Boletim Técnico da Escola Politécnica da USP

Departamento de Engenharia de Computação e Sistemas Digitais

ISSN 1413-215X

BT/PCS/9611

Um Meta-Editor Dirigido por Sintaxe

Margarete Keiko Iwai
João José Neto

São Paulo - 1996

O presente trabalho é parte da dissertação de mestrado "Um Meta-Editor Dirigido por Sintaxe para Linguagens Estruturadas em Blocos", apresentada em 24/02/95, por Margarete Keiko Iwai, sob orientação do Prof. Dr. João José Neto.

A íntegra da dissertação encontra-se à disposição com o autor e na Biblioteca de Engenharia Elétrica da Escola Politécnica da USP.

Iwai, Margarete Keiko
Um meta-editor dirigido por sintaxe / M.K. Iwai, J.José Neto. -- São Paulo : EPUSP, 1996.
17p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/9611)

1. Meta-sistemas 2. Meta-editores 3. Geração automática de software I. José Neto, João II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais III. Título IV. Série
ISSN 1413-215X

CDD 005.3
005
005.1

UM META-EDITOR DIRIGIDO POR SINTAXE

Margarete Keiko Iwai
João José Neto
Escola Politécnica da USP-Av.Prof.Luciano Gualberto, trav.3, n. 158
CEP 05508-900 - São Paulo - SP - Fone:(011)818- 5402
e-mail:mkiwai@dee.poli.usp.br

RESUMO

Este artigo propõe a criação de um meta-editor dirigido por sintaxe para linguagens estruturadas em blocos e apresenta os aspectos mais importantes relativos ao embasamento téorico necessário para a viabilidade de tal projeto, tais como algumas técnicas para formalização de linguagens de programação, tanto a nível sintático quanto semântico, assim como a especificação do sistema.

ABSTRACT

This paper presents the creation of a block-structured programming language syntax-driven editor generator and shows the most important aspects related to the necessary theorethical basis for the construction of this project, such as some techniques of programming language formalization, as well as the system specification.

1. Introdução

O desenvolvimento de sistemas de software exige vários recursos computacionais capazes de executar tarefas básicas e de suporte de modo eficiente e rápido. Isso requer ambientes de desenvolvimento contendo ferramentas integradas e de fácil manipulação, de modo a otimizar o uso desses recursos.

Estes ambientes devem ser compatíveis com os objetivos de cada projeto e fornecer as ferramentas adequadas a cada necessidade.

Com o objetivo específico de melhorar o desempenho de desenvolvimento de programas, surgiram os editores dirigidos por sintaxe, tais como o Emily [6], o Mentor [4], o Cornell Program Synthesizer [13], e outros. Esse editores diferenciam-se dos editores convencionais pela sua capacidade de verificar o ocorrência de erros de sintaxe durante a construção do programa segundo a gramática da linguagem de programação que estiver sendo utilizada. A existência de muitas linguagens de programação e a evolução de outras,

novas e mais complexas, motiva a criação de geradores de ambientes contendo editores baseados na linguagem de aplicação a que se destina.

Os editores dirigidos por sintaxe fazem parte de ambientes de programação interativos, com recursos para criar, editar, compilar, executar e depurar programas.

A característica principal destes editores consiste na edição de programas de forma vinculada às regras gramaticais da linguagem de programação. Para programas contendo sentenças que não obedeçam às regras gramaticais são emitidas mensagens de erros, alertando o usuário sobre essa ocorrência, e que são suprimidas após a sua correção. Em outros casos, tais mensagens podem ser armazenadas e exibidas por solicitação do usuário a qualquer momento.

No projeto da ferramenta proposta neste artigo, o núcleo principal de um editor dirigido por sintaxe opera basicamente com um analisador sintático incremental, podendo ser acrescido de um gerador de códigos a partir da árvore de análise sintática. Este meta-editor é o resultado de um trabalho de dissertação de mestrado [7], que procurou fazer um estudo dos trabalhos relacionados às técnicas de compilação incremental.*

1.1. Motivação

A constante evolução dos recursos computacionais, tanto no nível do hardware quanto do software, favorece a criação de novas ferramentas que auxiliam o desenvolvimento de sistemas.

O projeto proposto neste trabalho tem, por motivação, a criação de um sistema capaz de gerar editores dirigidos por sintaxe, que são ferramentas que auxiliam no desenvolvimento de programas. Esse gerador permite a criação de um novo editor a partir de especificações formais da linguagem de programação a que o editor deverá atender.

Estes editores têm aplicações práticas, orientadas ao treinamento de novos programadores e a facilitar o trabalho de usuários que não possuam obrigatoriamente perfeito domínio da linguagem de programação que estiver sendo utilizada, como é o caso de programadores eventuais.

---

* Este trabalho foi realizado no Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo com o auxílio do CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico.

1.2. Objetivos deste artigo

O objetivo principal deste artigo consiste na apresentação de uma ferramenta para o desenvolvimento interativo de programas. Esta ferramenta gera editores dirigidos por sintaxe para uma classe de linguagens de programação, mais precisamente, linguagens seqüenciais estruturadas em blocos.

Estes editores podem fazer parte de diversos tipos de ambientes de desenvolvimento de *software*, geralmente os ambientes interativos de programação, com recursos para a criação, compilação, execução e depuração de programas.

