Stones desde os 13 anos de idade.">
  <graph root="s85_500">
    <terminals>
      <t id="s85_2" word="Escuto" lemma="escutar" pos="v-fin"
        morph="PR 1S IND" extra="--" />
      <t id="s85_3" word="Stones" lemma="Stones" pos="prop"
        morph="M P" extra="--" />
```

```
10          <t id="s85_4" word="desde" lemma="desde" pos="prp"
11            morph="--" extra="--" />
12          <t id="s85_5" word="os" lemma="o" pos="art"
13            morph="M P" extra="artd" />
14          <t id="s85_6" word="13" lemma="13" pos="num"
15            morph="M P" extra="card" />
16          <t id="s85_7" word="anos" lemma="ano" pos="n"
17            morph="M P" extra="--" />
18          <t id="s85_8" word="de" lemma="de" pos="prp"
19            morph="--" extra="--" />
20          <t id="s85_9" word="idade" lemma="idade" pos="n"
21            morph="F S" extra="--" />
22          <t id="s85_10" word="." lemma="--" pos="pu"
23            morph="--" extra="--" />
24        </terminals>
25        <nonterminals>
26          <nt id="s85_500" cat="s">
27            <edge label="STA" idref="s85_501" />
28          </nt>
29          <nt id="s85_501" cat="fcl">
30            <edge label="P" idref="s85_2" />
31            <edge label="ACC" idref="s85_3" />
32            <edge label="ADVL" idref="s85_502" />
33          </nt>
34          <nt id="s85_502" cat="pp">
35            <edge label="H" idref="s85_4" />
36            <edge label="P<" idref="s85_503" />
37          </nt>
38          <nt id="s85_503" cat="np">
39            <edge label=">N" idref="s85_5" />
40            <edge label=">N" idref="s85_6" />
41            <edge label="H" idref="s85_7" />
42            <edge label="N<" idref="s85_504" />
43          </nt>
44          <nt id="s85_504" cat="pp">
45            <edge label="H" idref="s85_8" />
46            <edge label="P<" idref="s85_9" />
47          </nt>
48        </nonterminals>
49      </graph>
50    </s>
```

Este formato é particularmente difícil de utilizar programaticamente por conter secções diferentes relativas aos símbolos terminais e não terminais (o que obriga à criação de tabelas de indirecção).

3.2 SimTreeML

O formato Tiger-XML tem várias vantagens mas as suas árvore não estão directamente identificáveis.

O formato SimTreeML (exemplo seguinte) contém uma representação dos níveis da árvores mais fácil de identificar, e tem uma notação mais compacta. O extracto seguinte descreve em SimTreeML a mesma árvore da secção anterior.

```
<extract>
  <source>CF22-3 Escuto Stones desde os 13 anos de idade.</source>
  <tree cat='fcl' fun='STA'>
     <t cat='v-fin' lema='escutar' args='PR 1S IND' fun='P'>Escuto</t>
     <t cat='prop' lema='Stones' args='M P' fun='ACC'>Stones</t>
     <tree cat='pp' fun='ADVL'>
        <t cat='prp' lema='desde' fun='H'>desde</t>
        <tree cat='np' fun='P<'>
           <t cat='art' lema='o' args='M P' fun='>N' extra='artd'>os</t>
           <t cat='num' lema='13' args='M P' fun='>N' extra='card'>13</t>
           <t cat='n' lema='ano' args='M P' fun='H'>anos</t>
           <tree cat='pp' fun='N<'>
              <t cat='prp' lema='de' fun='H'>de</t>
              <t cat='n' lema='idade' args='F S' fun='P<'>idade</t>
           </tree>
        </tree>
     </tree>
     <punt ort='.'/>
  </tree>
</extract>
```

Este tipo de representação XML, permite um fácil processamento no sentido de construir ou visualizar as árvores.

Uma das funcionalidades óbvias exigida a qualquer processador de extractos da floresta é o permitir a visualização de árvores através da definição de um conjunto de CSS ou XSLT (correspondentes a um conjunto de vistas eficazes).

Na figura 2 mostra-se a visualização directa de árvores da Floresta no formato descrito, usando apenas um *browser* (Mozilla) e uma CSS. Esta CSS desenha uma área em volta de cada **tree** e esconde alguns dos atributos da informação (para não sobrecarregar o utilizador com demasiada informação).

3.3 Comparação dos formatos TigerXML – SimTreeML

Dum modo simplificado a abordagem TigerXML corresponde a codificar a informação da Floresta como sendo um grafo (descrevendo-se os seus nós e ramos). A abordagem SimTreeML corresponde a descrever árvores *puras*.

Os grafos TigerXML têm uma leitura mais complexa em XML do que as árvores SimTreeML, no entanto oferecem um maior poder expressivo (que pode ser usado para enriquecer futuramente as árvores anotadas com informação adicional semântico-linguística diversa como por exemplo ligação entre os pronomes e os nomes que referem).

O TigerXML tem a vantagem de ser um formato bastante usado e difundido e dispor de um conjunto de ferramentas de suporte como sejam o TigerSearch[7].

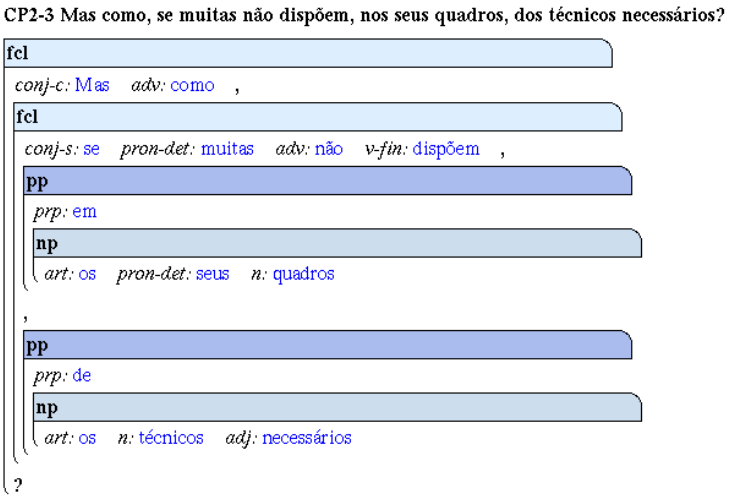


Figura 2. Extracto formatado com uma CSS de uma árvore da floresta no formato SimTreeML

Por outro lado, juntamente com os tradutores para o formato SimTreeML foi construído um módulo Perl (`Lingua::TreeBank::Dirty`), que facilita a construção de processadores da Floresta Sintáctica a partir deste formato.

4 O módulo Lingua::TreeBank::Dirty

O módulo Perl `Lingua::TreeBank::Dirty` usa a Floresta em formato SimTreeML para construir um conjunto de estruturas internas que tornem viável a criação de processadores usando métodos habituais em XML, mas que não obriguem a ter a totalidade da estrutura em memória principal.

Um dos objectivos deste módulo é que a descrição dos processadores seja feita dum modo tão compacto e simples quanto possível.

Nesta secção apresentaremos apenas dois exemplos baseados na função `downTr` (*down translate*), com o intuito de mostrar que é possível construir com pouco esforço pequenos programas específicos que dificilmente encontraríamos em aplicações de tratamento de corpora anotados.

4.1 Exemplo para estudos *cognitivos*

Por vezes em estudos cognitivos é por vezes necessário procurar palavras relacionadas com outras em certos contextos de uso.

Neste exemplo (um pouco artificial) construiu-se um programa Perl usando o `Lingua::TreeBank::Dirty` que, dado um verbo (comer), procura todas os extractos que contêm esse lema e processa a respectiva frase extraindo os nominais nela inscluídos.

Este processador imprime todas as palavras cuja categoria (`$v{cat}`) seja `n` (i.e. os nominais)

O `Lingua::TreeBank::Dirty` usa internamente o módulo XML::DT [3,14], herdando dele uma variedade de funções (que aqui não serão detalhadas) que permitem consulta do contexto, alteração de subárvores e de atributos.

Dum modo simples, a construção dum processador é feito através de:

- difinição de um padrão textual de domínio de interesse;
- definição de uma função de processamento para os símbolos terminais (palavras);
- definição de uma função de processamento para os nós internos da árvore sintáctica.

Cada uma destas funções recebe como parámetro os atributos (%v) e o conteúdo ($c).

