INES: Ambiente para Construção Assistida de Editores Estruturados baseados em SGML

Alda Reis Lopes José Carlos Ramalho
Pedro Rangel Henriques

arl@ci.uminho.pt,{jcr,prh}@di.uminho.pt
grupo de Especificação e Processamento de Linguagens
Departamento de Informática
Universidade do Minho
Braga – Portugal

Resumo

Este documento descreve a especificação e implementação do ambiente INES. A premissa do projecto foi a aplicação de técnicas e ferramentas tradicionalmente usadas na compilação e no universo das linguagens de programação ao processamento de documentos. Assim, surgiu INES, um ambiente para construção assistida de editores estruturados baseados em SGML.

Para responder a todo um novo conjunto de necessidades, os documentos têm que estar disponiveis em vários formatos e plataformas. Para que isto seja viável, é necessário que as várias formas de um documento sejam geradas a partir de uma mesma representação inicial. Em 1987, um comité internacional criou um standard para a escrita de documentos que foi designado por SGML —*Standard Generalized Markup Language*—, ficando a ser o standard ISO 8879.

Actualmente, há muitos editores de SGML comercialmente disponíveis. Uns foram desenvolvidos de raiz; a outros, que já existiam como processadores de texto, foi-lhes acoplado um módulo com a funcionalidade do SGML. Todos estes editores correm, em *batch*, o processo de validação do documento em edição; a validação é despelotada após a edição por decisão do utilizador.

A nossa ferramenta é um gerador de editores que explora o paradigma de *parsing* e validação incremental, quer no gerador, quer nos editores gerados. O gerador de editores aceita a sintaxe do SGML para a especificação de um determinado tipo de documento (DTD) e gera um editor, estruturado e dirigido pela sintaxe, para os documentos desse novo tipo.

Estes editores (gerados pelo INES) têm algumas vantagens inerentes ao paradigma seguido na sua geração: a validação do texto introduzido é automática e imediata; o utilizador não se perde com os detalhes sintácticos do documento, podendo concentrar-se no seu conteúdo.

Abstract

This paper describes the specification and implementation of the development environment INES. The application of programming language technics and tools to document processing was the motivation for this project. INES was the result of that goal pursuit: a development environment for assisted construction of SGML based syntax-directed editors.
New needs are rising in document processing world. To be able to cope with these new demands documents should be available in several formats and platforms. The satisfaction of this requirement will only be feasible if all the different formats of a given document were generated automatically from the same representation, that should be a standard. In 1987, SGML was introduced as the standard for document production.
To join this standard, the use of a dedicated SGML editor is deeply advised. At present there are plenty of those editors commercially available. Some are simply new products while others are extensions to existing word processors. Most of them run the validation process in batch mode upon user decision.
Our environment, INES, is an editor generator that exploits the incremental parsing and validation paradigm. Both the generator and the generated editors follow this principle.
The generator accepts the SGML syntax as the specification of a document type definition (DTD) and generates a syntax directed editor for that type of documents. This approach offer some advantages: the structure of a new document is verified at edit time; the user only has to worry about content, not about syntax; the same environment is used both for creating a new editor and for producing new documents.

1 Introdução

Coisas como a acessibilidade (cada vez maior) do utilizador comum à Internet, a produção e disponibilização de informação em cd-rom, e a publicação electrónica, vieram dar uma nova dimensão ao processamento documental.

Para responder a todo este novo conjunto de requisitos, os documentos têm que estar disponíveis em vários formatos e plataformas. Para que isso seja viável, é necessário que as várias formas de um documento possam ser obtidas a partir de uma mesma representação inicial, representação essa que terá obviamente de ser *bastante rica* para satisfazer as várias utilizações pretendidas.

Face a esta constatação, formou-se em 1987 um comité para criar um standard para a escrita de documentos. O sistema de descrição a ser adoptado deveria possuir como características fundamentais: grande poder expressivo, para permitir uma aplicação generalizada; independência em relação a plataformas de software e hardware. Como resultado final, esse comité criou o SGML [Gol90] —*Standard Generalized Markup Language*— o standard ISO 8879.

O SGML surgiu, não como uma nova linguagem para escrever uma determinada classe de documentos (como o é, p.ex., o LaTeX, ou o HTML), mas sim como uma

meta-linguagem com capacidades de definição estrutural, isto é, possibilidade de descrever a constituição dos documentos de uma família.

A noção de *tipo de documento* foi, precisamente, a grande novidade introduzida — assim, o SGML começou a ser usado para definir DTDs [MA96] —*Document Type Definitions*— que estabelecem a estrutura de todos os documentos desse tipo e, mais ainda, a sintaxe que deve ser usada para editar documentos do tipo em causa.

Uma vez que a ideia base, que presidiu à definição do SGML, foi a criação duma linguagem que permitisse a *marcação de textos*, conclui-se que, usando DTDs, é possível originar *linguagens de anotação descritiva* que servirão, então, para produzir os documentos concretos do tipo em causa. HTML é, sem dúvida, o exemplo mais conhecido de linguagem de anotação descritiva definida através de um DTD escrito em SGML; nesse caso, a linguagem permite construir as páginas de documentos hipermédia que serão distribuídos pela Internet e manipulados por *browsers* apropriados.

