Uma Extensão de Pascal Orientada a Objetos

Marco Túlio de O. Valente[1]
Orientador: Leacir Nogueira Bastos[2]

Universidade Federal de Viçosa
Departamento de Informática
CEP: 36570 - Viçosa - MG
Fone: (031) 899-2394

Resumo

Nesse artigo descreve-se uma extensão da linguagem Pascal, chamada PASCAL OBJ, que incorpora conceitos do paradigma de programação orientada a objetos, como classes, objetos, instâncias, métodos e mensagens. A extensão incorpora ainda os conceitos de encapsulamento, herança e polimorfismo (esse último de forma parcial). A extensão proposta tem a característica de usar um esquema de tradução baseado em pré-processamento, isto é, os comandos da linguagem são traduzidos em comandos de Pascal padrão. Dada à sua simplicidade, a extensão é adequada ao ensino de programação orientada a objetos.

Abstract

This paper describes a extension of Pascal programming language, called PASCAL OBJ, which supports concepts of object-oriented programming, like classes, objects, instances, methods and messages. The extension supports yet the concepts of encapsulation, inheritance and polymorphism (the late only in a parcial form). The traduction of the extension is based in a pre-processor schema: the commands of the language are translated in commands of standard Pascal. Due to its simplicity, PASCAL OBJ is indicated to teaching object-oriented programming.

[1]Estudante de graduação em Informática (último período). Áreas de interesse: Programação Orientada a Objetos e Construção de Compiladores. E-mail: valente@brufv.bitnet

[2]Prof. Adjunto, PhD em Ciência da Computação (Univ. Clayton, USA). Áreas de interesse: Programação Orientada a Objetos, Construção de Compiladores e Sistemas Operacionais. E-mail: leacirnb@brufv.bitnet

1 Introdução

O desenvolvimento de software de acordo com o paradigma de **Programação Orientado a Objetos (POO)** é relativamente recente. O termo *orientado a objeto*, na verdade, surgiu em meados da década de 70, com o desenvolvimento da linguagem Smalltalk [Gol89]. O objetivo da POO é proporcionar um salto de qualidade no processo de desenvolvimento de software, seja pela produção de programas com estruturas mais aderentes à estrutura dos problemas a que se destinam, seja pela ênfase dada à reutilização de código.

Com a difusão das idéias do paradigma, foram sendo produzidas **Linguagens de Programação Orientadas a Objetos (LOO)**. Essas linguagens são assim denominadas por incorporarem características que viabilizam a produção de software de acordo com os princípios da POO. A primeira dessas linguagens foi Smalltalk. Gradativamente, no entanto, foram surgindo outras linguagens, como Eiffel [Mey88], C++ [Str86b] e Turbo Pascal 5.5 [Bor89].

O objetivo desse trabalho é descrever uma extensão de Pascal orientada a objetos, isto é, um superconjunto de Pascal padrão que incorpora conceitos de POO. A linguagem desenvolvida, denominada PASCAL OBJ, atende tanto ao *paradigma estruturado* como ao *paradigma orientado a objetos*, podendo ser classificada como uma LOO híbrida. Uma outra característica dessa extensão é o seu esquema de tradução baseado em pré-processamento, isto é, os recursos de POO da linguagem são *expandidos* por um pré-processador em recursos de Pascal padrão [Jen88].

Inicialmente, nesse artigo, descrevem-se nas seções 2 e 3 conceitos básicos do paradigma orientado a objetos e de linguagens orientadas a objetos, respectivamente. A seção 4 trata da definição da linguagem PASCAL OBJ. Na seção 5, descreve-se o esquema de pré-processamento de PASCAL OBJ e a implementação de um pré-processador para a linguagem.

2 O Paradigma de Programação Orientado a Objetos

De acordo com as idéias introduzidas por Smalltalk, o Paradigma de Programação Orientado a Objetos possui cinco conceitos básicos: objeto, método, mensagem, classe e instância [Gol89].