```
1  use Lingua::TreeBank::Dirty "/home/jj/linguateca/FS";
2  downTr({ -patt => "-w comer",
3           -end => sub{print "\n"},
4              t => sub{print "$c, "  if $v{cat} eq "n" } });
```

Notas:

- linha 2 – selecciona os extractos contendo o lema “comer”. Naturalmente que a aplicação deste padrão tem consequências na quantidade de árvores a analisar e portanto no tempo total de execução.
- linha 3 – mudar de linha após processar cada extracto;
- linha 4 – imprimir as palavras que estejam classificadas na categoria “n”.

Quando aplicado à Floresta Sintáctica, a saída do programa anterior é:

```
1  carvão, restaurante, tribunal, carne,
2  invernos, caminhos, aldeia, tangerinas, cascas, montes, neve,...
3  praia, colina, copos, água, restaurante, trutas, castanhas, lado,
4  milhões, pessoas, música, relações, mês, muçulmano,
5  ......
6  arroz, marisco,
7  vinho, vinho, milhão, portugueses, crianças, sopas, cavalo,
8  bicicleta, sacos-cama, roupa, dinheiro,
9  pinguim, carne, bode, magote, sertanejos,
10 tipos, refeições, símbolos, valor, pessoas,
```

Como se constacta, foi possível em poucas linhas fazer a construção de um processador. Esta facilidade viabiliza a construção de processadores para resolver situações pontuais e esporádicas, i.e., ajuda à existência de uma bancada de trabalho onde se possa tirar partido da Floresta.

4.2 Exemplo gramatical

Neste exemplo processa-se (uma parte de) a Floresta para fazer a extracção de uma gramática implícita.

Cada ramificação corresponde ao uso de uma produção. Faz-se também a acumulação e contagem das produções encontradas.

```
1  downTr( { tree => sub{ $p = "$v{cat} -> $c";
2                         $prod{$p}++;
3                         $v{cat} },
4               t => sub{ "[$v{cat}]" },
5            punt => sub{ $v{ort} }
6          });

7  #... imprimir ordenadas por ordem inversa de ocorrências
```

Este programa quando aplicado à Floresta, constrói um ficheiro de produções sendo as primeiras linhas as seguintes:

```
1  pp ->  [prp] np           17250
2  np ->  [art] [n]          5670
3  np ->  [art] [n] pp       4621
4  np ->  [art] [prop]       2843
5  pp ->  [prp] [n]          2224
6  np ->  [n] pp             1768
7  pp ->  [prp] icl          1398
8  pp ->  [prp] [prop]       1302
9  np ->  [art] [n] [adj]    1221
10 np ->  [pron-det] [n]     1107
11 vp ->  [v-fin] [v-pcp]    749
12 np ->  [n] [adj]          718
13 ...
14 fcl ->  np [v-fin] np .   188
```

5 Conclusões

A conversão entre formatos permitiu a descoberta localizada de diversos erros na sintaxe do formato árvores deitadas. Assim, desenvolveram-se *scripts* para validação da Floresta. Os relatórios de erros encontrados podem ser enviados aos linguistas que revêm a Floresta.

A distribuição da Floresta Sintáctica em diversos formatos permite abranger um maior número de pessoas que estudam e desenvolvem trabalho no processamento do português, assim como um maior leque de possibilidades para o acesso e manipulação deste recurso.

Para além das ferramentas clássicas de consulta, visualização e estatísticas, salienta-se a importância da construção de módulos que facilitem a escrita de processadores novos. Na verdade, é nossa convicção que é importante associar

a cada formato em que se codifique a floresta, um conjunto de ferramentas que tornem simples e eficiente o seu processamento.

Além da comparação do Tiger-XML com o SimTreeML, seria ainda importante fazer a comparação com um terceiro formato, usado pelo XML Corpus Encoding Standard[5].

Referências

1. Susana Afonso. A floresta sintá(c)tica como recurso. Documentação disponível na Linguateca, 2003.
2. Susana Afonso. *Árvores deitadas: Descrição do formato e descrição das opções de análise na Floresta Sintá(c)tica*, 2004.
3. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, Maio 1999.
4. Eckhard Bick. The parsing system palavras, automatic grammatical analysis of portuguese in a constraint grammar framework. 2000.
5. Vassar College Department of Computer Science. XCES — Corpus Encoding Standard for XML. `http://www.xml-ces.org/`.
6. Stefanie Dipper, Thorsten Brants, Wolfgang Lezius, Oliver Plaehn, and George Smith. The tiger treebank. In *Third Workshop on Linguistically Interpreted Corpora*, 2001.
7. Esther König, Wolfgang Lezius, and Holger Voormann. *TIGERSearch 2.1 User's Manual*, 2003.
8. Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewic. Building a large annotated corpus of english: The penn treebank. In *Computational Linguistic*, volume 19, pages 313–330, June 2003.
9. Geoffrey Sampson. *SUSANNE Corpus and Analytic Scheme*, June 2004.
10. Diana Santos. O projecto processamento computacional do português: Balanço e perspectivas. In Maria das Graças Volpe Nunes, editor, *V Encontro para o processamento computacional da língua portuguesa escrita e falada*, pages 105–113, São Paulo, 2000. ICMC/USP.
11. Diana Santos. The floresta experience. Presentation for the Swedish Treebank Symposium Växjö, Novembro 2002.
12. Diana Santos. Timber! issues in treebank building and use. In Nuno J. Mamede, Jorge Baptista, Isabel Trancoso, and Maria das Graças Volpe Nunes, editors, *Computational Processing of the Portuguese Language, 6th International Workshop*, pages 151–158. Springer Verlag, 2003.
13. Diana Santos, Alberto Simões, Ana Frankenberg-Garcia, Ana Pinto, Anabela Barreiro, Belinda Maia, Cristina Mota, Débora Oliveira, Eckhard Bick, Elisabete Ranchhod, José João Dias de Almeida, Luís Cabral, Luís Costa, Luís Sarmento, Marcirio Chaves, Nuno Cardoso, Paulo Rocha, Rachel Aires, Rosário Silva, Rui Vilela, and Susana Afonso. Linguateca: um centro de recursos distribuído para o processamento computacional da língua portuguesa. In IBERAMIA 2004, editor, *Workshop on Linguistic Tools and Resources for Spanish and Portuguese*, 2004.
14. Alberto Simões. XML::DT - down translating xml. *The Perl Review*, 1(1), 2004.

XcUBE - The XUL User Interface Language Unified Build Environment

Luis Filipe Almeida Santos — FEUP
Nelson Jorge Silva Rodrigues — FEUP
Ricardo Jorge Marques Veloso — FEUP

Resumo

XML User Interface Language (XUL), é um dialecto XML para a descrição de interfaces gráficas com o utilizador.

O XcUBE é um ambiente de desenvolvimento (IDE) para este dialecto, englobando entre outros componentes um editor visual para a linguagem e um editor textual com syntax highlighting.

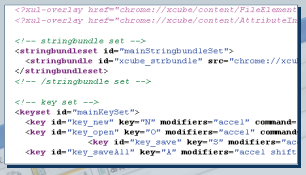
Este trabalho foi realizado no âmbito da disciplina de Processamento Estruturado de Documentos da Licenciatura em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.

Embora o XcUBE não esteja, neste momento, com todas as funcionalidades completamente acabadas, o seu desenvolvimento passou para a comunidade open source, envolvido na licença Mozilla Public License (MPL 1.1).

XcUBE

The XUL Unified Build Environment

A XML User Interface Language (XUL), é um dialecto XML para a descrição de interfaces gráficas com o utilizador.
O XcUBE é um ambiente de desenvolvimento (IDE) para este dialecto, englobando entre outros componentes um editor visual para a linguagem e um editor textual com *syntax highlighting*.
Este trabalho foi realizado no âmbito da disciplina de Processamento Estruturado de Documentos da Licenciatura em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.



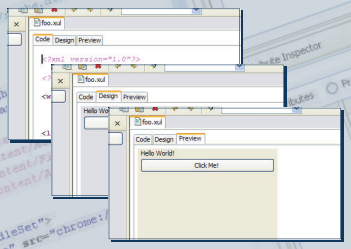
Editor de texto com *syntax highlighting* para XML e dialectos derivados.

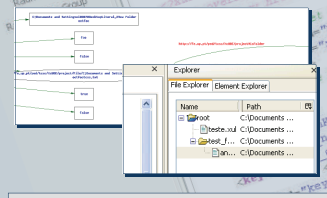