O quadro que acabamos de descrever colocou-nos o desafio de estudar uma forma sistemática e eficaz de *processar documentos*. Rapidamente verificamos que a resolução dessa questão envolve:

- a escolha de um formalismo para descrever os documentos;
- a definição duma representação interna (em memória) adequada ao arquivo dos documentos e à sua transformação;
- a eleição dum ambiente de trabalho onde facilmente se possam descrever e implementar as várias transfomações pretendidas.

Foi com este cenário como pano de fundo que avançamos com a proposta do projecto DAVID —aprovado pela JNICT em 1995— dedicado a investigar o processamento de documentos, sugerindo-se a reutilização de métodos e ferramentas bem conhecidos no domínio do processamento formal de linguagens (de programação) integradas com um ambiente de programação algébrica baseada em modelos que nos é familiar (que tem vindo a mostrar-se altamente adequado ao desenvolvimento de aplicações).

No contexto do projecto DAVID temos ensaiado várias abordagens ao problema. Uma delas assume as linguagens definidas em SGML como notação para descrever os documentos e recorre a gramáticas de atributos para definir a representação interna e as operações de transformação dos documentos.

Neste artigo vamos apresentar uma das ferramentas que concebemos e estamos a desenvolver nesse âmbito: *o ambiente para construção assistida de editores estruturados baseados em SGML*, designado por INES.

A edição de textos tradicional é não assistida. Como a anotação de documentos em SGML torna a escrita um pouco pesada e não trivial, há toda a conveniência em optar pelo paradigma da edição assistida para possibilitar uma ampla aceitação desta norma e facilitar a sua disseminação. Dentro da edição assistida a opção mais comum é baseada em menus de opções com forte utilização do rato: FrameMaker[Inc96], SGML Author[Cor96], PSGML[Sta]. Neste trabalho, que vem no seguimento de trabalhos

anteriores [RAH95, RAH96], optamos por uma outra alternativa baseada na edição dirigida pela sintaxe.

Um editor estruturado é um editor de texto especializado para uma dada linguagem e dirigido pela sintaxe dessa linguagem. Quer isso dizer que o editor vai conduzindo o utilizador, de acordo com a gramática da linguagem, preenchendo automaticamente todas as partes fixas da frase (inserindo as chamadas palavras-reservadas) e indicando que alternativas é que estão disponíveis em cada momento. Deste modo, o utilizador não se perde com detalhes sintácticos do documento —porque alguém já pensou nisso e construíu o editor de maneira a que ele só admite a estrutura correcta— podendo concentrar-se no conteúdo (semântica) do que está a escrever.

Como será detalhado na secção 3, o ambiente INES serve para gerar automaticamente editores desse género. Para isso, a ferramenta geradora precisa de conhecer o tipo de documentos específico, para o qual se pretende o editor, bastando fornecer-lhe o DTD dessa classe de documentos.

Com vista a facilitar a interacção com essa ferramenta, decidiu-se que ela seria, também, construída segundo o mesmo paradigma da edição guiada pela sintaxe. Assim, o gerador INES está preparado para aceitar a introdução de um DTD (que de acordo com o que se disse acima, é a informação que lhe tem de ser apresentada para poder produzir o editor final desejado), auxiliando o utilizador na escrita dessa especificação.

Na secção 4 explicar-se-á a forma como foi implementado o referido gerador INES, através do uso duma ferramenta bem conhecida na área da compilação designada por Synthesizer Generator [TR81, RT89a]. Essa explanação, além do mais, ajudará a perceber o género de editores que serão obtidos.

O sistema Synthesizer Generator não fornece somente um editor dirigido pela sintaxe. Produz, também, um compilador incremental que será activado automaticamente para processar (reconhecer e traduzir) a frase em simultâneo com a sua edição — deste modo, o texto que está a ser introduzido é logo analisado e qualquer erro, léxico/sintático ou semântico, detectado será assinalado imediatamente. Após uma alteração no texto-fonte, o compilador só recompila o estritamente necessário para manter o seu significado correcto (daí a designação de *incremental*).

Esta característica dos editores/compiladores gerados pelo Synthesizer Generator é viável por esse sistema assentar o seu princípio de funcionamento no paradigma da *compilação orientada pela semântica* baseando a especificação do processador numa *gramática de atributos transformacional*, um pouco ao contrário da maioria dos sistemas SGML existentes, que utilizam na sua implementação o *parser* SGMLS[Cov], ou a biblioteca SP[Cla], que foram desenvolvidos de acordo com o paradigma da *tradução dirigida pela sintaxe*, o que faz com que certas funcionalidades desejadas sejam dificeis de implementar: por exemplo, a maior parte destas ferramentas não tem capacidade de disponibilizar a forma estrutural do documento depois de o processar (no fim não existe uma representação interna do mesmo).

Por isso, dedicam-se algumas linhas a rever os conceitos básicos associados às gramáticas de atributos e ao cálculo incremental na subsecção 2.2, depois de se ter passado em revisão as noções essenciais relativas à anotação de documentos e ao standard SGML (subsecção 2.1).

2 Revisão dos Conhecimentos Fundamentais

Esta secção, dividida em duas partes, destina-se a rever os conhecimentos que nos pareceram fundamentais para se entender o contexto em que surge o ambiente INES, a sua filosofia de funcionamento e a estratégia que seguimos para o implementar.

Primeiro, na subsecção 2.1, resumimos as noções associadas à anotação de documentos e ao standard SGML.