Um *objeto* é um componente significativo do mundo real que é mapeado em um programa. Como exemplos de objetos temos máquinas, números, filas, pilhas, dicionários, polígonos etc. A abstração de um objeto do mundo real em um programa consiste em uma área de memória, contendo os *atributos* desse objeto e um conjunto de operações ou *métodos* que o objeto é capaz de realizar. Uma requisição para que um objeto realize uma de suas operações é feita enviando-se a esse objeto uma *mensagem*.

Todos objetos são membros de uma *classe*, onde são descritas as características (atributos e métodos) dos objetos dessa classe. A classe **Automóvel**, por exemplo, pode descrever os atributos (marca, cor, placa etc) e métodos (acelerar, frear, trocar de marcha etc) dos objetos automóveis. Um objeto é sempre uma *instância* de uma classe. Por exemplo, o automóvel Fiat Uno, branco, placa GX-1750 etc é uma instância da classe **Automóvel**.

A POO baseia-se ainda em outros três importantes conceitos:

- Encapsulamento, pelo qual detalhes de implementação de um objeto não são visíveis fora do seu escopo. A emissão de mensagens é a única forma de comunicação entre objetos, sendo que uma mensagem define apenas qual operação o objeto deve executar e não como executá-la. A idéia de Encapsulamento, surgida da teoria de Tipos Abstratos de Dados (TAD), introduz na linguagem os conceitos de *modularidade*, *abstração de dados* e *ocultamento de informações* (*information hiding*).

- Herança, pelo qual uma classe *herda* propriedades (atributos e métodos) de uma outra classe, chamada de sua *superclasse*. Permite que na definição de uma classe especifique-se apenas as características que a diferenciam de sua superclasse, viabilizando assim a *reusabilidade e a extensibilidade de código*.

- Polimorfismo, pelo qual um objeto pode, em tempo de execução, referir-se a instâncias de mais de uma classe. Do conceito de polimorfismo decorre que uma LOO deve suportar *ligação dinâmica* (*dynamic binding*) de mensagens a métodos.

3 PASCAL OBJ e Outras Extensões Orientadas a Objetos

Como afirmado anteriormente, PASCAL OBJ pode ser classificada como uma LOO híbrida. Essa estrutura híbrida possui duas vantagens:

- Possibilidade de reutilização de qualquer rotina originalmente desenvolvida para Pascal padrão, uma vez que a extensão produzida é um "super-conjunto verdadeiro" de Pascal.

- Facilidade de aprendizado, principalmente para aqueles programadores com domímio de Pascal. Estima-se que nesse caso uma semana seja suficiente para que todos os novos conceitos da linguagem sejam assimilados. Essa é uma grande vantagem sobre outras linguagens orientadas a objetos, que normalmente possuem uma curva de aprendizagem bastante baixa. Em Smalltalk, por exemplo, [Gol89] estima que de 3 a 6 meses são necessários para um bom aprendizado.

Essa última característica torna PASCAL OBJ ideal para o ensino de POO, onde ela exerceria o papel de primeira LOO a ser aprendida.

Um outra característica fundamental de PASCAL OBJ é que ela foi projetada tendo em vista um esquema de tradução baseado em pré-processamento. Essa metodologia pode ser comparada aos pré-processadores de *Fortran Estruturado* desenvolvidos na década de 70, sendo Ratfor um dos mais conhecidos deles [Ker76].

Se por um lado o esquema de pré-processamento simplifica a produção de tradutores para PASCAL OBJ, ele limitou um pouco os recursos que foram introduzidos na linguagem, pois não bastava simplesmente definir a sintaxe e a semântica de sua parte orientada a objetos. Era necessário também definir como os comandos dessa parte seriam transformados em comandos de Pascal padrão. Uma ênfase especial foi dada no uso de Pascal padrão tal como definido em [Jen88], de modo que o código Pascal gerado pelo pré-processador possa vir a ser compilado pelo maior número possível de compiladores.

