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.

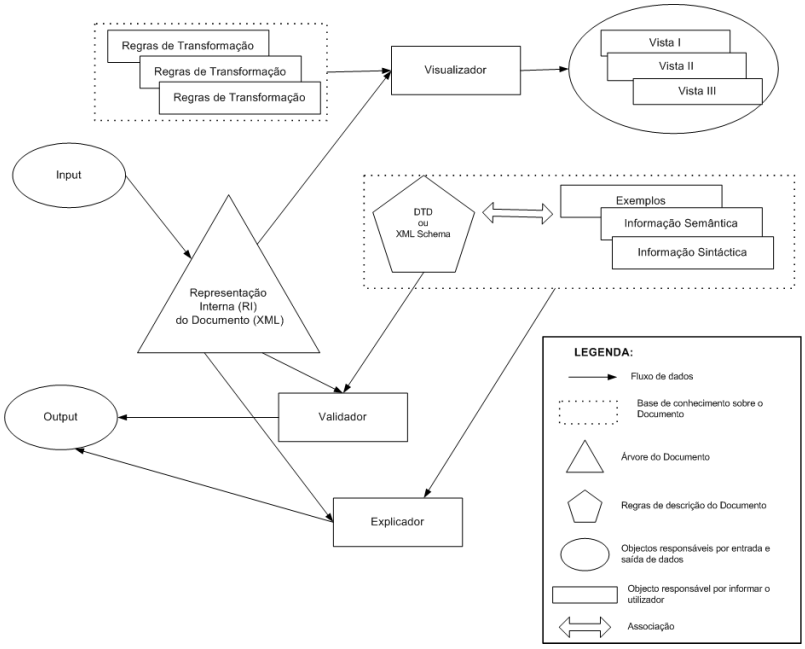

Figura 1. Arquitectura do PcWDa

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.

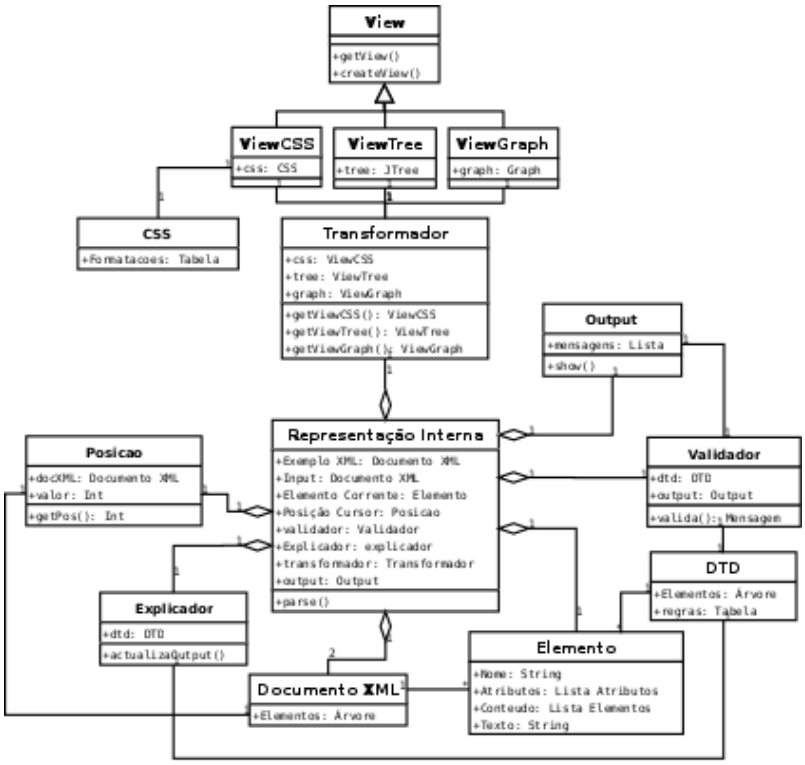

Figura 2. Diagrama de Classes do PcWDa

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;