- É implementado em JavaScript com recurso à API DOM e encapsulado como um componente XBL.
- A coloração é possível por análise léxica do código e criação da respectiva árvore XHTML em tempo real.

Visualização sincronizada de três vistas — código, desenho e pré-visualização — para ficheiros XUL.

- Vista de desenho utiliza o paradigma *Drag & Drop*.
- Vista pré-visualizada funciona como *sandbox* permitindo o teste das funcionalidades do código desenvolvido.

Nos restantes ficheiros é aberto um editor de texto genérico ou é utilizado o editor de sistema associado.





A tecnologia RDF é fundamental para descrever conceitos centrais da aplicação tais como o projecto e os componentes XUL/XBL disponibilizados.

- RDF relativo ao projecto é criado e mantido pela aplicação em concordância com as opções do utilizador.
- RDF dos componentes é definido estaticamente.

A plataforma Mozilla foi usada com bastante sucesso para desenvolver este projecto. As funcionalidades oferecidas permitem a fácil criação de aplicações (*web based* ou não).
XUL como dialecto para definição de interfaces gráficas, mesmo que tenha sido desenvolvido com um enfoque bastante específico, revelou-se bastante completo e eficiente.
O XcUBE ainda está em fase de prova de conceito, no entanto o seu desenvolvimento prosseguirá no seio da comunidade *open source*.

Trabalho realizado por:
Luis Filipe Almeida Santos <luis.santos@fe.up.pt>
Nelson Jorge Silva Rodrigues <nelson@fe.up.pt>
Ricardo Jorge Marques Veloso <ricardo.veloso@fe.up.pt>

Mais informações em:
http://xcube.sourceforge.net

Universidade do Porto
Faculdade de Engenharia
FEUP

Especificação e Geração Automática de Navegadores para Redes Semânticas baseados em Interfaces Web

Miguel Domingues — Dep. Informática, U. Minho

Resumo

Os Topic Maps são um conjunto de standards que resultam da investigação contemporânea em uma nova área - Semantic Web.

A ideia base deste projecto é propôr e desenvolver uma arquitectura aplicacional para a navegação Web sobre um modelo relacional de Topic Maps, apresentando e discutindo as opções estruturais mais importantes.

No decorrer deste trabalho e como principal resultado, pretende-se obter uma estrutura de objectos intermédia que constitua uma API de interacção, suficientemente robusta e modular, entre o modelo relacional e o cliente, com a particular utilização de um dos standards da família - "Xml Topic Maps"(XTM).

XATA2005
XML: Aplicações e Tecnologias Associadas

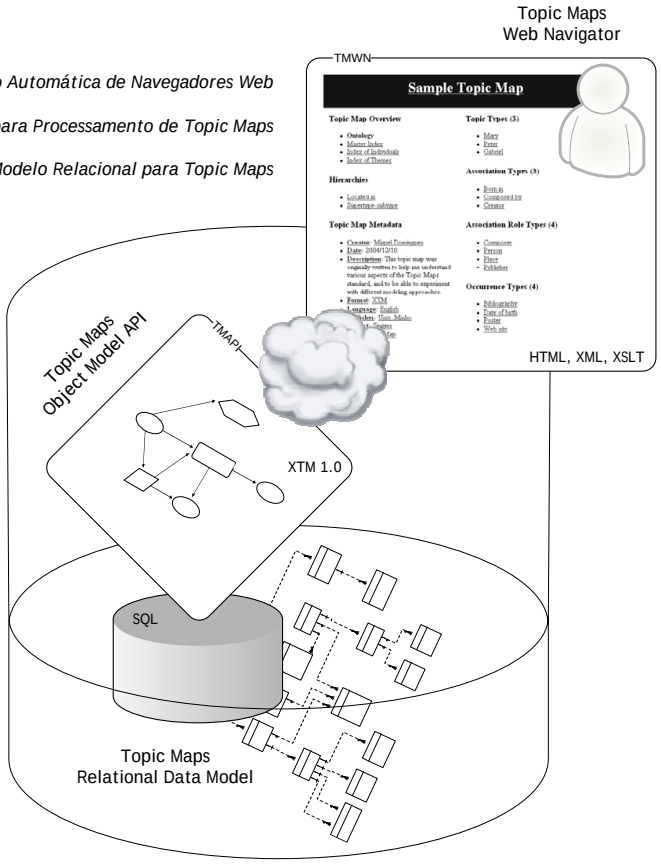
Especificação e Geração Automática de Navegadores para Redes Semânticas baseados em Interfaces Web

Miguel Domingues, Universidade do Minho

Geração Automática de Navegadores Web

API para Processamento de Topic Maps

Modelo Relacional para Topic Maps



GML Plug-in for JUMP

Manuel Mesquita Gomes — FEUP / INESC Porto
Fernando Raimundo Gomes — FEUP

Resumo

O projecto visa a realização de um plug-in para o JUMP, aplicação para visualização e manipulação de informação geográfica. Quer-se tornar possível importar documentos GML no JUMP, sem ser necessário escrever manualmente ficheiros template, o que acontece actualmente.

Estes templates são gerados transformando o conteúdo do documento GML com XSLT.

XATA2005
XML: Aplicações e Tecnologias Associadas

GML Plug-in for JUMP

Manuel Mesquita T. F. Gomes, F.E.U.P. / INESC Porto
Fernando Raimundo Gomes, F.E.U.P.

GML

A "*Geography Markup Language*" é uma especificação, em *XML Schema*, para a modelação, transporte e armazenamento de informação geográfica.

Objectivos

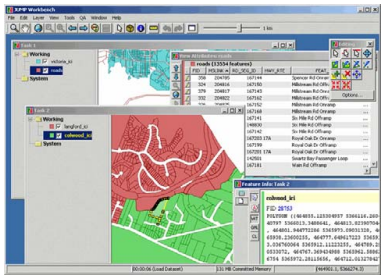
- disponibilizar uma ferramenta aberta e não proprietária para a representação de dados geoespaciais
  - separar o conteúdo da forma de visualização dos mapas
  - armazenar as aplicações e informação geográfica em GML, ou converter de outro formato de armazenamento e usar o GML apenas para transporte dos dados

JUMP Unified Mapping Platform

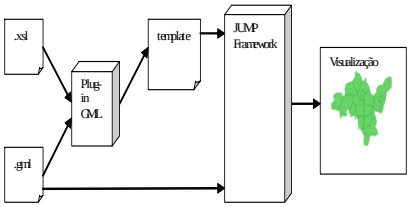
Alicação para visualização e processamento de dados geoespaciais;

Inclui funções comuns a outros produtos GIS para análise e manipulação de dados geoespaciais;

Apresenta uma *framework* extensível para desenvolvimnto e execução de aplicações customizadas



GML Plug-in for JUMP



A partir do documento GML e de um ficheiro XSL, onde estão definidas as regras de transformação dos elementos GML, o *plug-in* gera um ficheiro *template*.

Este *template* é enviado, juntamente com o documento GML, para a *framework* do JUMP, que gera o mapa de visualização, utilizando as ferramentas já existentes para importação de documentos GML com o respectivo ficheiro *template*.

Especificação e implementação de um repositório de objectos de ensino

Paulo Jorge Dias Domingues — Dep. Informática, U. Minho

Resumo

Num Learning Management System (LMS) um dos componentes é o repositório dos Learning Objects (LO). Em termos práticos, um LO é composto pelos ficheiros ou recursos de aprendizagem e pelos metadados (LOM) que o catalogam.

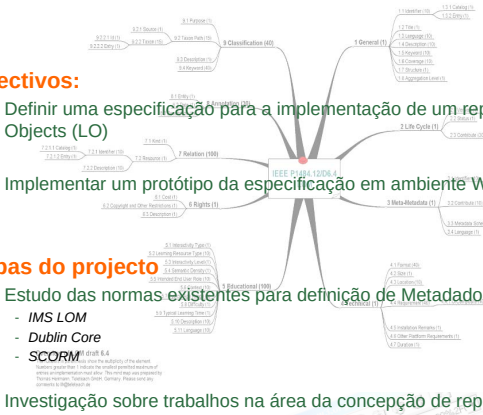
O primeiro passo deste trabalho, consistirá no estudo das normas existentes e na definição de uma especificação para a definição de um Learning Objects Repository (LOR) de acordo com a norma SCORM.

Em continuação, será efectuada uma implementação da especificação sobre uma BD relacional, bem como a construção de uma camada de interface para manutenção e query aos dados armazenados.

XATA2005
XML: Aplicações e Tecnologias Associadas

Especificação e implementação de um Repositório de Objectos de Ensino