Pré-processadores para extensões orientadas a objetos de outras linguagens que não Pascal já foram produzidos. Classes [Str83] e OOPC (*Object Oriented Pre-Compiler*) [Cox83] são dois exemplos, sendo ambos extensões da linguagem C. Em Classes, a ênfase é dada na definição e uso de *tipos abstratos de dados*. Em OOPC, procura-se simular conceitos típicos de Smalltalk, mantendo-se a sintaxe da linguagem C. Esses dois pré-processadores evoluíram posteriormente para linguagens não mais baseadas em pré-processamento. Classes deu origem à linguagem C++ e OOPC à linguagem Objective-C.

Para a linguagem Pascal, já foi proposta uma disciplina para programação orientada a objetos, descrita em [Jac87]. Pela disciplina de Jacky & Kalet, os conceitos de POO, como classes, objetos e métodos, são mapeados em comandos e estruturas típicos de Pascal padrão. Esse mapeamento é realizado pelo próprio programador, pois a disciplina não introduz nenhum recurso sintático na linguagem.

Posteriormente, foi proposta por Vecchio, em sua tese de mestrado, uma extensão de Pascal, chamada Pastalk [Vec89], inspirada nas idéias de Smalltalk e propondo alguns aperfeiçoamentos na disciplina de Jacket & Kalet. O principal aperfeiçoamento foi a introdução de alguns recursos sintáticos na linguagem, a fim de eliminar detalhes de implementação que antes ficavam a cargo do programador. Em Pastalk, esses recursos sintáticos são expandidos por um macro-expansor em comandos de Turbo Pascal 4.0. No entanto, o poder de Pastalk ainda permaneceu limitado pelo fato de ser originalmente baseado em uma disciplina para POO. Além disso, a tradução de Pastalk em comandos de Turbo Pascal 4.0, usando recursos tipícos dessa extensão, diminui a portabilidade de seus programas.

A sintaxe da parte orientada a objetos de PASCAL OBJ foi inspirada em duas LOO baseadas em Pascal: Object Pascal e Turbo Pascal 5.5. PASCAL OBJ incorpora também muitos conceitos de Smalltalk, procurando principalmente adotar a sua terminologia.

4 A Linguagem PASCAL OBJ

Nessa seção, descreve-se de forma sucinta os principais comandos e estruturas da parte orientada a objetos de PASCAL OBJ. Uma descrição detalhada pode ser encontrada no relatório de definição da linguagem [Bas92].

4.1 Classes, Objetos, Atributos, Métodos e Mensagens

Em PASCAL OBJ, *objetos* consistem em uma estrutura de dados semelhante aos registros de Pascal. Assim como um registro possui campos, um objeto possui *atributos*, que descrevem o seu estado. Diferentemente de registros, no entanto, um objeto possui um conjunto de operações que é capaz de realizar. Essas operações são implementadas por subprogramas chamados de *métodos*. A solicitação para que um objeto execute um de seus métodos é feita enviando a esse objeto uma *mensagem*. Todo objeto é uma instância de uma *classe*, onde são especificados os seus atributos e métodos.

A declaração de classes em PASCAL OBJ é feita em uma seção especial, designada pela palavra-chave **class**. A declaração de classes deve vir logo após a declaração de tipos.