De seguida, na subsecção 2.2, apresentam-se os conceitos principais relativos às gramáticas de atributos e ao paradigma da tradução orientada pela semântica. Abordam-se, ainda, as ideias essenciais referentes ao cálculo de atributos incremental.

2.1 Anotação de Documentos: SGML

Historicamente, o termo inglês *markup*, em português *anotação*, foi usado para descrever as anotações ou outras marcas que eram colocadas nos textos para instruir os compositores e tipógrafos de como estes deveriam ser impressos e/ou compostos. Por exemplo, palavras sublinhadas com linha ondulada deveriam ser impressas em *boldface* (carregado). Com a automação das tarefas de formatação e impressão o termo começou a ser utilizado num sentido mais amplo, cobrindo todas as espécies de códigos de anotação inseridos em textos electrónicos para dirigir a sua formatação, impressão ou outro tipo de processamento.

Generalizando, podemos definir anotação de um texto como um meio de tornar explícita uma interpretação desse texto. Como exemplo mais banal de anotação podemos olhar para os sinais de pontuação que induzem uma determinada interpretação do texto em que estão inseridos.

A ideia de etiquetar um documento surge naturalmente quando se pensa num documento específico. Por exemplo, uma carta comercial, é um documento com uma determinada estrutura intrínseca, e quase todas as empresas as escrevem de acordo com essa estrutura (a importância do standard). Ora, para o computador será muito mais fácil processar este tipo de documento se ele estiver "marcado": o que é o cabeçalho, a assinatura, a data, ..., pois caso contrário teria que andar pelo texto fora a adivinhar quem é o quê.

Normalmente, os sistemas de processamento de texto requerem que o texto natural seja enriquecido com informação adicional. Esta informação adicional, que tem a forma de etiquetas, serve duas linhas de acção:

(a) Divide o documento nas suas componentes lógicas; e

(b) Especifica qual ou quais as funções de processamento que devem ser aplicadas naquele componente.

Ou seja, formatação (aspectos visuais relacionados com a apresentação) e conteúdo aparecem misturados ao longo do documento; por exemplo, o título e o autor de um livro aparecem no meio de anotações que especificam qual a fonte e o tamanho

a utilizar para os caracteres, misturando assim uma característica estrutural com questões de representação.

Uma vez que, as anotações relativas ao formato (fonte, tamanho da fonte, estilo, ...) variam de produto para produto (alguns destes conjuntos de anotações são mesmo proprietários), a exportação de um sistema para outro torna-se muito difícil.

O SGML veio resolver este problema, isolando toda a informação visual, separando definitivamente os pormenores de formatação da forma (estrutura) e do conteúdo de um documento.

O SGML é uma meta-linguagem para definição de linguagens de anotação descritiva [Her94] (por oposição à anotação procedimental[1]). Assim, adoptando este tipo de anotação o utilizador apenas se preocupa com o conteúdo.

Visto desta maneira o SGML é uma linguagem para especificar gramáticas, tornando-se uma escolha natural para linguagem base do nosso sistema, uma vez que se pretendem aplicar técnicas e ferramentas da compilação que se baseiam em especificações gramaticais.

Um documento SGML compreende três partes:

- uma parte declarativa, onde se coloca informação que irá configurar/parametrizar o *parsing*; por exemplo: qual a tabela de caracteres que irá ser usada; qual o tamanho máximo dos identificadores; ...
- o DTD (*Document Type Definition*) [MA96], que contém a definição do tipo de documento. Entre outras funções o DTD vai:
  - especificar quais as etiquetas disponíveis para anotar o documento deste tipo (cada elemento estrutural do documento irá ser limitado por duas etiquetas, uma marcando o seu início e outra o seu fim).
  - especificar quais os atributos disponíveis e quais os necessários para cada elemento estrutural.
  - especificar a estrutura de cada elemento: quais os seus subelementos, onde pode aparecer texto e onde podem aparecer outras coisas como imagens.
- o texto do documento, anotado com o conjunto de etiquetas definido na secção anterior.

A seguir apresenta-se um exemplo de um documento SGML contendo uma mensagem de *email*:

```
<!DOCTYPE email [
<!ELEMENT email - - (cabec, mensagem)>
<!ELEMENT cabec - - (de, data, cc?, assunto?)>
<!ELEMENT (de, data, cc, assunto) - - (#PCDATA)>
<!ELEMENT mensagem - - (#PCDATA)> ]>
```

[1] Normalmente, a anotação procedimental aparece associada à formatação visual de um documento, por exemplo, o comando *vspace* em LaTeX recebe um argumento numérico e como resultado produz um espaço em branco vertical no documento.

```
<email>
  <cabec>
    <de>Jose Carlos Ramalho</de>
    <data>21 de Abril de 1997</data>
    <assunto>21 de Abril de 1997</assunto>
  </cabec>
  <mensagem>
    O artigo est'a quase pronto esperemos que siga a tempo...
  </mensagem>
</email>
```

2.2 Compilação com Gramáticas de Atributos

Durante muitos anos os compiladores foram especificados através de gramáticas tradutoras — *gramáticas independentes do contexto* a que se acrescenta uma *acção semântica* no fim de cada produção—, sendo então implementados segundo o paradigma da *tradução dirigida pela sintaxe*. De acordo com essa filosofia, todo o funcionamento do processador é controlado pelo *parser*: é este módulo que invoca o analisador léxico, quando necessita do próximo símbolo do texto-fonte para prosseguir a análise; é, também, ele quem activa uma *acção semântica* logo que reconhece a regra de derivação respectiva. Nesta estratégia nunca se chega a criar uma representação completa do significado do texto-fonte. A *acção semântica* associada a uma produção é responsável pela análise semântica e pela geração de código locais a essa produção.