Além disso, este trabalho procura apresentar uma pequena revisão dos principais conceitos relacionados ao desenvolvimento deste tipo de sistemas.

2. Conceitos relativos aos sistemas dirigidos por sintaxe

Este capítulo tem o objetivo de apresentar, de maneira geral, algumas técnicas utilizadas na formalização de linguagens de programação para a geração de editores dirigidos por sintaxe, procurando, dessa forma, efetuar uma revisão dos conceitos necessários à utilização de tais técnicas, constituindo assim uma análise das formas possíveis da construção de tais ferramentas.

Os conceitos discutidos ao longo deste capítulo concentram-se basicamente em dois assuntos. O primeiro item corresponde à geração do editor, uma vez que a especificação correta da linguagem, tanto ao seu nível sintático quanto ao nível semântico, é fundamental para que a geração do editor seja feita de modo apropriado. O segundo item corresponde às atividades exercidas pelo editor dirigido por sintaxe, o qual efetua a análise do programa-fonte de modo incremental. Esta característica possibilita a codificação de programas de modo tal que o editor possa efetuar simultaneamente a edição do programa e a verificação da sua correção sintática.

2.1 Descrição formal de linguagens

A maioria das descrições de linguagens de programação que chegam ao alcance do programador através de manuais e livros populares costuma ser feita de maneira totalmente informal, sendo elas expressas de modo narrativo, através da utilização de linguagem natural ao invés de alguma notação formal, e portanto rigorosa.

Para muitas pessoas, a primeira classe de descrições é satisfatória, embora para outras, tais como projetistas e estudiosos da área, isso resulte em problemas, ocasionados pela falta de precisão e de rigor, sempre presentes nas especificações informais da linguagem.

No meta-editor apresentado neste artigo, a formalização da linguagem de programação a ser utilizada é um fator muito importante na geração dos seus editores dirigidos por sintaxe, já que das regras gramaticais em questão depende a operação da função de verificação sintática do editor. Esta verificação deve também estender-se aos aspectos dependentes de contexto da linguagem, como por exemplo, entre diversos outros, a consistência dos tipos das variáveis e a verificação dos parâmetros das funções.
São comentadas a seguir algumas técnicas, usualmente empregadas para a formalização da sintaxe e da semântica das linguagens.

2.1.1 Formalização da sintaxe e da semântica

As técnicas empregadas para a especificação formal de linguagem de programação variam em graus de complexidade. Os métodos utilizados para descrever somente a sintaxe são mais simples do que as técnicas que também especificam a semântica de uma linguagem de programação.
Existem várias técnicas que são utilizadas na especificação de linguagens, e que variam de acordo com o nível de descrição exigidos para a sintaxe e a semântica. As técnicas podem ser divididas basicamente em três grupos, os quais formalizam respectivamente os aspectos livre de contexto, dependente de contexto e a semântica da linguagem de programação. O livro [10] constitui uma boa referência a respeito do assunto.
As notações que descrevem principalmente os aspectos livres de contexto da linguagem, podem ser consideradas meta-linguagens, e em geral descrevem apenas a sintaxe da linguagem, formalizando as suas regras de produção, não incorporando os aspectos relacionados ao contexto. Alguns exemplos são a notação BNF e a notação de Wirth.
As notações que especificam gramáticas dependentes de contexto permitem descrever tanto a sintaxe quanto a semântica estática da linguagem de programação, como ocorre, por exemplo, com as gramáticas de atributos e as gramáticas W.
Esses aspectos dependentes de contexto de um programa, caracterizados pela semântica estática da linguagem de programação, envolvem a utilização de identificadores, a verificação do número e do tipo de parâmetros de procedimentos, e outros aspectos que exigem mais do que uma simples verificação da sintaxe do programa, devendo ser tratados de uma forma mais cuidadosa.
Para a descrição da semântica, algumas das formulações utilizadas são as chamadas semânticas denotacional, operacional e axiomática [9].

2.2 Análises Sintática e Semântica Convencionais

A análise sintática representa uma fase muito importante no processo de compilação de um programa, pois ela promove a verificação da estrutura sintática que deve corresponder à gramática da linguagem.

A análise semântica na construção de um compilador é incorporada em parte à análise sintática através da execução de ações semânticas, promovidas pelo analisador sintático. Manifesta-se também através das rotinas de geração de código a partir da árvore de sintaxe abstrata do programa.

O presente trabalho não tem a intenção de fazer um estudo aprofundado das técnicas de análise sintática e semântica convencionais. Existem boas referências a respeito do assunto, que podem ser encontradas em livros como [1] e [2].

2.3 Análises Sintática e Semântica Incrementais

Uma das características mais importantes nos ambientes dirigidos por sintaxe é a capacidade de se verificar a correção da sintaxe do programa enquanto este estiver sendo editado. Em alguns ambientes, que ofereçam como recurso a possibilidade da execução do programa, a análise semântica, voltada à interpretação do mesmo e à geração de código, pode ser efetuada simultaneamente.

Para isso, podem ser empregados analisadores sintáticos e semânticos incrementais, que reutilizam parte do material construído quando de qualquer eventual modificação do programa. Esta reanálise envolve apenas as partes do programa efetivamente afetadas pelas alterações, poupando, assim, tempo de processamento.

Convém, naturalmente, que essas reanálises sejam rápidas o suficiente para que o usuário não perceba o tempo empregado para a localização e posterior computação da parte afetada da árvore de análise sintática.