Suponha um programa educativo para ensino de geometria plana. Certamente, nesse programa é necesário armazenar dados e executar operações sobre polígonos. Em PASCAL OBJ, esses poligonos podem ser representados como objetos da seguinte classe:

```
class
  Poligono =  subclass (Object)
                n: integer;              (* numero de lados *)
                method Inicializar (n2: integer);
                method ObterN (var n2: integer);
                method NumDiagonais (var d: integer);
              end;
```

Objetos de uma classe, isto é, instâncias dessa classe, são declarados em uma seção incorporada a PASCAL OBJ designada pela palavra-chave **obj**. Essa seção deve vir logo após a seção de variáveis. O exemplo abaixo mostra a declaração de objetos da classe *Poligono*:

```
obj
  umPoligono: Poligono;
  p1, p2, p3: Poligono;
```

Em uma classe, são declarados os *atributos* e os *métodos* dos objetos dessa classe. Objetos da classe Poligono, por exemplo, possuem o atributo *n* e os métodos *Inicializar*, *ObterN* e *NumDiagonais*. A declaração de atributos deve vir sempre antes da declaração de métodos.

A declaração de uma classe especifica apenas o cabeçalho de seus métodos. A definição completa de um método é feita juntamente com a definição de subprogramas em Pascal, no nível sintático do programa principal, sendo o método qualificado com a classe a que pertence.

Mostra-se abaixo a definição do método *Poligono.NumDiagonais*:

```
method Poligono.NumDiagonais (var d: integer);
         (* Calcula o numero de diagonais de um poligono *)
begin
 d:= n * (n-3) div 2
end;
```

4.1.1 Enviando Mensagens a um Objeto

A solicitação para que um objeto realize uma de suas operações, isto é, execute um de seus métodos, é feita enviando a esse objeto uma *mensagem*. Mensagens correspondem, portanto, a chamada de subprogramas em Pascal. Em PASCAL OBJ, mensagens possuem a seguinte sintaxe: *receptor.seletor*, onde *receptor* especifica o objeto que receberá a mensagem e *seletor*, a mensagem que será enviada.

A mensagem abaixo, por exemplo, inicializa *umPoligono* com seu número de lados:

```
umPoligono.Inicializar (3)
```

4.1.2 Criação de Objetos

A declaração de um objeto através da cláusula **obj**, mostrada anteriormente, não cria um objeto em tempo de execução, o que somente ocorre enviando a esse objeto a mensagem pré-declarada *new*, como no exemplo abaixo:

```
umPoligono.new
```

Vê-se, portanto, que a instanciação de um objeto em PASCAL OBJ envolve a declaração e a criação desse objeto. A declaração é feita em tempo de compilação e a criação em tempo de execução. Essa estratégia deve-se ao fato de objetos em PASCAL OBJ serem sempre alocados dinamicamente no *heap*. Na verdade, quando se declara um objeto na cláusula **obj**, está sendo declarado um ponteiro (referência) para a área de memória onde os atributos desse objeto serão armazenados. Essa área de memória é alocada enviando-se ao objeto a mensagem *new*.

Como objetos são alocados dinamicamente e a linguagem não dispõe de nenhum mecanismo de *coleta de lixo* (*garbage collection*), cabe ao programador liberar a área de memória alocada a um objeto. Isso é feito enviando a esse objeto a mensagem pré-declarada *dispose*, como mostra o exemplo abaixo:

```
umPoligono.dispose
```

4.1.3 Acessando os Atributos de um Objeto

Os atributos de um objeto somente são acessíveis no interior de métodos da classe desse objeto. Em um método, os atributos são referenciados do mesmo modo que variáveis de Pascal, sendo que subentende-se que esses atributos pertencem ao objeto receptor da mensagem associada ao método. Não há necessidade, portanto, de qualificar referências a atributos. O acesso a um atributo fora do escopo de um método somente pode ser feito enviando-se uma mensagem a um método que apenas retorne ou altere o valor desse atributo.