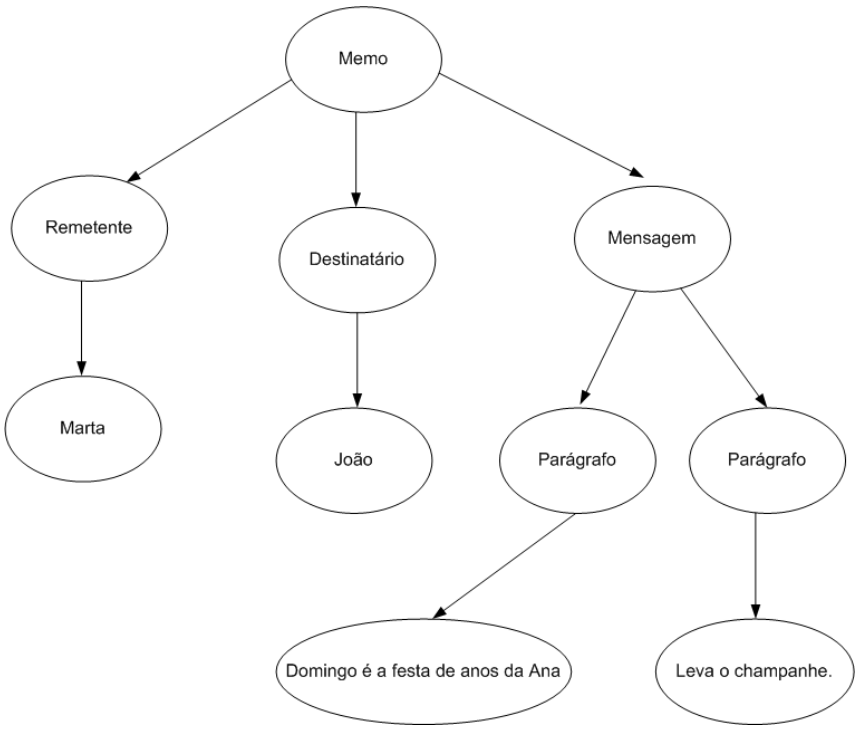

Figura 3. Estrutura em árvore do Memorando XML

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`,

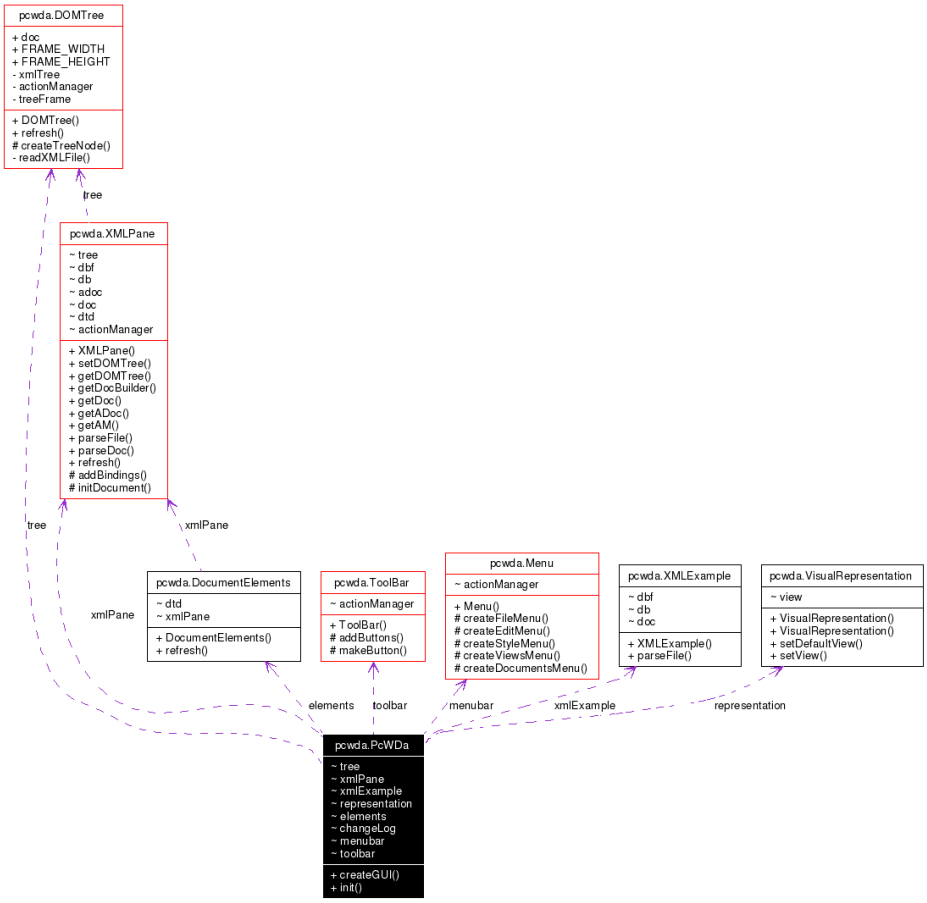

Figura 4. Diagrama de classes Java.

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;
}
```