A análise sintática incremental é uma das principais características na implementação de um editor dirigido por sintaxe. Um analisador sintático incremental constrói a árvore de análise sintática do programa em estudo a partir do aproveitamento de partes de uma árvore que tenha sido previamente construída para uma versão anterior do programa.

Um dos métodos que podem ser utilizados para a análise sintática incremental é apresentado em [3], e explora uma técnica de reutilização dos estados do analisador sintático. Esta técnica consiste em ligar estados do analisador sintático a marcadores ("*tokens*"). Após uma modificação ter sido feita no programa, o analisador é reativado a partir de um estado adequado, anteriormente salvo, e promove o prosseguimento da

análise do programa em um ponto do texto, que esteja situado em alguma posição conveniente do mesmo, anterior àquela em que tenha sido efetuada a modificação. A análise termina quando o analisador encontrar uma destas marcas em algum ponto do texto situado após a região modificada na antiga configuração.

Em [5] é apresentado um método alternativo para a realização da análise sintática incremental de linguagens livres de contexto. Se a cadeia $xx'zy'y$ é modificada para $xx'z'y'y$, onde x' e y' têm comprimento k, sendo k o *look-ahead* necessariamente consultado pelo analisador sintático, então as árvores sintáticas previamente geradas para x e y serão ainda válidas após encerrada a modificação. Este método consiste em evitar a reanálise da parte esquerda da cadeia de caracteres, xx', até o ponto em que ocorre a modificação e de modo simétrico, evitar também a parte direita da cadeia, yy'.

Em [11], encontra-se mais um método de análise sintática incremental. Neste, a análise incremental pode ser realizada através de um reconhecedor de sub-cadeias de caracteres, e a re-análise sintática incide sobre o texto ao nível de uma sub-árvore sintática apenas. Após uma modificação no programa, deve-se localizar, na árvore de análise sintática da versão não-modificada do programa, a sub-cadeia s' pertencente à menor sub-árvore possível que contenha integralmente a região do texto sobre a qual incidiram as alterações. O analisador sintático incremental age da seguinte forma:

Se sub-árvore escolhida for do tipo T e a sub-cadeia s' puder ser analisada com sucesso, com a geração de uma nova sub-árvore que seja também do tipo T, então esta última pode substituir a anterior na árvore original. No entanto, se s' não puder ser analisada, isto pode ser o caso de que a modificação introduziu erros sintáticos, ou que a sub-árvore escolhida tenha sido pequena demais. Estes dois casos devem estar bem distinguidos, para que o analisador sintático incremental possa agir de acordo com cada caso.

Nestes três métodos apresentados, deve-se levar em conta ao menos a quantidade de armazenamento de informações que deve ser feita, e o tempo gasto no processamento, pois estes fatores têm uma infuência significativa no caso de um sistema que se propõe a editar programas de tamanhos variados, como é, em particular, o caso de um editor dirigido por sintaxe.

Como foi visto na seção anterior, as técnicas utilizadas para a especificação da semântica de uma linguagem de programação constituem uma tarefa bem mais complexa do que a representada pela especificação da sintaxe. Nesta seção é possível identificar um fenômeno similar entre os processos de análise sintática e semântica incrementais.

Algumas técnicas de análise semântica incremental, empregadas na construção de sistemas orientados à linguagem, estão baseados em gramáticas de atributos [12] e em suas extensões, como é o caso dos *action equations* [8].

Através das gramáticas de atributos, a semântica da linguagem de programação é descrita como um conjunto de declarações de atributos associados com cada símbolo da gramática e uma coleção de equações semânticas, cada qual associada a uma produção, definindo os valores dos atributos dos símbolos das produções. Os valores dos atributos são determinados pela avaliação de todas as equações semânticas como um conjunto de equações simultâneas. Durante a edição do programa, um algoritmo incremental reavalia estes atributos, cujos valores podem ser alterados como resultado da substituição de uma sub-árvore em cada operação de edição. Esta sub-ávore está ligada a uma árvore semântica, uma árvore de derivação tal que, para os atributos de cada nó estão associados um valor ou um marcador especial.

A técnica que utiliza o paradigma dos *action equation* procura utilizar uma extensão das gramáticas de atributos para que seja possível expressar a semântica dinâmica de programas e também a semântica estática. Isto é feito através de equações semânticas ativas e passivas que podem ser alteradas, algumas se transformam de passivas para ativas e retornam a passivas novamente, de acordo com os comandos externos do usuário e computações internas envolvendo equações de propagação e de espera.

3. Especificação do Projeto

Neste capítulo é apresentada uma especificação do meta-editor dirigido por sintaxe proposto, bem como a metodologia adotada para a sua realização. Foi adotada a construção de um protótipo mínimo para o desenvolvimento desta ferramenta, de modo que fosse permitida a expansão do sistema pela incorporação de novos módulos, tal como um compilador incremental ou um interpretador, e também para que seja possível visualizar o funcionamento das operações básicas do programa.

Este gerador deve criar editores que verifiquem a ocorrência de erros de sintaxe à medida que o usuário constroe seu programa. Foi projetado principalmente para atender ao grupo de linguagens do tipo estruturadas em blocos.

Este gerador procura atender às necessidades de dois principais grupos de usuários, a dos projetistas (que construirão os editores) e a dos usuários propriamente ditos (que utilizarão os editores gerados pelos projetistas).