Paulo Jorge Domingues, Universidade do Minho

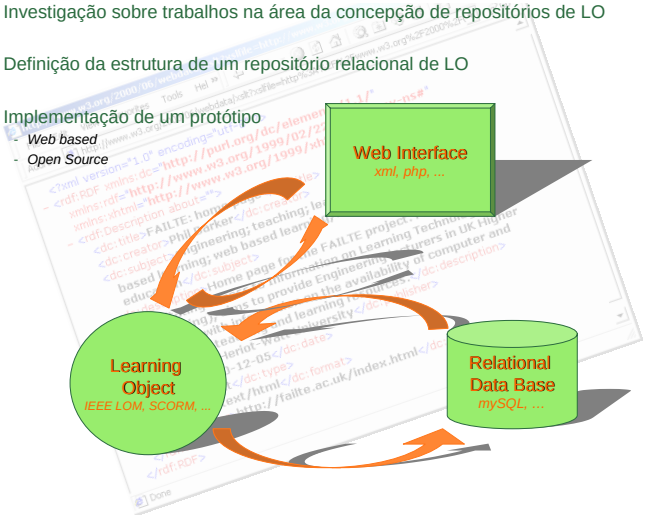


Objectivos:

- Definir uma especificação para a implementação de um repositório de Learning Objects (LO)
- Implementar um protótipo da especificação em ambiente Web Open Source

Etapas do projecto

- Estudo das normas existentes para definição de Metadados
  - *IMS LOM*
  - *Dublin Core*
  - *SCORM*
- Investigação sobre trabalhos na área da concepção de repositórios de LO
- Definição da estrutura de um repositório relacional de LO
- Implementação de um protótipo
  - *Web based*
  - *Open Source*



DisQS - Web Services Based Distributed Query System

Marco Fernandes — IEETA, Universidade de Aveiro
Joaquim Arnaldo Martins — IEETA, Universidade de Aveiro
Joaquim Sousa Pinto — IEETA, Universidade de Aveiro
Pedro Almeida — IEETA, Universidade de Aveiro
Helder Zagalo — IEETA, Universidade de Aveiro

Resumo

DisQS é um subsistema, baseado em web services, para acesso e pesquisa num repositório distribuído. Permite a gestão remota de catálogos, comunicação entre computadores por mensagens XML, mecanismos de replicação de documentos, balanceamento de carga e cria transparência quanto às diferentes tecnologias de indexação.

DisQS

Web Services based Distributed Query System

Marco Fernandes . Joaquim Arnaldo Martins . Joaquim Sousa Pinto . Pedro Almeida . Helder Zagalo

Objectivos

- Criação de um módulo de acesso e pesquisa sobre um repositório distribuído, escalável e de grande interoperabilidade.
- Permitir a (des)agregação de repositórios de forma transparente de um repositório virtual único.
- Criação de um Web Service de administração para gestão remota de catálogos de informação.
- Implementação de repositórios digitais distribuídos através do módulo DisQS.
- Transparência quanto às tecnologias de indexação.

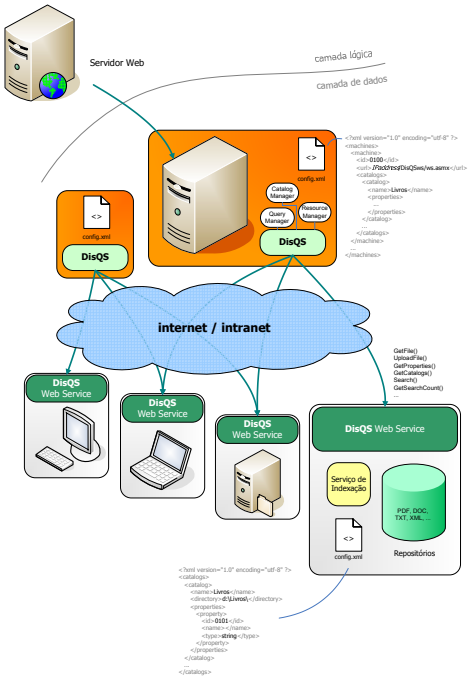
Funcionalidades

- Pesquisa simultânea em todos os repositórios da rede DisQS.
- Criação e edição remota de catálogos.
- Mecanismos de replicação de dados e balanceamento de carga.
- Implementação de caches de recursos e de resultados de pesquisas.
- Mecanismos de substituição ou remoção automática de recursos desactualizados.
- Métodos genéricos para fazer o *upload* e o *download* de ficheiros XML e ficheiros de imagens.

  Por exemplo: redimensionamento e/ou conversão de formatos prévio ao pedir uma imagem.

Tecnologias

- Linguagens de programação
  - C# (.NET Framework)
- Standards
  - XML, SOAP, WSDL (Web Services)
- Serviços de indexação
  - Indexing Service (Microsoft)
  - Swish-e
  - Bases de dados relacionais



Protótipo

- Integração no SInBAD (Sistema Integrado para Bibliotecas e Arquivos Digitais)
  - Programa Aveiro Digital 2003-2006
  - POSI
- Rede DisQS constituída por diversos computadores da Universidade de Aveiro.
- Pesquisa distribuída em diversos tipos de documentos (PDF, XML, HTML, DOC, TXT, entre outros).

Resultados

- Integração em Bibliotecas Digitais e Arquivos Digitais para implementar repositórios e mecanismos de pesquisa distribuídos.
- Resultados provisórios (protótipo):
  - 6 computadores ligados à rede
  - cerca de 1 milhão de documentos armazenados no repositório
  - pesquisas duram entre centésimos de segundo (para poucas centenas de resultados) e 2-3 segundos para palavras muito comuns

EspiritUs – Sistema de Anotações XML para Documentos Web

Marco Fernandes — IEETA, Universidade de Aveiro
Miguel Alho — IEETA, Universidade de Aveiro
Joaquim Arnaldo Martins — IEETA, Universidade de Aveiro
Joaquim Sousa Pinto — IEETA, Universidade de Aveiro
Pedro Almeida — IEETA, Universidade de Aveiro

Resumo

O espiritUs é um sistema de anotações XML para documentos web. As anotações ficam associadas ao URL do documento e podem ser públicas ou privadas. Uma vez que o acesso ao repositório é feito através de Web Services, o sistema tem uma grande interoperabilidade e permite o acesso às anotações de qualquer ponto de ligação à rede.

espiritUs

sistema de anotações xml para documentos web

Marco Fernandes . Miguel Alho . Joaquim Arnaldo Martins . Joaquim Sousa Pinto . Pedro Almeida

Objectivos

- Desenvolver um sistema de anotação de documentos web de elevada interoperabilidade.
- Criar uma ligação entre as anotações e os documentos web anotados.
- Permitir a visualização e a edição das anotações em qualquer ponto de acesso à Internet.
- Permitir efectuar pesquisas no repositório de anotações.

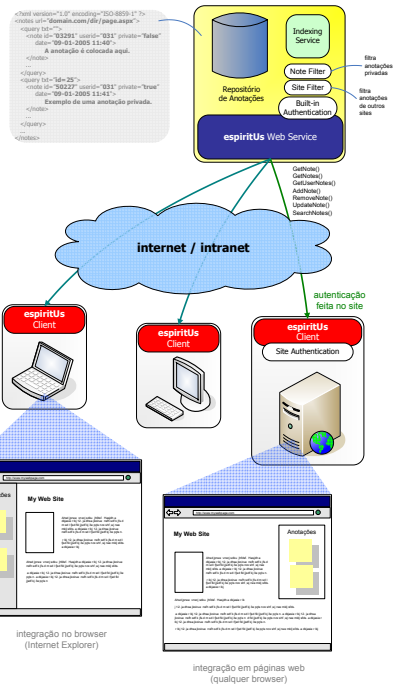
Funcionalidades

- Anotações armazenadas em documentos XML.
- Utilização de um XML único por URL (a menos do *querystring*):

  Por exemplo, uma anotação ao documento http://servidor.com/pagina.aspx?id=2 fica armazenada no ficheiro pagina.aspx.xml no directório [Repositorio]/servidor.com
- Utilização de um filtro XML para indexação das anotações pelo Indexing Service.
- Distinção entre anotações públicas e privadas.
- As anotações públicas são de livre consulta, mas com edição limitada ao seu autor.
- As anotações privadas estão disponíveis apenas para o autor.
- Pesquisa de anotações por texto livre, autor e data.
- Desenvolvido um «band object», para integração no browser Internet Explorer, e um módulo web, para integrar em páginas web (em qualquer browser).

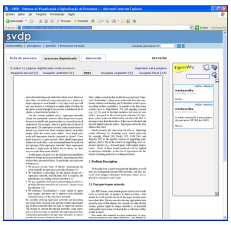
Tecnologias

- Linguagens de programação
  - C#, VB.NET (.NET Framework)
- Standards
  - XML, SOAP, WSDL (Web Services)
- Indexação
  - Indexing Service (Microsoft)
  - Filtro XML QLXFilter (QuiLogic)