Com o aparecimento das gramáticas de atributos [Knu68] —extensão formal à *gramática independente do contexto*, que permite especificar a sintaxe e a semântica duma linguagem— e a evolução (em capacidade de memória central e em velocidade de processamento) dos computadores, passou-se a seguir o paradigma da *tradução dirigida pela semântica* na implementação dos compiladores. À luz desta nova filosofia, a tarefa de análise léxica continua a ser executada a pedido do *parser*, mas a análise semântica e a tradução passam a ser independentes entre si e da análise sintáctica, sendo executadas globalmente e encadeadas umas nas outras. Durante a análise sintáctica é construída, em memória, uma árvore de derivação, que descreve a estrutura do texto fonte. A árvore é, então, passada ao analisador semântico que a *decora*, calculando o valor dos atributos associados a cada nodo, e a *valida*, verificando se as condições contextuais são todas respeitadas. Por fim, essa árvore de sintaxe decorada é passada ao tradutor que a transforma na representação final pretendida.

Desta forma, a árvore de sintaxe decorada, construída e mantida em memória, constitui uma *representação interna do significado* do texto a ser processado.

2.2.1 Gramática de Atributos

Para clarificar a caracterização, que acabamos de fazer, sobre *tradução dirigida pela semântica*, apresenta-se uma definição muito sucinta de gramática de atributos. Para mais detalhes sobre a descrição deste formalismo, a sua utilização e os problemas e soluções relativos ao cálculo dos atributos, sugere-se a leitura de [Hen92], entre muitas outras obras disponíveis [DJL88, DJ90, AM91].

Segundo o seu criador, Knuth [Knu68], a gramática de atributos deve servir para permitir a definição local (sem recurso a variáveis globais) do significado de cada símbolo, num estilo declarativo.
Símbolos Terminais têm atributos intrínsecos (que descrevem a informação léxica que lhes é associada) e os símbolos Não-terminais são associados a atributos genéricos; informação semântica pode, assim, ser sintetizada pela árvore de sintaxe acima (das folhas para a raíz), mas também pode ser herdada pela árvore abaixo (da raíz para as folhas, ou entre nodos irmãos), permitindo referências explícitas a dependências contextuais.
Formalmente, uma gramática de atributos AG é um tuple $AG =< G, A, R, C >$, onde:

- G é uma GIC, formada pelo tuplo $G =< T, N, S, P >$, com:
  - T é o conjunto dos símbolos terminais (o alfabeto)
  - N é o conjunto dos símbolos não-terminais
  - S é o *símbolo inicial*, ou o *axioma da gramática*: $S \in N$
  - P é o conjunto das *produções*, ou regras de derivação, cada uma da forma

    $$A \to \delta,\ A \in N\ \wedge\ \delta \in (T \cup N)^*$$

    que se representa habitualmente como

    $$X_0 \to X_1\ X_2\ \cdots\ X_n$$

- A é a união dos conjuntos $A(X)$ para cada $X \in (T \cup N)$, e denota o conjunto de todos os atributos (cada um tem um nome e um tipo); o valor dos atributos de um símbolo terminal (*atributos intrínsecos*) não precisa de ser calculado;
- R é a união dos R_p, o conjunto de todas as *regras de cálculo dos atributos* dos atributos associados aos símbolos de cada produção $p \in P$
- C é a união dos C_p, o conjunto das *condições contextuais* relativas aos valores dos atributos dos símbolos de cada produção $p \in P$

Dada uma frase de $\mathcal{L}_{\mathcal{AG}}$ —a linguagem gerada pela gramática de atributos AG— seja AST a correspondente *árvore de sintaxe abstracta*. Cada nodo dessa árvore está associado a um símbolo da gramática e ao identificador da produção que foi usada para derivar esse símbolo. Cada nodo será, então, enriquecido com os atributos (quer *herdados*, quer *sintetizados*) do respectivo símbolo. Uma *árvore de sintaxe abstracta decorada* (DAST) é a AST inicial após o *processo de cálculo dos atributos*, que deixa todas as ocorrências de atributos ligadas ao seu valor concreto e contextual.
Para que essa árvore seja, de facto, uma DAST válida é necessário que todas as *condições de contexto*, relacionadas com a produção que etiqueta cada nodo, sejam satisfeitas.
Para especificar a tradução, é usual acrescentar à gramática de atributos AG, acima definida, um novo elemento $Trad$ que representa a união dos conjuntos de *regras*

de transformação, $Trad_p$, associados a cada produção. Essas regras são acções que produzem o resultado final manipulando os valores dos atributos dos símbolos que intervêm na produção em causa.

2.2.2 Cálculo de Atributos

Ao basear a implementação dos compiladores em gramáticas de atributos, seguindo o já mencionado princípio da *tradução dirigida pela semântica*, assume-se que a análise semântica (realizada separadamente, após a conclusão da análise sintáctica) é construída à custa de duas grandes tarefas, aplicadas a cada nodo da DAST:

- o cálculo do valor das ocorrências dos atributos
- a validação das condições contextuais associadas a cada nodo da DAST

Essas duas tarefas não são complicadas de realizar uma vez que o cálculo se faz à custa da aplicação das funções descritas explicitamente na gramática de atributos sob a forma de regras de cálculo (constituindo o citado conjunto R), ou dos predicados também explicitados na forma de condições de contexto (agrupados no referido conjunto C).