4.1.4 O Parâmetro *self*

Em todo método, há um parâmetro implícito, de nome *self*, que se refere ao receptor do método. Normalmente, *self* é usado quando, no interior de um método, deseja-se enviar uma mensagem ao receptor da mensagem associada a esse método. Esse tipo de mensagem teria a seguinte forma: *self.seletor*. No método abaixo, usa-se *self* para calcular o número de diagonais do objeto receptor:

```
method Poligono.MetodoX;
begin
  ............
  self.NumDiagonais (d);       (* num. diagonais do receptor *)
end;
```

4.1.5 Compatibilidade para Atribuição e para Operação Relacional

Atribuição e passagem de parâmetros envolvendo objetos em PASCAL OBJ devem obedecer à seguinte regra: Um objeto *y* da classe *C1* é compatível para atribuição com um objeto *x* (*x:= y*) se x também é da classe *C1*. A atribuição não envolve cópia de atributos desses objetos, sendo apenas uma atribuição de ponteiros.

Semelhantemente ao comando de atribuição, as operações relacionais = (igualdade) e <> (diferença), as únicas que podem ser realizadas entre objetos, trabalham com referências, isto é, $obj1 = obj2$ se eles ocupam a mesma área de memória e $obj1 <> obj2$ se ocupam áreas de memórias distintas, independente do fato de os valores de seus atributos serem os mesmos.

4.1.6 Passagem de Parâmetros

A passagem de objetos como parâmetros é indicada pela palavra-chave **obj**, antes da lista de parâmetros formais (semelhante à palavra **var** no caso de passagem por referência). Esse tipo de passagem de parâmetros, denominado em PASCAL OBJ de *chamada por objeto*, indica que está sendo passado um ponteiro para a área de memória reservada ao objeto. Com isso, um subprograma sempre pode alterar o estado de um objeto recebido como parâmetro.

4.1.7 Escopo

A declaração de uma classe e de seus métodos deve ser sempre global, isto é, classes e métodos só podem ser declarados no programa principal. Objetos podem ser declarados localmente a um subprograma, obecedendo, desse modo, às regras usuais de escopo de Pascal. Somente pode-se enviar uma mensagem a um objeto se o método correspondente já tiver sido definido anteriormente no programa. A opção **forward** pode ser usada da mesma forma que em Pascal.

4.2 Herança

Suponha que no mesmo programa para ensino de geometria plana surja a necessidade de representar um triângulo. Uma abordagem natural é representar um triângulo como um polígono, acrescido de alguma informação extra para distingui-lo dos demais polígonos. O mecanismo de herança possibilita declarar triângulo como uma *subclasse* de polígono, compartilhando todos atributos e métodos de polígonos e acrescentando novos, específicos de triângulos. Em PASCAL OBJ essa declaração seria da seguinte forma:

```
class
  Triangulo = subclass (Poligono)
                a, b, c: real;      (* lados do triangulo *)
                method new (a2, b2, c2: real);
                method ObterLados (var a2, b2, c2: real);
                method Perimetro (var p: real)
                method Area (var s: real);
              end;
```

Nesse caso, a declaração da classe *Triangulo* especifica que ela é um *subclasse* da classe *Poligono*. Da mesma forma, *Poligono* é a *superclasse* de *Triangulo*. PASCAL OBJ provê apenas *Herança Simples*, isto é, um objeto possui apenas uma superclasse.

O mecanismo de herança dá origem a uma hierarquia de classes em forma de árvore. A raiz dessa árvore é uma classe pré-declarada de nome *Object*.

Ao se enviar uma mensagem a um objeto, o método correspondente é procurado primeiro dentre os métodos da classe do objeto. Se não for encontrado, procura-se então dentre os métodos da superclasse da classe do objeto, dentre os métodos da super-superclasse da classe do objeto e assim sucessivamente. Apenas emite-se uma mensagem de erro quando o método não for encontrado na classe pré-definida *Object*.

Veja o seguinte exemplo:

```
umTriangulo.NumDiagonais
```

Como a classe *Triangulo* não possui um método *NumDiagonais*, procura-se na superclasse de *Triangulo*, no caso *Poligono*, onde o método é encontrado e executado.