- Web
  - IIS 6.0 (Microsoft)



integração no browser (Internet Explorer)

integração em páginas web (qualquer browser)



Caso prático

Sistema de Visualização e Digitalização de Processos da Provedoria de Justiça

- Componente de anotações integrado na aplicação web desenvolvida.
- Anotação dos processos feita pelos funcionários da Provedoria.
- Complementar informação da base de dados já existente e permitir uma pesquisa mais orientada.
- Filtro ISAPI transforma URLs para que a distribuição das anotações seja feita por um número maior de ficheiros XML.

  Por exemplo, transforma .../processo.aspx?num=23/02 (1 XML para todos os processos) em .../processos/2002/23/ (1 XML por processo).

Resultados

- Sistema flexível que permite a anotação de qualquer documento web.
- Repositório centralizado, numa arquitectura servidor-cliente, permite acesso às anotações em qualquer ponto de acesso à rede.
- Arquitectura baseada em Web Services permite reutilizar *middleware* tanto em aplicações web como desktop.

Aplicação de Reverse Engineering: Java para UML

Luis Ramos — FEUP
Pedro Strecht — FEUP

Resumo

Desenvolvimento de uma aplicação web de reverse engineering para conversão de código Java em diagramas de classes UML, produzindo o resultado em dois dialectos XML: XMI e SVG.

O formato XMI descreve a semântica das classes e o posicionamento dos elementos gráficos no diagrama, podendo ser importado por ferramentas CASE para visualização e edição. O formato SVG descreve objectos gráficos e pode ser visualizado num web browser.

Inicialmente ocorre a transformação de código Java para o dialecto JavaML com o propósito de identificar todas as classes, atributos e métodos. Esta transformação é efectuada por um conversor desenvolvido por Greg Badros. De seguida efectua-se a transformação de JavaML em XMI, através de uma XSLT.

O documento XMI produzido inclui a componente semântica das classes e a estrutura da componente de layout, sem informação do posicionamento dos elementos gráficos.

O cálculo do posicionamento das classes no diagrama é feito através de um módulo que utiliza a interface DOM da framework .NET alterando o documento XMI.

A utilização de uma XSLT desenvolvida por Mario Jeckle, converte o resultado de XMI em SVG, permitindo também oferecer este formato ao utilizador.

XATA2005

XML: Aplicações e Tecnologias Associadas

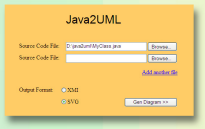
Aplicação de Reverse Engineering: Java para UML

Luís Ramos e Pedro Strecht

lramos@fe.up.pt | pstrecht@fe.up.pt

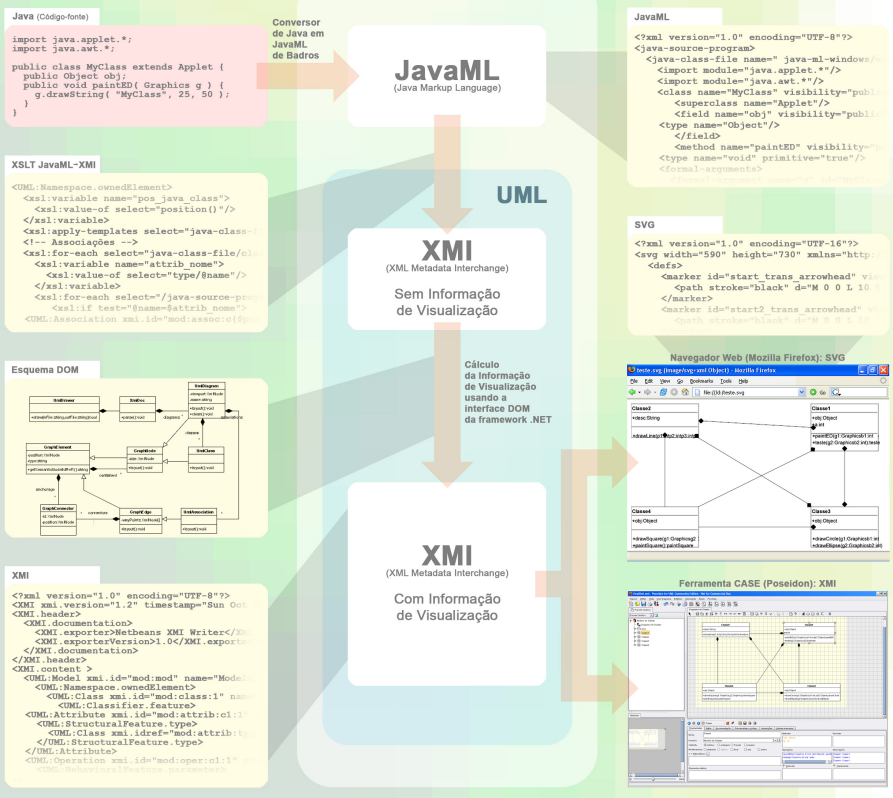
FEUP

Faculdade de Engenharia da Universidade do Porto



No âmbito da disciplina de "Processamento Estruturado de Documentos" do Mestrado em Engenharia Informática, foi desenvolvida uma aplicação web de conversão de código-fonte em Java para a sua representação em UML, disponibilizando dois formatos de saída:

- **XMI representação de metainformação sobre modelos em formato XML;**
- **SVG para apresentação directa num web browser..**



XATA2005, 10 e 11 de Fevereiro de 2005, Casa da Torre, Vila Verde, Braga

Definição e implementação de um sistema de testes e exames para e-Learning

António Lira Fernandes — Dep. Informática, U. Minho

Resumo

Com a globalização dos sistemas de e-Learning e a constante evolução das tecnologias e ferramentas de produção de conteúdos importa encontrar normas que garantam a interoperabilidade dos sistemas e a reutilização de conteúdos. Existem alguma organizações responsáveis por definir essas normas. Fazem-no através da criação de fóruns livres, entre os principais fornecedores de soluções e fornecedores de conteúdos, instituições académicas e governamentais entre outras, destinados a promoverem a negociação de um modelo global.

Neste projecto vamos centrar a nossa atenção no conjunto de normas QTI/SCORM no domínio dos testes e exames e em especial as propostas da IMS Global.

A IMS Global é uma organização não lucrativa que produz a especificação Question & Test Interoperability (QTI) que está na versão 2.0 sobe a forma de public draft desde Junho de 2004.

As questões de partida deste projecto são: modelo de dados para testes e exames; granularidade dos elementos constituintes dessas avaliações; interligação com os percursos de aprendizagem e com os Learning Object; modelo de dados para repositórios de questões (itemBank); articulação com sistemas de aprendizagem (LearningSystem); requisitos e estrutura dos sistemas de avaliação (AssessmentDeliverySystem), compatibilidade das aplicações de produção de questões (AuthoringTool).

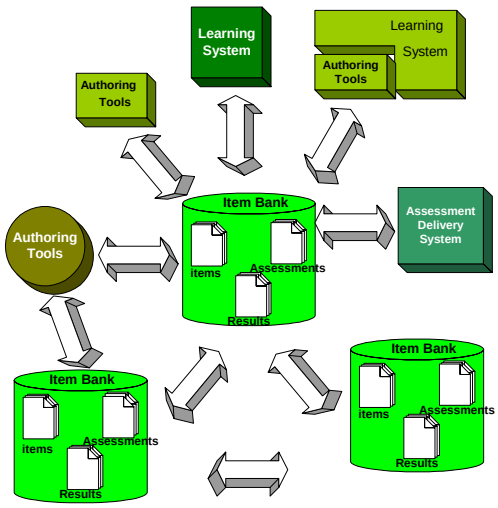
O primeiro objectivo deste projecto é apresentar um modelo geral que responda às principais questões dos testes e exames em e-Learning representando um modelo em UML para os diversos sistemas e uma especificação em XML para as avaliações (Assessment).

Um segundo objectivo será implementar um protótipo de um sistema de testes e exames para e-Learning no Ensino Secundário.

XATA2005
XML: Aplicações e Tecnologias Associadas

Definição e implementação de um sistema de testes e exames para e-Learning

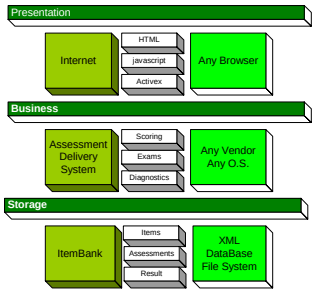
António José Lira R. Fernandes , Universidade do Minho



Garantir a interoperabilidade dos sistemas e a reutilização de conteúdos.

Resultados esperados:

- **Estudo da norma SCORM para testes e exames**
- **Estudo da oferta nacional, nomeadamente a do ministério da educação**
- **Modelo relacional de acordo com QTI para testes e exames**