Ao usuário projetista, este gerador procura oferecer a opção da utilização de uma ferramenta que o auxilie no desenvolvimento de novos sistemas, como ocorre, por exemplo, quando da criação de uma nova linguagem de programação, viabilizando uma posterior incorporação deste editor como parte de sistemas maiores, integrado a outras ferramentas similares. Este usuário deve ter um mínimo de conhecimento dos princípios de

funcionamento das ferramentas LEX e YACC, assim como de técnicas de formalização de linguagens de programação e de construção de compiladores.

Para o segundo grupo de usuários, esta ferramenta procura gerar produtos que possam atender especialmente às necessidades daqueles programadores que não tenham obrigatoriamente conhecimentos profundos da linguagem para a qual o editor foi desenvolvido.

Este editor deverá também estar preparado para dar a seu usuário uma orientação em relação à sintaxe do programa que estiver sendo codificado, assumindo assim finalidades didáticas, de apoio ao treinamento de novos programadores para a linguagem.

Um outro grupo de usuários destes editores inclui aqueles que sabem programar, e que se utilizam deste editor como ferramenta de auxílio ao desenvolvimento de sistemas de *software*, o que proporciona uma redução do tempo de codificação dos programas através da imposição e da verificação automática da sintaxe, impedindo a construção de partes do programa que contenham erros de sintaxe.

Como características principais do gerador e dos editores por ele criados podem ser destacadas as seguintes:

- Os editores gerados deverão ser capazes de efetuar a edição de, ao menos, um tipo de documento: os programas de computação;
- As linguagens a serem atendidas pelos editores gerados deverão ser, pelo menos, as do tipo estruturado em blocos;
- O gerador deverá permitir a utilização de gabaritos na edição do programa, possibilitando ainda a escolha, por parte do usuário, da edição pelo método convencional, isto é, o modo de edição de texto normalmente utilizado, em que o usuário pode editar e manipular livremente o texto, sem as restrições impostas pelos gabarito (*templates*) estruturas pré definidas com um padrão pré-formatado de caracteres e marcas de pontuação, onde as palavras-chave, pontuações e formato de indentação não podem ser alterados.
- Quando operando em modo convencional, os editores gerados deverão efetuar a verificação da ocorrência de erros de sintaxe, e simultânea emissão das mensagens correspondentes;
- Quando em modo dirigido por sintaxe, os editores não deverão permitir a introdução de erros de sintaxe, pela imposição da sintaxe correta através do uso de gabaritos.
- Os editores deverão garantir um certo nível de documentação, através da imposição da presença de comentários em pontos pré-estabelecidos do texto-fonte editado, alertando assim ao programador sobre a necessidade de documentar o código do programa à medida que este estiver sendo codificado.

- O editor deverá estar preparado para incorporar funções de compilação, para a geração automática de código executável, possibilitando portanto a execução do programa mesmo estando este incompleto (este recurso não está incorporado à primeira versão do editor, podendo ser acrescentado posteriormente).

Alguns dos principais requisitos impostos para o sistema são:

- **modularidade** - O sistema devide-se um vários módulos, tais como o analisador léxico, o analisador sintático, o módulo de criação e manipulação da árvore sintática, o sincronizador do cursor com o texto do programa, o módulo que manipula a apresentação do texto na tela, o manipulador dos arquivos, entre outros. E também existe o programa principal, que executa a tarefa de gerenciar todos os módulos, realizando a tarefa destinada à interface. Essa modularidade permite que sejam facilitadas operações tais como a manutenção, a atualização, e a modificação do sistema, sem prejuízo dos outros módulos.
- **facilidade de migração** - Com o intuito de tornar o sistema o mais acessível possível aos diversos possíveis usuários, a escolha da plataforma de trabalho (micro-computadores pessoais compatíveis com IBM, sendo executados sob sistema operacional MS-DOS), bem como as ferramentas adotadas (LEX e YACC) para a construção do gerador e do editor, tiveram como meta aumentar a possibilidade de migração do sistema a outras plataformas com o mínimo de alterações. .
- **facilidade de escolha da linguagem** - As linguagens que podem ser utilizadas pelos editores procuram ser diversas, além das que pertencem à classe das linguagens estruturadas em blocos, uma vez que o usuário pode criar seus próprios editores, desde que, para isso, seja capaz de definir a sintaxe da linguagem desejada nos moldes esperados pelas conhecidíssimas ferramentas LEX e YACC.
- **personalização** (*customizing*) - O gerador permite que o usuário personalize seus editores segundo critérios próprios, podendo assim o usuário configurar algumas características do editor, como, por exemplo, uma particular formatação do texto na tela, definindo as tabulações e indentações dos comandos da linguagem

3.1 Ferramentas adicionais

Um critério, adotado para o desenvolvimento deste sistema, consiste na utilização de recursos disponíveis, que facilitem o processo de sua construção. Entre estes recursos podem ser destacados os geradores automáticos de analisadores léxico e sintático, LEX e YACC, já mencionados em capítulos anteriores.

Os principais motivos para a escolha dos programas LEX e do YACC são a disponibilidade de tais ferramentas em diversas plataformas, além do fato de que os programas por elas gerados estarem codificados em linguagem C, que é de grande aceitação e portatilidade. Para tornar todo o sistema ainda mais portátil, pode ser destacada a adoção da linguagem C para o desenvolvimento de todo o projeto.