Porém essas funções calculam o valor de atributos à custa do valor de outros atributos, locais à produção, e os predicados estabelecem restrições aos valores desse atributos. Tal impõe dependências entre os atributos, dependências essas que têm de ser respeitadas na ordem de cálculo para garantir que sempre que uma função vai ser invocada os seus argumentos já estão calculados.

A determinação da ordem de cálculo e de verificação das condições é uma tarefa complexa, que é executada pelo sistema gerador do compilador a partir da gramática de atributos (cf. [Hen92]). Além disso e dado o número de ocorrências de atributos que podem surgir numa DAST (muitos deles tomando valores em tipos estruturados complicados), o processo de cálculo consome grande parte do tempo total da compilação.

Pelo que acaba de ser dito, surgiu a ideia de proceder ao *cálculo incremental dos atributos* no contexto de compiladores que são activados em simultâneo com editores estruturados (cf. [RTD83, Jal85]). Nesses ambientes, o compilador tem de ser reactivado (para voltar a analisar o texto e a proceder novamente à tradução) cada vez que se detecta uma alteração nesse texto-fonte.

O cálculo incremental (seguindo a ideia subjacente às *folhas de cálculo electrónicas*) tem por objectivo minimizar o esforço dispendido na análise semântica para que o processo de edição/compilação seja o mais rápido possível. A ideia é manter-se, em tempo de compilação, a informação sobre as dependências entre os atributos de modo a que, após uma alteração qualquer na AST inicial, se identifiquem os atributos dos símbolos que foram alterados e aqueles que deles dependem, por forma a garantir que só serão calculados os valores dos atributos afectados, sendo conservados todos os restantes.

3 INES: apresentação

Apesar de ter sido um passo em frente na caótica produção documental, o SGML não é de todo acessível ao utilizador comum. Quando numa determinada linha administrativa se faz a opção de produzir toda a documentação em SGML algum cuidado deve ser colocado com a adaptação dos utilizadores a esta nova maneira de "pensar"e "produzir"documentos.

Foi esta a preocupação principal que deu origem ao INES. A constatação do que se passa com as páginas WWW e o HTML reforçou a importância e urgência do projecto. Não é preciso ser-se um utilizador muito experiente para conseguir produzir documentos para disponibilizar em WWW. Muita gente, nas mais variadas áreas, produz páginas em HTML; na maior parte das vezes, esses utilizadores não têm consciência de que aquilo que estão a fazer é escrever um documento SGML de acordo com um DTD (o DTD do HTML). É, justamente, essa transparência e simplicidade de utilização que nos interessou captar para o ambiente INES.

O sistema INES é um gerador de editores composto por 3 grandes módulos:

- um editor de DTDs dirigido pela sintaxe (este editor ajuda a criar um DTD para uma novo tipo de documentos);
- um reconhecedor sintático-semântico incremental que funciona em simultâneo com o editor;
- um transformador, que traduz a descrição do DTD que foi editada para a especificação dum novo editor, mas agora especilizado para o tipo de documentos que se quer criar.

A especificação produzida pelo módulo transformador é, posteriormente, usada para criar automaticamente, recorrendo ao Synthesizer Generator (cf. secção seguinte), o editor concreto.

O editor de DTDs do gerador INES, proporciona, a cada momento, ao autor dos documentos um leque de declarações alternativas (além da possibilidade de introdução de comentários), a saber: *Element* (elemento do DTD, que cumpre o papel de anotação), *Attribute* (atributos SGML[2]), *Entity* (entidades que se usam para introduzir abreviaturas e incluir no documento componentes externas), *Notation* (notações que associam formatos de componentes a ferramentas externas que as podem tratar) e *Marked-Section* (secções especiais onde o processamento do documentoi é restringido).

Por sua vez, cada declaração terá de ser descrita segundo uma estrutura gramatical própria, que é evidenciada pelo editor quando o autor selecciona essa declaração.

Na figura 1 mostra-se uma janela do editor de DTDs onde se ilustra, através de textos concretos, cada um destes tipos de declaração. No exemplo apresentado, há partes que são palavras-reservadas do SGML para descrição dum DTD (p.ex., `<!DOCTYPE` e `<!ELEMENT`) —esses símbolos são inseridos automaticamente e no ponto

[2]Entenda-se atributos dos elementos, ou etiquetas, definidos em SGML.