- **Especificação dos componentes de um sistema para testes e exames**
- **Implementação de um protótipo de sistema de apresentação de testes e exames**



Camadas independentes do fornecedor e das tecnologias envolvidas.

Desenho de um Sistema de Workflow baseado em XML Web Services

Ricardo Luis — Dep. Informática, U. Minho

Resumo

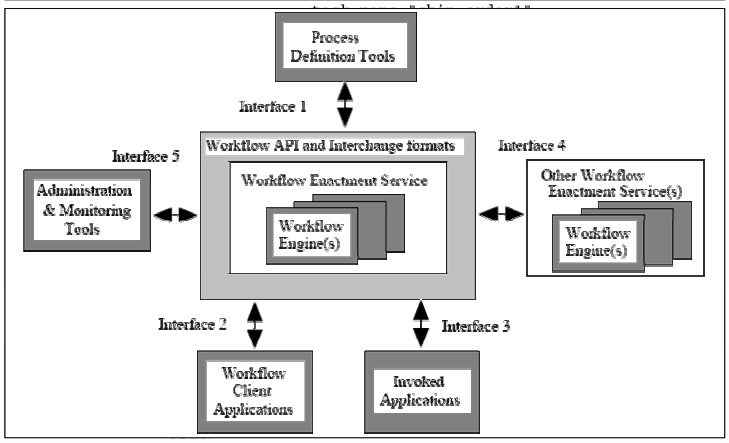
A ideia principal é a de criar uma plataforma para especificação de workflow em que os intervenientes serão Web Services remotos. Para tal, é necessário começar por criar uma linguagem para especificação de workflows, analisar algumas das linguagens mais representativas e estabelecer um estudo comparativo, calcular um denominador comum e especificar uma linguagem de anotação para esse denominador. Depois pretende-se criar um processador que a partir de uma destas especificações gere um motor de gestão do Workflow especificado.

XATA2005
XML: Aplicações e Tecnologias Associadas

Desenho de um Sistema de Workflow baseado em XML Web Services

Ricardo Manuel Meira Ferrão Luis, Universidade do Minho

- Criar de uma plataforma para especificação de workflow em que os intervenientes serão Web Services remotos
- Criar de uma linguagem para especificação de workflows
- Analisar de algumas das linguagens mais representativas e estabelecer um estudo comparativo (XPDL, ebXML, BizTalk, XRL)
- Calcular um denominador comum e especificar uma linguagem de anotação para esse denominador
- Criar um processador que a partir de uma destas especificações gere um motor de gestão do Workflow especificado



Ricardo Manuel Meira Ferrão Luis < rml@net.sapo.pt > Universidade do Minho

$CD - ROM = Template_cdrom + Dados$

Sergio Ferreira — Inst. Pol. Cávado e Ave

Resumo

Para as entidades que necessitam de publicar periodicamente, em papel ou em qualquer formato digital (web, CD-ROM, etc.), e cujo aspecto da edição se revela uma mais valia, necessitam de ter grandes meios de produção, humanos e técnicos, no sentido de garantir resultados. Por outro lado, se não existe por base um suporte técnico de automatização dos processos, normalmente originam um importante desgaste nos recursos envolvidos, não só humanos como no tempo.

É licito questionar então até que ponto é possível automatizar a produção de soluções desta natureza (muitas vezes mal apelidado por multimédia), com o recurso a recentes inovações tecnologias de informação, nomeadamente linguagens descritivas (tipo Markup Languages) e tecnologias de desenvolvimento de interfaces adaptativas (tipo Flash)?

Até que ponto poderá será possível a um utilizador ter capacidade e autonomia de escolha do layout associado aos dados que pretenda publicar, seja ela para a web ou não?

O objectivo deste trabalho assenta na análise de um caso de estudo que integra a tecnologia Flash, actualmente um, praticamente aceite, padrão profissional destinado a produzir experiências de grande impacto, com linguagens descritivas, neste caso específico o XML. A arquitectura base deste projecto, numa analogia a um belo quadro de pintura, assenta na presença de três níveis de informação complementares: o quadro (template em Flash), o desenho (dados e Meta-dados) e as cores (CSS + Actionscript), de forma a garantir a separação da estrutura lógica, da aparência final do trabalho.

Uma vez definidos os templates, caberá ao utilizador apenas escolher aquele que entende mais adequado e fornecer os dados que quer ver representados.

A natureza do título desta publicação assenta no caso de estudo comum associado à produção de um CD-ROM. Para o produzir-mos a partir de uma publicação web, deverá ser necessário somente a aplicação de um novo template, após a decisão sobre quais dados representar: $CD - ROM = Template_cdrom + Dados$

XATA 2005

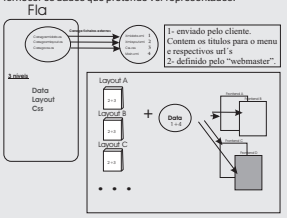
XML : Aplicações e Tecnologias Associadas

CD-ROM = Template_cdrom + Dados

Sérgio Ferreira
Instituto Politécnico do Cávado e do Ave

O objectivo deste trabalho foca o estudo da integração da tecnologia Flash, actualmente um, praticamente aceite, padrão profissional destinado a produzir apresentações de grande impacto, com linguagens de anotação, neste caso específico o *XML*. A arquitectura base deste trabalho, numa analogia a um belo quadro de pintura, assenta na presença de três níveis de informação complementares: o quadro (*Template* em *Flash*), o desenho (dados e Meta-dados) e as cores (*CSS + Actionscript*), de forma a garantir a separação da estrutura lógica do trabalho relativamente à sua apresentação.

Uma vez definidos os *templates*, caberá ao utilizador "apenas" escolher aquele que entende ser mais adequado e fornecer os dados que pretende ver representados.

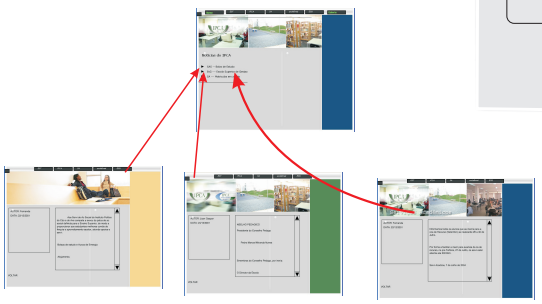


Caso de estudo IPCA

Temos a página inicial referente ao IPCA, com o respectivo menu superior com *links* para as respectivas páginas dos órgãos do Instituto. Na zona central surgem tópicos de notícias associadas ao respectivo órgão.

Ao clicarmos num tópico somos redireccionados para a página correspondente ao órgão a que se refere e ao respectivo conteúdo.

O *template* que surge tem por base as informações (notícias). Ou seja se colocarmos em vez de uma, duas notícias referentes ao SAS, continuará nos dois casos a surgir os *templates* do SAS, sem qualquer intervenção do utilizador.



Noticia1.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<NOTICIAid="1" publicacion="22/12/2001" orgao="sas">
<AUTOR>Fernanda</AUTOR>
<CORPO>
```