3.2 Descrição Funcional

Neste item serão descritas as principais funções executadas pelo sistema de geração dos editores dirigidos por sintaxe. Para a geração de cada editor, as funções do gerador são utilizadas apenas para a criação do editor específico, podendo este sofrer posteriores alterações, caso o produto não esteja de acordo com as exigências do projetista.

O gerador de editores apóia-se nas seguintes funções principais:

- **Especificação da linguagem** - Esta função permite que o usuário especifique as características léxicas, sintáticas e semânticas da linguagem de programação escolhida. Tais especificações são utilizadas pelo gerador para a montagem do editor, na fase de criação dos seus principais módulos, e poderão servir como base para um futuro gerador de código, a ser incorporado no sistema.
- **Especificação da formatação do texto na tela** - Esta função pode ser utilizada caso o usuário deseje alterar a formatação do texto produzida automaticamente pelo sistema na tela, podendo influir, por exemplo, na indentação, na tabulação e em algumas outras características. Caso contrário, o gerador utilizará valores *default* para esta tarefa.

Para a elaboração do editor, constata-se que as grandes funções deste programa são basicamente as mesmas utilizadas em qualquer ambiente convencional de desenvolvimento de arquivos de texto de programas (criação, alteração e salvamento de programas), com opções de entrada via arquivo já editado ou então via teclado. Adicionalmente, deseja-se ter a opção de escolha entre uma edição convencional e uma edição sintática, restrita à imposição ditada pelos gabaritos.

Para a criação e alteração de programas, o sistema gera, simultaneamente, dois tipos de

arquivos, um contendo o código ASCII que representa o programa fonte, e um outro, contendo uma árvore sintática, que é a representação estrutural interna do programa. Quando o usuário quizer salvar o programa que está sendo editado, os dois arquivos são integrados em um único. Isto é feito gravando-se a árvore sintática após o texto do programa. As duas partes são separadas pelo símbolo "@@".

Se o usuário quizer salvar apenas o texto do programa, então ele utiliza uma função que realiza a exportação do programa em código ASCII para um arquivo.

As funções principais (modos de operação) dos editores gerados pelo sistema podem ser resumidas às seguintes:

- criação de um novo arquivo;
- carregamento de um arquivo já existente;
- transferência (importação) de arquivos (em código ASCII);
- transferência (exportação) de arquivos (em código ASCII);
- mudança do modo de edição;
- salvamento de arquivos.

Nas duas primeiras funções, o sistema se encarrega de verificar a existência dos arquivos contendo o programa em ASCII, e a árvore sintática associada.

Para a terceira função, o sistema deverá "traduzir" o arquivo externo de modo a se adequar ao esquema tratado pelo editor. Além disso, será criado o arquivo de representação interna, no caso, a árvore sintática.

A quarta função realiza o que foi dito anteriormente. Salva o programa na forma de código ASCII, sem a árvore sintática.

A última função salva os arquivos na forma padrão do sistema, e, se necessário, faz uma cópia de segurança (*backup*).

3.3 Metodologia adotada

A metodologia adotada para a construção do sistema aqui descrito sofreu forte influência das ferramentas LEX e YACC, adotadas para a geração automática de analisadores léxicos e sintáticos.

A seguir serão apresentados alguns aspectos do projeto, incluindo as principais estruturas de dados adotadas, a arquitetura e a forma de interação com o usuário.

Estruturas de dados - Uma das principais estruturas de dados utilizada pelo sistema é a *árvore sintática* do programa, que representa o programa analisado pelo editor através de uma árvore n-ária (árvore cujos nós possuem um número variado de descendentes).

Esta árvore é construída à medida que o programa vai sendo editado, podendo ser modificada em qualquer época da análise. Como a árvore e o texto do programa estão intimamente relacionados, por serem representações equivalentes do mesmo programa, qualquer alteração realizada pelas operações de edição obviamente deverá afetar a ambas.

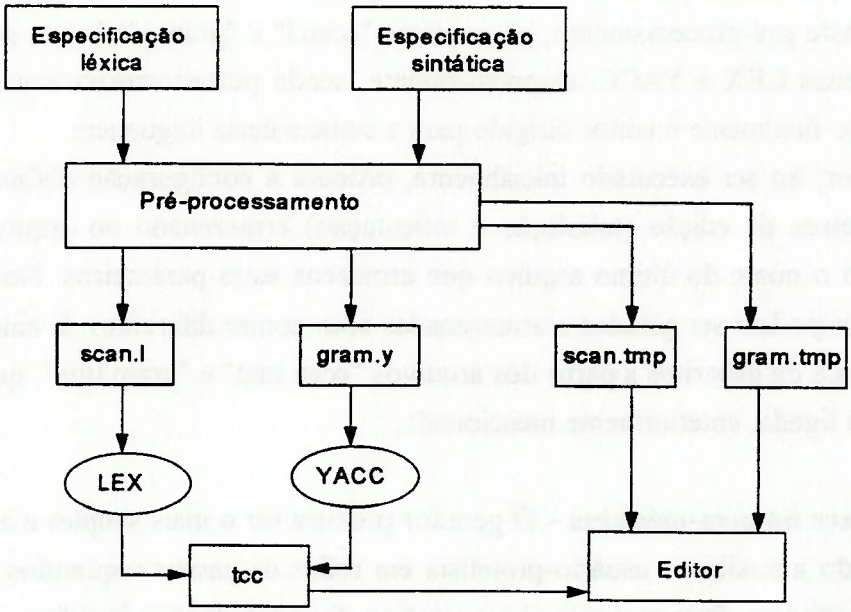
Para proporcionar ao usuário uma navegação rápida pelo programa, os nós da árvore sintática são interligados por ponteiros bidirecionais, isto é, cada célula (ou nó) da árvore aponta para o respectivo ancestral, para o seu primeiro descendente, e para os seus nós-irmãos direito e esquerdo. Os nós que representam símbolos terminais apresentam, ainda, dois ponteiros adicionais, responsáveis pela formação do contorno da árvore sintática. Estes ponteiros apontam, portanto, apenas para nós-terminais, formando uma lista duplamente ligada de nós terminais, permitindo assim o acesso fácil a elementos terminais vizinhos.