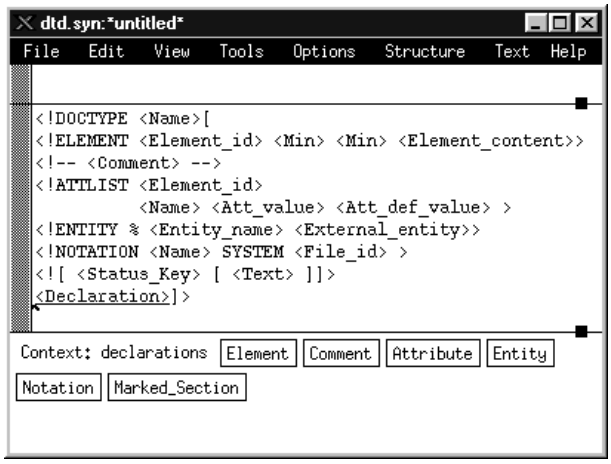

Figura 1: Editor INES: Janela de edição dum DTD

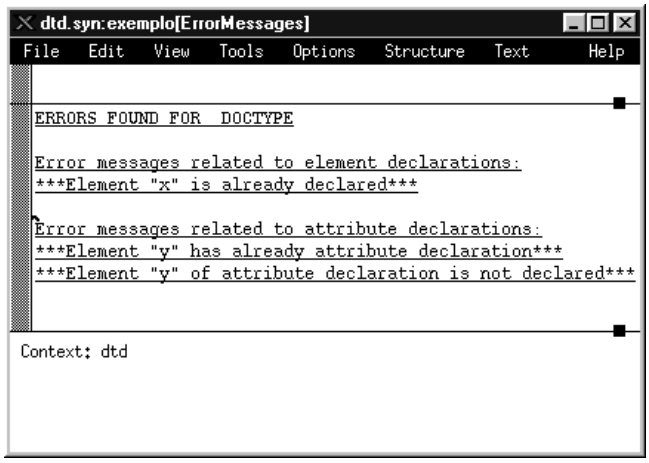

Figura 2: Editor INES: Janela de apresentação de mensagens de erro

certo pelo editor; as outras partes, que por convenção se vêem capitalizadas e entre '<' e '>', indicam os elementos variáveis dum DTD que deverão ser preenchidos pelo autor —esses símbolos são também colocados no écran pelo editor, mas serão substituidos logo que o autor seleccione a expansão desejada (cujas alternativas vão sendo indicadas, também automaticamente, na parte inferior da janela do editor, cf. figura 1).

Pelo facto da análise léxica e sintático-semântica decorrer em paralelo, o texto que vai sendo composto é imediatamente validado e todos os erros detectados são logo reportados. A figura 2 exemplifica a janela com as mensagens de erro que poderia ter surgido durante a escrita do DTD exemplo.

Atente-se, finalmente, na figura 3 que mostra o exemplo de um DTD concreto —para definir o que é o tipo de documentos "`carta`"— já totalmente editado.

Como se vê na figura 3, a janela de edição apresenta no cabeçalho o nome do programa executável que a dispara, "`dtd.syn`", e o nome do ficheiro onde é guardado

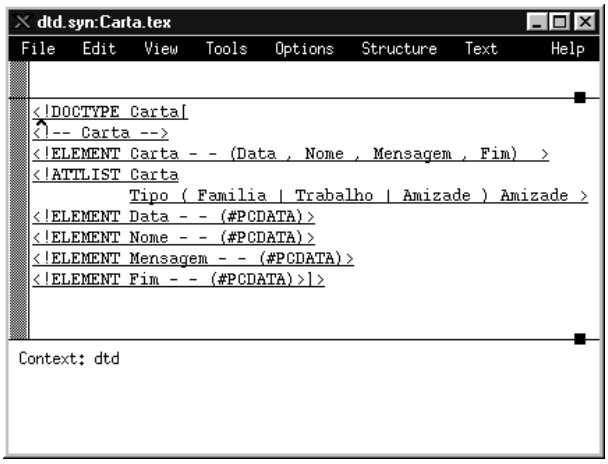

Figura 3: Editor INES: Exemplo dum DTD para o tipo de documentos "`Carta`"

o documento editado, "`Carta.tex`".
Como consequência da escrita dum DTD, o transformador INES constrói, incrementalmente ao longo da edição, um editor para o tipo de documentos definido.
Concretamente, após a introdução do DTD da referida figura 3, seria gerada a descrição necessária para se criar um editor estrutural dedicado à escrita de *cartas* (anotadas conforme o DTD estabelece). Com esse editor será, então, possível editar documentos como o que se exemplifica abaixo:

```
<Carta Tipo=Amizade>
   <Data> 96/07/31 </Data>
   <Nome> Ana Maria </Nome>
   <Mensagem> Desejo-te umas boas f'erias! </Mensagem>
   <Fim> Um abra,co! </Fim>
</Carta>
```

Note-se que, à semelhança do próprio editor do gerador INES, o novo editor apresenta as várias anotações que o texto deve (ou pode) conter, deixando a cargo do autor o seu preenchimento.

4 INES: implementação

Nas subsecções que se seguem, far-se-á a apresentação da ferramenta de software que sustentou a implementação do ambiente INES, e a explicação da estratégia usada.

4.1 Synthesizer Generator

O Synthesizer Generator [RT89a, RT89b] (SGen) é uma ferramenta que tem por objectivo implementar editores dirigidos pela sintaxe de uma dada linguagem. Estes são gerados a partir de uma especificação formal escrita na linguagem do SGen,

a Synthesizer Specification Language[3] (SSL). Os editores estruturados gerados pelo SGen trazem algumas vantagens ao desenvolvimento de programas, e poderão vir a ter um vasto domínio de aplicação —para uma discussão mais alargada sobre o assunto sugere-se a leitura de [Ram93, Roc92].