Aos Serviços de Acção Social do Instituto Politécnico do Cávado e do Ave compete a execução da política de acção social definida para o Ensino Superior, de modo a proporcionar aos estudantes melhores condições de frequência e aproveitamento escolar, através de apoios e serviços:

Bolsas de estudo e Auxílios de Emergência
Alojamento;
Alimentação em cantinas e bares;

```
</CORPO>
</NOTICIA>
```

Formalizing Markup Languages for User Interface

Luis G. M. Ferreira — Inst. Pol. Cávado e Ave

Resumo

Este trabalho tem como principal objectivo a aplicação de métodos formais na especificação da camada de apresentação das aplicações informáticas. Embora o contexto se cruze com a essência da HCI (Human Computer Interaction), este projecto foca essencialmente a forma como os métodos formais podem ser utilizados para especificar interfaces com o utilizador (user interface) descritos via linguagens de anotação (markup languages).

É analisado o estado da arte nas linguagens de anotação referentes a interfaces com o utilizador e é formalmente especificado o UIML - Interface Markup Language. Como caso de estudo, o componente gráfico "tabela", associado a serviços OLAP, é formalmente especificado em VDM-SL e animado com interfaces descritas em UIML.

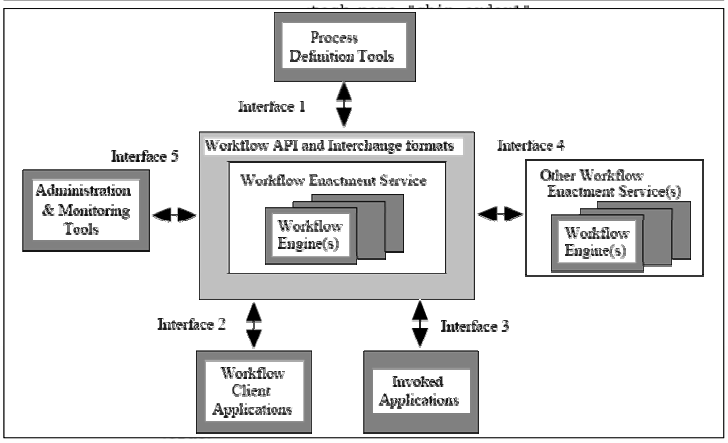
O resultado deste trabalho deverá ser entendido como um contributo para a construção de uma biblioteca de componentes visuais no sentido de suportar reutilização e composição de componentes.

XATA2005
XML: Aplicações e Tecnologias Associadas

Desenho de um Sistema de Workflow baseado em XML Web Services

Ricardo Manuel Meira Ferrão Luis, Universidade do Minho

- Criar de uma plataforma para especificação de workflow em que os intervenientes serão Web Services remotos
- Criar de uma linguagem para especificação de workflows
- Analisar de algumas das linguagens mais representativas e estabelecer um estudo comparativo (XPDL, ebXML, BizTalk, XRL)
- Calcular um denominador comum e especificar uma linguagem de anotação para esse denominador
- Criar um processador que a partir de uma destas especificações gere um motor de gestão do Workflow especificado



Ricardo Manuel Meira Ferrão Luis < rml@net.sapo.pt > Universidade do Minho

CodeLayer – Gerador de Código baseado em XML/XSL

Telmo Fonseca — FEUP

Resumo

O CodeLayer, desenvolvido no âmbito da cadeira de Processamento Estruturado de Documentos do Mestrado de Engenharia Informática da Faculdade de Engenharia da Universidade do Porto, tem como objectivo a produção automática de código em PHP ou Java para as camadas DataAccess e DataBusiness baseado na arquitectura n-tier.

Após uma ligação a uma base de dados (Oracle ou SQLServer) via ADO e recolhida a informação das tabelas e vistas do Schema em questão, é produzida uma imagem em XML das mesmas. Com a manipulação deste XML e aplicando o XSL para a camada correspondente a gerar, é produzido o ficheiro pretendido.

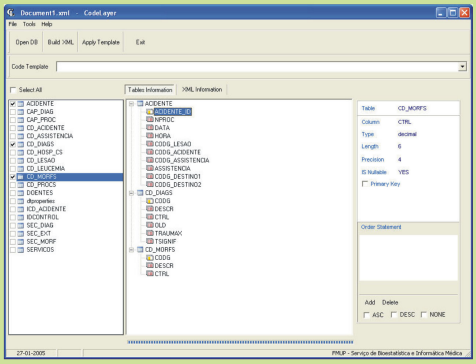
Os interfaces do CodeLayer foram desenvolvidos em Delphi e este está actualmente a ser utilizado pela equipa de desenvolvimento de software médico do Serviço de Bioestatística e Informática Médica da Faculdade de Medicina da Universidade do Porto.

XATA2005

XML: Aplicações e Tecnologias Associadas

CodeLayer

Telmo Fonseca , FEUP



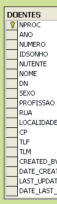
Selecção e configuração das Tabelas

Gerador automático de código com base em XML / XSL

Com base na selecção de objectos da base de dados, é gerado um XML correspondente. É permitida a alteração de chaves primárias para efeitos de geração de código e especificação de uma condição de ordenação.

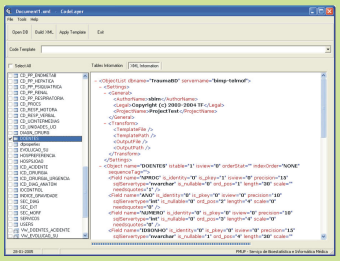
Após a selecção de um template xsl, é gerado código correspondente à *Layer* pretendida: Procedimentos; Camada de Acesso ou Camada de Negócio.

O CodeLayer está preparado para ligações a SQLServer e Oracle e geração de código PHP ou Java.



Tabela

CodeLayer



XML gerado

Aplicação de XSL

Documento PHP gerado

Aplicação de XSL

Documento Java gerado

Exemplo de transformação

Universidade do Porto
FEUP Faculdade de Engenharia
Faculdade de Engenharia - Universidade do Porto

Contacto: Telmo Fonseca
telmof@med.up.pt
http://users.med.up.pt/telmof

Serviço de Bioestatística e Informática Médica

XATA 2005 10 e 11 de Fevereiro de 2005, Casa da Torre, Vila Verde, Braga

Recuperação de sistemas legados usando XML

João A. Ferreira — Dep. Informática, U. Minho

Resumo

Este projecto insere-se na dissertação da tese de Mestrado em Informática tendo como principal objectivo estudar o "estado de arte"da problemática XML e bases de dados relacionais.

Partindo da actual panorâmica técnico-económico pretende-se demonstrar que o XML e tecnologias associadas podem ser usadas para a recuperação de sistemas legados (legacy systems). Para atingir uma interoperabilidade entre sistemas legados relacionais e tecnologias XML/Web surgem-nos duas problemáticas de fundo: publicar dados relacionais em forma de XML; guardar dados XML em SGBD relacionais.

Numa primeira parte do nosso projecto iremos abordar a problemática do mapeamento relacional em XML, sobretudo devido às diferenças existentes entre os dois modelos de dados: as relações são normalizadas; o XML é hierarquico e arbóreo. Iremos estudar vários estudos e soluções apresentados como o Silkroute, XPeranto, Rolex,Agora, Mars, bem como as soluções apresentadas pelos SGBD comerciais como o SQL Server SQLXML, DB2 XML Extender e o Oracle XML DB.

Na segunda parte, será estudada a problemática inversa: passar do XML para relações. Novamente, fazemos uma incursão pelo estado de arte, estudando diversas propostas e algoritmos como o Stored, Edge, Monet, XRel, XParent e abordagens que têm em conta um XML Schema bem definido.

XATA2005
XML: Aplicações e Tecnologias Associadas

Recuperação de sistemas legados usando XML

João A. Ferreira – Universidade do Minho (Braga)

Objectivos:

- Recuperar sistemas legados (SGBD) usando XML
- Estudar processos de mapeamento Relacional-XML
- Estudar processos de transformação XML-Relacional

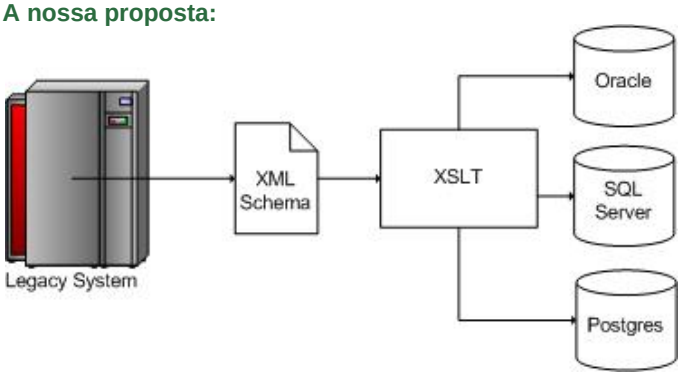
Relacional - XML

- XPeranto
- Silkroute
- Rolex
- Agora
- Mars

XML - Relacional

- Stored
- Edge
- XRel
- XParent
- XML Schema

A nossa proposta:



Mais detalhes:

www.progmat.com/Default.aspx?tabid=35

joao.ferreira@progmat.com

Author Index