Cada célula da árvore sintática apresenta um campo que armazena o símbolo (terminal ou não-terminal) que está representando. No caso dos nós que armazenam símbolos terminais, existem outros dois campos, que armazenam as coordenadas do primeiro caractere da cadeia na representação textual do terminal em questão, em relação ao início do programa que está sendo editado.

Outras duas estruturas adotadas são a de uma *pilha sintática*, utilizada pelos mecanismos de análise sintática, atrelados ao editor, para a construção da árvore e para a correta avaliação da validade sintática do texto do programa, também uma lista ligada que armazena a biblioteca de gabaritos.

A seguir será apresentado um esquema global do sistema, que procura transmitir uma idéia geral da arquitetura do meta-editor e seus principais módulos.

Arquitetura do sistema - A figura seguinte ilustra, em grandes blocos, os aspectos globais da arquitetura do sistema.



Segundo este esquema, os principais elementos da ferramenta estão localizados nos módulos pré-processador e editor.

O usuário da categoria dos projetistas encarrega-se da execução do *pré-processador*. Ele deve fornecer uma especificação da linguagem desejada, tanto ao nível léxico quanto ao nível sintático, utilizando para isso a notação usual, imposta pelas ferramentas LEX e YACC.

Estas especificações devem ser submetidas a um pré-processamento, que insere ações semânticas em pontos pré-determinados do programa, e de onde são retiradas algumas informações, necessárias para um posterior processamento. No caso da especificação léxica, por exemplo, são inseridas funções de edição do texto, gerando o arquivo "scan.l", e são obtidos os itens léxicos básicos *(tokens)* da linguagem, com seus respectivos símbolos terminais, gerando-se, com tais informações, o arquivo "scan.tmp".

No arquivos de especificação sintática, é inserida uma série de ações, tais como, por exemplo, a de edição do texto, da árvore sintática, da pilha sintática, sendo gerado um arquivo denominado "gram.y". Tais ações são inseridas nas regras de produção da gramática da linguagem, e na seção de funções semânticas exigidas pelo programa YACC. A inserção dessas ações entre as regras de produção deve ser feita de modo tal que não

sejam introduzidos conflitos na gramática, devendo o usuário-projetista estar atento a tais eventualidades.
Além das inserções, o pré-processador gera outro arquivo, de nome "gram.tmp" que retira qualquer ação semântica que estiver originalmente incluída na especificação fornecida. Este arquivo é posteriormente utilizado na geração da biblioteca de gabaritos.
Após este pré-processamento, os arquivos "scan.l" e "gram.y" devem ser submetidos aos programas LEX e YACC, respectivamente, sendo posteriormente compilados e ligados, gerando finalmente o editor dirigido para a sintaxe desta linguagem.
O editor, ao ser executado inicialmente, procura a configuração *default* relacionada aos parâmetros de edição (tabulação e indentação) armazenado no arquivo "conf.ini", que contém o nome do último arquivo que armazena estes parâmetros. Novas configurações também podem ser geradas e armazenadas com nomes diferentes. E ainda o editor gera a biblioteca de gabaritos a partir dos arquivos "scan.tmp" e "gram.tmp", que será armazenda na lista ligada, anteriormente mencionada.

Interface homem-máquina - O gerador procura ser o mais simples e amigável possível, de modo a auxiliar o usuário-projetista em todos os passos requeridos para a criação de novos editores. Para este primeiro protótipo, foi adotada uma interface do tipo que requer ao usuário apenas exatidão nas respostas às requisições feitas pelo gerador, como é o caso dos nomes dos arquivos das especificações da linguagem.
O editor conta com um sistema interativo de teclas funcionais, que procuram facilitar a manipulação da ferramenta. As principais teclas permitem carregar um arquivo existente, criar um novo arquivo, salvar o arquivo corrente, sair do editor e mudar os parâmetros de apresentação do texto na tela.
Durante a edição de programas, o editor opera inicialmente no modo de gabaritos, podendo passar para o modo convencional bastando para isso o acionamento de um comando próprio, fornecido pelo usuário.

5. Testes efetuados

O método empregado para efetuar os testes no meta-editor teve como objetivo verificar se a ferramenta era capaz de executar as principais atividades a que estava destinada.

Os testes foram centrados principalmente no pré-processador e no editor gerado, em pontos críticos do sistema, como será indicado a seguir.

No pré-processador foram testadas diversas linguagens experimentais estruturadas em blocos, para que pudesse ser verificado se ele era capaz de gerar o editor desejado. Os principais testes visaram exercitar:

- A inserção das ações semânticas nos arquivos de especificação; e
- A geração dos arquivos intermediários do sistema, a partir dos mesmos arquivos de especificação anteriores.