O SGen gera um editor estruturado e um compilador incremental para uma dada linguagem, partindo da especificação SSL. A SSL baseia-se, fundamentalmente, no conceito de gramática de atributos. Como tal, uma especificação compreende a análise léxica, sintáctica e semântica e geração de código. Para uma maior clareza e estruturação do desenvolvimento, estes componentes da especificação são construídos em diversos módulos:

- Módulo correspondente à análise léxica, onde se definem os lexemas.
- Módulo correspondente à análise sintáctica; neste módulo define-se a sintaxe concreta que especifica a linguagem real que o utilizador deve usar quando editar um texto.
- Módulo onde se especifica a sintaxe abstracta, i.e., o conjunto de produções segundo as quais os símbolos não-terminais são derivados, sem incluir as palavras-reservadas, ou outros símbolos que formam o chamado *açucar sintático* da linguagem concreta.
- Módulo onde se faz a correspondência entre os símbolos não-terminais da gramática concreta e da gramática abstracta.
- Módulo que inclui as regras de formatação, determinando a forma como os símbolos não-terminais são apresentados no editor (cria uma representação visual para a árvore de sintaxe abstracta). Para cada produção é declarada uma regra que especifica: as palavras-reservadas que devem surgir no texto; as zonas de introdução de informação, correspondentes aos não-terminais que têm de ser derivados; e o aspecto com que essas componentes devem aparecer na janela do editor.
- Módulos correspondentes à análise semântica: as declarações de atributos; a definição de funções auxiliares para o seu cálculo; e as condições contextuais a validar.
- Módulo correspondente à geração de código, onde se especifica o resultado final como sendo uma outra forma de visualizar a árvore de sintaxe abstracta que, internamente, representa o significado do texto-fonte.

4.2 O Gerador

A revisão de base apresentada na secção 2 e na subsecção anterior, e a descrição, na secção 3, da funcionalidade esperada para o INES, permitem-nos detalhar o desenvolvimento do sistema de forma concisa.

[3] a migrar, na versão mais recente, para Scheme.

A implementação do sistema INES compreende duas partes fundamentais:

- a especificação SSL do editor/reconhecedor de DTDs
- a especificação, também em SSL, do mecanismo de geração da descrição SSL do editor a gerar[4]

Assim, para construir o editor/gerador INES criaram-se os 7 módulos, típicos duma especificação SSL para o SGen, enumerados e descritos na subsecção anterior —os cinco primeiros dedicados ao editor/reconhecedor, o último descrevendo o gerador, e o sexto contendo atributos e regras de cálculo úteis para ambas as componentes.

Alguns atributos associados aos símbolos não-terminais visam a detecção de um conjunto de erros semânticos referentes às condições de contexto próprias da linguagem SGML para escrita de DTDs. Como foi já dito, a visualização das infracções a estas condições (erros semânticos) é feita numa janela dedicada, onde as mensagens são mantidos até que o utilizador os corrija. Essa funcionalidade é obtida à custa da introdução das regras apropriadas (que manipulam os referidos atributos) no módulo de *unparsing* (que define a formatação do texto na janela de edição).

Para além desses atributos para detecção de erros semânticos, são associados aos não-terminais outros atributos que irão conter toda a informação necessária para gerar a especificação SSL do novo editor. Esse resultado —o texto SSL gerado— é, uma vez mais, obtido como uma outra vista (construida fazendo uso dos atributos respectivos) da árvore de sintaxe abstracta.

Por outras palavras, o recurso aos atributos nesta fase da implementação prende-se, então, com a necessidade de criar a estrutura da linguagem do documento definido, já que a informação relacionada com todos os elementos do DTD é guardada em atributos. O processo de geração do código de cada módulo da especificação faz referência a esses atributos.

A seguir apresenta-se um exemplo onde é ilustrada a utilização de um dos atributos responsáveis pelas validações semânticas:

```
dtd : DocTypDef { /*Attributes related to element declarations errors*/
                  ...
                  local errorMesList errorStrE;
                  ...
                  /*Attribute equations for local attributes*/
                  errorStrE = ElemErrorStr(dElems);
                  ...
                };


/*This function returns a list of error messages.*/
errorMesList ElemErrorStr(elemSeq e)
 {
  with(e) (
           ElemSeqNull: ErrorMesListNull,
```

[4]Essa especificação será usada para produzir o editor do tipo de documentos definido pelo DTD introduzido pelo autor.

```
          ElemSeq(eI, es) :
                        ErrorMes("***Element", eI, "is already declared***")
                        :: ElemErrorStr(es),
         )
 };

/*Unparsing rules for example error messages*/

dtd : DocTypDef
      [ ErrorMessages ^ : "ERRORS FOUND FOR " ^ " DOCTYPE%n" ..
                          "%nError messages related to element declarations:
                          %n" errorStrE ... ];

/*List of error messages*/
errorMesList : ErrorMesListNull [ ErrorMessages ^ ::= ]
             | ErrorMesList [ ErrorMessages ^ ::= ^ "%n" ^ ];
```

Todos os detalhes relativos ao desenvolvimento da gramática de atributos em SSL para geração automática do sistema INES, com recurso ao SGen, encontram-se documentados no relatório [Lop96].

5 Conclusão e Trabalho Futuro

Ao longo deste artigo descrevemos a especificação e implementação dum ambiente para construção assistida de editores dirigidos pela sintaxe para documentos baseados em SGML, ao qual chamamos INES.

Esse trabalho tem vindo a ser realizado no âmbito do projecto DAVID[5] —*Manipulação e Especificação Algébrica de Documentos*— e teve como premissa a aplicação de técnicas e ferramentas tradicionalmente usadas na compilação e no universo das linguagens de programação ao processamento de documentos. Tem-se verificado, que as abordagens gramaticais ao processamento de documentos têm produzido óptimos resultados na standardização e transferência de documentos entre plataformas diferentes.