Esse exemplo ilustra uma das vantagens do mecanismo de herança: a *reutilizaçao de código*. Suponha que *Poligono* possua diversos descendentes (Triangulos, Quadrados, Pentagonos etc). Caso não existisse herança, haveria necessidade de escrever um método *NumDiagonais* para toda classe descendente de *Poligono*. Com o mecanismo de herança, essa duplicação é evitada, pois todas classes descendentes "usam" o método *NumDiagonais* de *Poligono*.

4.2.1 Pré-declaração de Atributos e Métodos

Em PASCAL OBJ a pré-declaração de um atributo ou método é efetuada incorporando-os à classe pré-declarada *Object*. Como todas as classes de um programa são descendentes de *Object*, elas herdam seus atributos e métodos. Dois métodos, *new* e *dispose*, são pré-declarados em PASCAL OBJ. Não há atributos pré-declarados em PASCAL OBJ.

4.2.2 Redefinição de Atributos e Métodos

Pode-se em uma classe redefinir um atributo ou método de uma superclasse. Isso é feito simplesmente redeclarando o atributo ou método nessa classe. Uma redefinição bastante comum é a dos métodos *new* e *dispose*. Muitas vezes esses métodos são redefinidos a fim de inicializar instâncias do objeto a ser criado, no caso do método *new*, ou para executar alguma rotina antes de liberar a área de memória alocada a um objeto, no caso do método *dispose*.

4.2.3 A variável super

Todo método pode fazer uso de uma pseudo-variável de nome *super*. Essa variável, assim como *self*, refere-se ao receptor do método. No entanto, quando uma mensagem tem *super* como receptor, a busca pela classe onde método foi declarado não começa dentre os métodos da classe do objeto receptor e sim dentre os métodos da superclasse do objeto receptor. O uso de *super* é mostrado a seguir na redefinição do método *new* da classe *Poligono*.

```
method Poligono.new (n2: integer);
         (* Inicializa um poligono *)
begin
  super.new;    (* cria objeto da superclasse Object *)
  n:= n2;       (* inicializa instancia de Poligono *)
end;
```

4.3 Encapsulamento

Como afirmado anteriormente, em PASCAL OBJ os atributos de um objeto só podem ser lidos ou modificados pelos métodos desse objeto. Essa limitação do escopo de um atributo constitui o **Conceito de Encapsulamento** de linguagens orientadas a objetos.

Para permitir o acesso a atributos fora do escopo de um método, o programador deve incorporar a suas classes métodos que apenas retornem o valor de certos atributos, como é o caso dos métodos *ObterN* e *ObterLados* dos objetos das classes *Poligono* e *Triangulo*, respectivamente. Sendo assim, quando for, por exemplo, necessário obter o número de lados de um polígono, deve-se enviar a ele a mensagem *ObterN*, da seguinte forma: `umPoligono.ObterN (n)`.

O encapsulamento torna mais fácil alterações em um programa, pois desde que se mantenha inalterado os cabeçalhos dos métodos de uma classe, a implementação desses métodos pode ser livremente modificada, sem que nenhuma mensagem enviada a objetos dessa classe tenha que ser alterada.

4.4 Polimorfismo

Em linguagens orientadas a objeto, o termo *polimorfismo* designa a propriedade de um objeto tornar-se instância de várias classes, em tempo de execução.

Em LOO tipadas, como Eiffel, o polimorfismo é implementado acrescentando uma regra de compatibilidade para atribuição. Essa regra possibilita que uma atribuição da forma $x := y$, x e y objetos, seja válida se a classe de y for uma subclasse da classe de x. Essa forma de polimorfismo implica que a associação entre uma mensagem e o método correspondente seja feita em tempo de execução, isto é, a linguagem deve suportar *ligação dinâmica* (*dynamic binding)* de mensagem/método. O exemplo abaixo ilustra essa situação:

```
obj1.mensagem1; (* executa metodo1 da classe de obj1 *)
obj1:= obj2;
obj1.mensagem1; (* executa metodo1 da classe de obj2 *)
```