Nos editores gerados a partir das especificações do usuário, foram feitos testes que procuraram avaliar se o editor é capaz de executar as principais funções de edição sobre programas editados via console, e também sobre arquivos importados. Os principais testes procuraram verificar:

- A construção das árvores sintáticas dos programas que estão sendo editados;
- A geração da biblioteca de gabaritos (*templates*);
- A edição incremental de programas, procurando avaliar o funcionamento do analisador sintático incremental.

Além disso, foram testados outros detalhes, tais como:

- mudanças de parâmetros das características do editor, efetuadas durante a edição de programas; e
- a interface homem-máquina.

5.1 Avaliações finais

Os testes apresentaram resultados satisfatórios, proporcionando inúmeras ocasiões para a localização de imperfeições no programa, e tendo contribuído para o aperfeiçoamento de diversas características do sistema.

O gerador mostrou um desempenho correto, realizando as inserções das ações semânticas e a geração dos arquivos intermediários, quando acionado com dados adequadamente fornecidos pelo usuário. Foram simulados casos em que poderiam ocorrer erros do usuário, e as respectivas soluções mostraram respostas satisfatórias, o mesmo acontecendo com os editores gerados, que, embora limitados, executaram adequadamente as tarefas especificadas.

6. Conclusão

O desenvolvimento desta ferramenta demonstrou, a despeito da mística que paira sobre programas desta natureza, ser viável a concepção e a implementação, a baixíssimo custo, de meta-sistemas, capazes de gerar programas com perfis específicos, impostos e controlados pelo usuário.

Este trabalho teve como objetivos principais dar uma contribuição à área através da descrição da concepção e da criação de um protótipo de uma ferramenta que pudesse exercer as operações de um gerador de editores dirigidos por sintaxe para linguagens estruturadas em blocos, e também apresentar uma visão geral de alguns trabalhos relacionados à área, bem como mostrar algumas das principais bases teóricas necessárias à construção de uma ferramenta desta categoria.

Este meta-editor procurou também levar em consideração seu importante potencial como um instrumento de trabalho como ferramenta didática para o ensino e treinamento de novos programadores, e de estudantes de computação em geral.

Além disso, serviu como estudo de caso para as aplicações de métodos de compilação incremental à geração de meta-sistemas, introduzindo técnicas de análise sintática incremental, bem como o de compilação, embora este último caso não tenha sido diretamente aplicado, na prática, na elaboração do presente trabalho.

Referências Bibliográficas

[1] AHO, A.V.; SETHI, R.; ULLMAN, J.D. **Compilers:** principles, techniques, and tools. Reading, Addison-Wesley, 1986.

[2] AHO, A.V.; ULLMAN, J.D. **The theory of parsing, translation and compiling**. Englewood Cliffs, Prentice-Hall, 1973. 2.v.

[3] CELENTANO, A. Incremental LR parsers. **Acta Informatica**, v.10, n.4, p.307-21, 1978.

[4] DONZEAU-GOUGE, V.; HUET, G.; LANG, B. Programming environments based on structured editors: The MENTOR Experience. In: BARSTOW, D.R.; SHROBE, H.E.; SANDEWALL, E., ed. **Interactive programming environments**. New York, McGraw-Hill, 1984

[5] GHEZZI, C; MANDRIOLI, D. Incremental parsing. **ACM Transaction on Programming Languages and Systems**, v.1, n.1. p.58-70, 1979.

[6] HANSEN, W.J. User engineering principles for interactive systems. In: FALL JOINT COMPUTER CONFERENCE, Las Vegas, 1971. **Proceedings**. Montvale, AFIPS Press, 1971. p.523-32.

[7] IWAI, M.K. **Um meta-editor dirigido por sintaxe para linguagens estruturadas em blocos**. São Paulo, 1994. 105p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo

[8] KAISER, G.E. Incremental dynamic semantics for language-based programming environments. **ACM Transactions on Programming Languages and Systems**, v.11, n.2, p.169-93, 1989.

[9] MEYER, B. **Introduction to the theory of programming languages**. New York, Prentice-Hall, 1990 (Prentice-Hall International Series in Computer Science)

[10] PAGAN, F.G. **Formal specification of programming languages:** a Panoramic Primer. New Jersey, Prentice-Hall, 1981.

[11] REKERS, J.; KOORN, W. Substring parsing for arbitrary context-free grammars. **ACM SIGPLAN Notices**, v.26, n.5, p.59-66, 1991.