Resumindo, o INES apresenta como *front-end* um editor. Este editor aceita especificações válidas de DTDs. Depois de terminada a especificação de um DTD, o sistema gera uma especificação SSL; o sistema é, então, retro-alimentado com esta especificação, gerando um novo editor. Por fim, o novo editor fica disponível para o utilizador que nele pode criar os seus documentos.

O INES já se tornou bastante útil e está a ser aplicado para produzir documentação em SGML ou, às vezes, simplesmente para especificar DTDs. No entanto, algumas evoluções estão em curso:

- está a ser feito um estudo no sentido de tornar mais simples a sintaxe do editor de DTDs (simplificar a sintaxe do SGML).
- está em estudo uma possível versão do INES para XML[6].

[5] http://www.di.uminho.pt/ jcr/projectos/david/princ.html
[6] http://www.textuality.com/sgml-erb/WD-xml.html

- irá ser adicionada uma camada de especificação de estilo, provavelmente em DSSSL[7].
- está também a ser pensada uma camada funcional para a especificação de invariantes relacionados com a validação, não da estrutura (é o que fazem os parsers de SGML), mas sim do conteúdo, com o objectivo de garantir a qualidade da informação.

Agradecimentos

Os autores gostariam de agradecer em primeiro lugar aos revisores do 2º Simpósio Brasileiro de Linguagens de Programação.
O projecto DAVID decorre no âmbito do contrato nacional PBIC/C/TIT/2479/95, cabendo-nos agradecer à JNICT o financiamento disponibilizado.

Referências

[AM91] H. Alblas and B. Melichar, editors. *Attribute Grammars, Applications and Systems.* Czech Technical University – Prague, Springer-Verlag, Junho 1991. Lecture Notes in Computer Science, nu. 545.

[Cla] James Clark. Sp: An sgml system conforming to international standard iso 8879. http://www.jclark.com/sp/index.htm.

[Cor96] Microsoft Corporation. *Microsoft SGML Author for Word.* Microsoft Corporation, 1996.

[Cov] Robin Cover. Sgml parsers. http://www.sil.org/sgml/publicSW.html#parsers.

[DJ90] P. Deransart and M. Jourdan, editors. *Attribute Grammars and their Applications.* INRIA, Springer-Verlag, Setembro 1990. Lecture Notes in Computer Science, nu. 461.

[DJL88] P. Deransart, M. Jourdan, and B. Lorho. Attribute grammars: Main results, existing systems and bibliography. In *LNCS 341.* Springer-Verlag, 1988.

[Gol90] C. Goldfarb. *The SGML Handbook.* Clarendon Press - Oxford, 1990.

[Hen92] Pedro R. Henriques. *Atributos e Modularidade na Especificação de Linguagens Formais.* PhD thesis, Universidade do Minho, Dezembro 1992.

[Her94] E. Herwjnen. *Practical SGML.* Kluwer Academic Publishers, 1994.

[Inc96] Adobe Systems Inc. *FrameMaker + SGML.* Adobe, 1996.

[Jal85] Fahimeh Jalili. A general incremental evaluator for attribute grammars. *Science of Computer Programming*, 5:83–96, 1985.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[Lop96] Alda Lopes. Editor de dtds. Technical report, Dep. Informática - Univ. Minho, Portugal, Aug. 1996.

[MA96] E. Maler and J. Andaloussi. *Developing SGML DTDs – from text to model to markup.* Prentice Hall, 1996.

[7] http://www.jclark.com

[RAH95] J.C. Ramalho, J.J. Almeida, and P.R. Henriques. David - algebraic specification of documents. In A. Nijholt, G. Scollo, and R. Steetskamp, editors, *TWLT10 - Algebraic Methods in Language Processing - AMiLP95*, number 10 in Twente Workshop on Language Technology, Twente University - Holland, Dec. 1995.

[RAH96] J.C. Ramalho, J.J. Almeida, and P.R. Henriques. Document semantics: two approaches. In *SGML'96: Celebrating a decade of SGML*, Sheraton-Boston Hotel, Boston, USA, Nov. 1996.

[Ram93] José Carlos Ramalho. Um Compilador para o GLiTCH. Master's thesis, Departamento de Informática, Universidade do Minho, Setembro 1993.

[Roc92] Jorge G. Rocha. The synthesizer generator - tutorial. Relatório de instalação, G.D. Ciências da Computação, D.I./ Univ. Minho, Universidade do Minho, 1992.

[RT89a] Thomas Reps and Tim Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Texts and Monographs in Computer Science. Springer-Verlag, 1989.

[RT89b] Thomas Reps and Tim Teitelbaum. *The Synthesizer Generator Reference Manual*. Texts and Monographs in Computer Science. Springer-Verlag, 1989.

[RTD83] T. Reps, T. Teitelbaum, and A. Demers. Incremental context-dependent analysis for language-based editors. *ACM Trans. Programming Languages and Systems (TOPLAS)*, 5(3):449–477, 1983.

[Sta] Lennart Staflin. Psgml : a gnu emacs major mode for editing sgml. http://www.lysator.liu.se/projects/about_psgml.html.

[TR81] T. Teitelbaum and T. Reps. The cornell program synthesizer: A syntax-directed programming environment. *Communications of the ACM*, 24(9), Setembro 1981.