Como Pascal não suporta ligação dinâmica, nem dispõe de meios para que esse recurso seja implementado eficientemente, PASCAL OBJ não implementa polimorfismo em toda a sua potencialidade. Em PASCAL OBJ, a associação mensagem/método é estática, de acordo com a classe do objeto receptor. Isso não impede, no entanto, que objetos de classes distintas respondam de maneira distinta a uma mesma mensagem, como mostra o exemplo abaixo:

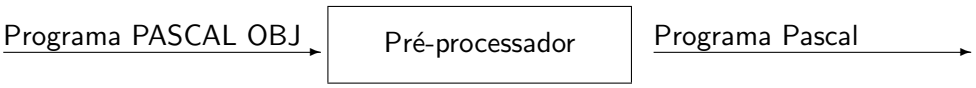
```
obj
  p: Pilha;
  q: Lista;

 begin
   p.imprimir;    (* imprime elementos de uma pilha *)
   q.imprimir     (* imprime elementos de uma lista *)
 end;
```

Nesse exemplo, a mesma mensagem *imprimir* executa métodos diferentes, conforme seja enviada a um objeto da classe *Pilha* ou da classe *Lista*. Essa "forma estática" de polimorfismo é semelhante ao recurso denominado *sobrecarga de operadores* (*operator overloading*) em ADA.

5 Pré-Processamento de PASCAL OBJ

PASCAL OBJ foi projetada tendo em vista um esquema de tradução baseado em pré-processamento. Nessa forma de tradução, os recursos incorporados à linguagem a fim de suportar o paradigma de orientação a objetos são "expandidos" por um pré-processador em recursos de Pascal padrão, como mostra a figura abaixo:



Foi definido como cada estrutura da parte orientada a objetos de PASCAL OBJ é traduzida para Pascal padrão. Isso envolve basicamente o pré-processamento de classes, objetos, métodos e mensagens. Nesse artigo, no entanto, não será descrito o esquema de pré-processamento de PASCAL OBJ. Uma descrição completa desse esquema pode ser encontrada em [Bas92].

A fim de facilitar a implementação do pré-processador, foi definido que as linhas de programa que usam recursos próprios de PASCAL OBJ devem possuir um caráter # na primeira coluna. Apesar de ser um desconforto para o programador, essa estratégia simplifica bastante a implementação do pré-processador.

Um protótipo do PPOBJ (Pré-processador de PASCAL OBJ) foi desenvolvido para o sistema operacional Unix. Esse protótipo foi implementado em C [Ker78], usando-se o compilador padrão cc que acompanha o Unix. Usou-se também duas ferramentas auxiliares para produção de compiladores: o gerador de analisadores léxicos Lex [Les] e o gerador de analisadores sintáticos Yacc [Joh]. O uso dessas duas ferramentas acelerou bastante a implementação do pré-processador. O código pré-processado foi testado usando-se o compilador SVS Pascal [SVS89], tendo sido satisfatórios os resultados obtidos.

6 Conclusão

Apesar de não possuir o mesmo poder de expressão que linguagens como Smalltalk e Eiffel, a extensão proposta nesse relatório pode ser útil em diversas aplicações de pequeno e médio porte. Além disso, a linguagem é adequada para o ensino de programação orientada a objetos, dada à sua simplicidade.

A definição dos conceitos de objeto, classe, método, mensagem e instância da extensão mostrou-se bastante completa, estando disponíveis as principais características desses conceitos. Dos conceitos de encapsulamento, herança e polimorfismo, PASCAL OBJ define adequadamente apenas os dois primeiros. O conceito de polimorfismo, devido às restrições impostas pelo pré-processamento em Pascal padrão, foi definido de forma parcial, sendo mais correto afirmar que a linguagem suporta sobrecarga de operadores e não polimorfismo. Apesar dessas deficiências, o poder de PASCAL OBJ é superior ao de metodologias para POO sem suporte sintático em Pascal, como as descritas em [Jac87] e [Vec89], e é apenas um pouco inferior ao de outras LOO baseadas em Pascal, como Object Pascal e Turbo Pascal 5.5.