[12] REPS, T.W. **Generation Language-Based Environments**. Cambridge, M.I.T. Press, 1983. (ACM Doctoral Dissertation Award Series)

[13] TEITEUBAUM, T.; REPS, T. The Cornell program synthesizer. **Communications of the ACM**, v.24, n.9, p.563-73, 1981.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PCS/9301 - Interligação de Processadores através de Chaves Ômicron - GERALDO LINO DE CAMPOS, DEMI GETSCHKO

BT/PCS/9302 - Implementação de Transparência em Sistema Distribuído - LUÍSA YUMIKO AKAO, JOÃO JOSÉ NETO

BT/PCS/9303 - Desenvolvimento de Sistemas Especificados em SDL - SIDNEI H. TANO, SELMA S. S. MELNIKOFF

BT/PCS/9304 - Um Modelo Formal para Sistemas Digitais à Nível de Transferência de Registradores - JOSÉ EDUARDO MOREIRA, WILSON VICENTE RUGGIERO

BT/PCS/9305 - Uma Ferramenta para o Desenvolvimento de Protótipos de Programas Concorrentes - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9306 - Uma Ferramenta de Monitoração para um Núcleo de Resolução Distribuída de Problemas Orientado a Objetos - JAIME SIMÃO SICHMAN, ELERI CARDOSO

BT/PCS/9307 - Uma Análise das Técnicas Reversíveis de Compressão de Dados - MÁRIO CESAR GOMES SEGURA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9308 - Proposta de Rede Digital de Sistemas Integrados para Navio - CESAR DE ALVARENGA JACOBY, MOACYR MARTUCCI JR.

BT/PCS/9309 - Sistemas UNIX para Tempo Real - PAULO CESAR CORIGLIANO, JOÃO JOSÉ NETO

BT/PCS/9310 - Projeto de uma Unidade de Matching Store baseada em Memória Paginada para uma Máquina Fluxo de Dados Distribuído - EDUARDO MARQUES, CLAUDIO KIRNER

BT/PCS/9401 - Implementação de Arquiteturas Abertas: Uma Aplicação na Automação da Manufatura - JORGE LUIS RISCO BECERRA, MOACYR MARTUCCI JR.

BT/PCS/9402 - Modelamento Geométrico usando do Operadores Topológicos de Euler - GERALDO MACIEL DA FONSECA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9403 - Segmentação de Imagens aplicada a Reconhecimento Automático de Alvos - LEONCIO CLARO DE BARROS NETO, ANTONIO MARCOS DE AGUIRRA MASSOLA

BT/PCS/9404 - Metodologia e Ambiente para Reutilização de Software Baseado em Composição - LEONARDO PUJATTI, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9405 - Desenvolvimento de uma Solução para a Supervisão e Integração de Células de Manufatura Discreta - JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9406 - Método de Teste de Sincronização para Programas em ADA - EDUARDO T. MATSUDA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9407 - Um Compilador Paralelizante com Detecção de Paralelismo na Linguagem Intermediária - HSUEH TSUNG HSIANG, LÍRIA MATSUMOTO SAITO

BT/PCS/9408 - Modelamento de Sistemas com Redes de Petri Interpretadas - CARLOS ALBERTO SANGIORGIO, WILSON V. RUGGIERO

BT/PCS/9501 - Sintese de Voz com Qualidade - EVANDRO BACCI GOUVÊA, GERALDO LINO DE CAMPOS

BT/PCS/9502 - Um Simulador de Arquiteturas de Computadores "A Computer Architecture Simulator" - CLAUDIO A. PRADO, WILSON V. RUGGIERO

BT/PCS/9503 - Simulador para Avaliação da Confiabilidade de Sistemas Redundantes com Reparo - ANDRÉA LUCIA BRAGA, FRANCISCO JOSÉ DE OLIVEIRA DIAS

BT/PCS/9504 - Projeto Conceitual e Projeto Básico do Nível de Coordenação de um Sistema Aberto de Automação, Utilizando Conceitos de Orientação a Objetos - NELSON TANOMARU, MOACYR MARTUCCI JUNIOR

BT/PCS/9505 - Uma Experiência no Gerenciamento da Produção de Software - RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/9506 - MétodOO - Método de Desenvolvimento de Sistemas Orientado a Objetos: Uma Abordagem Integrada à Análise Estruturada e Redes de Petri - KECHI HIRAMA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9601 - MOOPP: Uma Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo - ELISA HATSUE MORIYA HUZITA, LÍRIA MATSUMOTO SATO

BT/PCS/9602 - Estudo do Espalhamento Brillouin Estimulado em Fibras Ópticas Monomodo - LUIS MEREGE SANCHES, CHARLES ARTUR SANTOS DE OLIVEIRA

BT/PCS/9603 - Programação Paralela com Variáveis Compartilhadas para Sistemas Distribuídos - LUCIANA BEZERRA ARANTES, LIRIA MATSUMOTO SATO

BT/PCS/9604 - Uma Metodologia de Projeto de Redes Locais - TEREZA CRISTINA MELÓ DE BRITO CARVALHO, WILSON VICENTE RUGGIERO

BT/PCS/9605 - Desenvolvimento de Sistema para Conversão de Textos em Fonemas no Idioma Português - DIMAS TREVIZAN CHBANE, GERALDO LINO DE CAMPOS

BT/PCS/9606 - Sincronização de Fluxos Multimídia em um Sistema de Videoconferência - EDUARDO S. C. TAKAHASHI, STEFANIA STIUBIENER

BT/PCS/9607 - A importância da Completeza na Especificação de Sistemas de Segurança - JOÃO BATISTA CAMARGO JÚNIOR, BENÍCIO JOSÉ DE SOUZA

BT/PCS/9608 - Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança - BRÁULIO COELHO ÁVILA, MÁRCIO RILLO

BT/PCS/9609 - Implementação de Engenharia Simultânea - MARCIO MOREIRA DA SILVA, MOACYR MARTUCCI JÚNIOR

BT/PCS/9610 - Statecharts Adaptativos - Um Exemplo de Aplicação do STAD - JORGE RADY DE ALMEIDA JUNIOR, JOÃO JOSÉ NETO