Na modelagem de algumas aplicações em PASCAL OBJ, uma deficiência notada foi a falta de ortogonalidade entre os conceitos de POO introduzidos na linguagem e conceitos típicos de Pascal. Esse fato é notado principalmente na impossibilidade de declarar estruturas com conceitos de ambos os paradigmas, como, por exemplo, um vetor de objetos. O motivo dessa falta de ortogonalidade foi a intenção desse o início da definição da linguagem de simplificar o seu pré-processador. No entanto, em uma versão futura, deve-se tornar possível a construção de estruturas envolvendo conceitos dos dois paradigmas, mesmo que isso torne o seu pré-processamento mais complexo.

Tendo como base a linguagem desenvolvida nesse projeto, um projeto futuro pode ser a definição de uma LOO genuína, sem o compromisso de ser uma extensão de Pascal padrão ou de ser pré-processada em

comandos dessa linguagem. Essa nova linguagem deverá certamente incorporar o conceito de polimorfismo e permitir a compilação em separado de classes.

References

[Bor89] Borland. *Turbo Pascal 5.5 object oriented programming guide*. Borland International, 1989.

[Cox83] Cox, B.J. *The object oriented pre-compiler*. SIGPLAN Notices 18, (1):15-22, January 1983.

[Dig88] Digitalk. *Smalltalk/V 286 tutorial and programming handbokk*. Digitalk Inc., 1988.

[Ghe85] Ghezzi, C. e Jazayeri, M. *Conceitos de linguagens de programação*. Campus, 1985.

[Gol89] Goldberg, A. and Robson, D. *Smalltalk the language*. Addison-Wesley, 1989.

[Jac87] Jacky, J.P. and Kalet, I.J. *An object oriented discipline for standart pascal*. Comm. ACM 30, (9):772-776, September 1987.

[Jen88] Jensen, K. and Wirth, N. *PASCAL ISO manual do usuário e relatório*. Campus, 1988.

[Joh] Johnson, S.C. *Yacc: yet another compiler-compiler*.

[Ker76] Kernighan, K. and Plauger, P.J. *Software tools*. Addison-Wesley, 1976.

[Ker78] Kernighan, K. and Ritchie, D.M. *The C programming language*. Prentice-Hall, 1978.

[Kor90] Korson, T. and McGregor, J.D. *Understanding object-oriented: a unifying paradigm*. Comm. ACM 33,(9):41-60, September 1990.

[Les] Lesk, M.E. and Schmidt, E. *Lex - a lexical analyzer generator*.

[Mey88] Meyer, B. *Object oriented software construction*. Prentice-Hall, 1988.

[Str83] Stroustrup, B. *Classes: an abstract data type facility for the C language*. SIGPLAN Notices 17, (1):42-51, January 1982.

[Str86b] Stroustrup, B. *The C++ programming language*. Addison-Wesley, 1986.

[Str88] Stroustrup, B. *What is object-oriented programming?*. IEEE Software, May 1988.

[SVS89] Silicon Valley Software *SVS Pascal language reference manual*. November, 1989.

[Tak90] Takahashi, T., Liesenberg, H.K.E, Xavier, D.T. *Programação orientada a objetos: uma visão integrado do paradigma de objetos*. VII Escola de Computação, São Paulo, 1990.

[Bas92] Bastos, L.N. e Valente, M.T.O. PASCAL OBJ: uma extensão de pascal orientada a objetos. Relatório Técnico DMA 001/92-CC, Depto de Matemática, UFV, 1992.

[Vec89] Vecchio, L.H.A *Sobre o desenvolvimento de sistemas em linguagens procedimentais utilizando o paradigma de programação orientado a objetos*. Dissertação de Mestrado, Depto de Ciência da Computação, UFMG, 1988.