Aram: uma linguagem de programação para o ambiente *Virtuosi*

Aron Borges
Projeto Final II
Bacharelado em Ciência da Computação
Pontifícia Universidade Católica do Paraná
aron.borges@icet.pucpr.br

Resumo

Nos últimos anos, o paradigma de programação mais utilizado para a construção de software foi o orientado a objetos. Cada vez mais tem aumentado a pesquisa sobre programação orientada a objetos, sistemas distribuídos e linguagens que possuem suporte à reflexão computacional. Contudo, no "código objeto" gerado pelas linguagens atuais (código nativo de máquina ou até "byte-code" no caso de Java[3]) tem-se uma perda semântica e isso limita a reflexão que se pode ter. O código gerado de acordo com o meta-modelo[1] *do ambiente* Virtuosi*[2] não tem perda alguma de semântica. A linguagem Aram é um subconjunto da linguagem Java e segue os conceitos do meta-modelo da* Virtuosi. *A linguagem abrange um conjunto mínimo de funcionalidades que possibilite aos futuros usuários da mesma implementar um código que atenda aos requisitos funcionais de um sistema, mesmo que para isso ele tenha que fazer combinações de sentenças para alcançar os objetivos desejados na representação semântica escrita no código fonte.*

Palavras-chaves: {Programação Orientada a Objetos, Máquinas Virtuais, Compiladores}.

1 Indrodução

Está em desenvolvimento o Projeto *Virtuosi* sob coordenação do Professor Alcides Calsavara no PPGIA-PUCPR[2], o qual visa construir uma plataforma de execução distribuída de sistemas de software orientados a objetos. O código interpretado por uma máquina virtual da *Virtuosi* consiste em uma coleção de objetos correspondentes a definições de classes, devidamente interligados, a fim de preservar toda a semântica da aplicação em tempo de execução, e assim, aumentar o potencial de reflexão computacional. Os objetos que representam uma classe são instanciados a partir de um meta-modelo[4], isto é um modelo de classes que representam todos os conceitos de orientação a objetos que podem estar presentes em uma aplicação. Uma coleção de objetos que representa uma certa aplicação é denominada "árvore de programa".

Aram é uma linguagem de programação construída com o intuito de facilitar o aprendizado do desenvolvimento orientado a objetos. Ela tem características de uma linguagem orientada a objetos, seguindo as definições do meta-modelo da *Virtuosi*.

A linguagem proposta não permite o uso de recursos de programação imperativa, obrigando a utilização rigorosa do paradigma de programação orientada a objetos. Pois, foi observado nas aulas de monitoria de programação orientada a objetos que os alunos

[1] Modelo dos objetos que representam o código executável

[2] Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná

iniciantes nesse paradigma, quase todas as vezes que não conseguiam encontrar um modelo orientado a objetos para um problema qualquer, acabavam utilizando soluções imperativas para resolver esse problema.

Nesse primeiro trabalho, Aram é um subconjunto da linguagem Java[3], contendo um número mínimo de características que possibilite um programador implementar um código que atenda aos requisitos funcionais de um sistema.

Procurou-se definir a sintaxe a mais próxima possível de Java, devido ao fato que na grande maioria dos casos a primeira linguagem ensinada nas universidades é Java, C/C++[6] ou Pascal. Além disso, Java tem muito mais aceitação no mercado e no mundo acadêmico que outras linguagens, tal como Eiffel[5], mesmo essa sendo considerada a linguagem atual mais "*pura*" orientada a objetos.

Este artigo está organizado da seguinte maneira: na seção 2 são mostrados os trabalhos relacionados, na seção 3 uma breve descrição do escopo da linguagem é apresentado, na seção 4 são mostrados os elementos léxicos e na 5 os elementos gramaticais, na seção 6 são mostradas as possíveis sentenças da linguagem, na seção 7 é apresentada uma breve descrição do compilador e finalmente na seção 8 as conclusões deste trabalho são apresentadas.

2 Trabalhos Relacionados

Para especificar a linguagem foram utilizadas as especificações das linguagens C++[3][6], Java[3] Eiffel[5]. Aproveitando caracteristicas que cada uma tinha de melhor.

3 Escopo da Linguagem

Essa seção mostra os elementos que a linguagem Aram possui e as restrições que ela tem.

3.1 Elementos da linguagem

A linguagem Aram abrange os seguintes ítens:

- classes;
- atributos;
- métodos;
- *if* e *else*;
- *while*;
- encapsulamento baseado em lista de exportações;
- váriaveis locais;
- comparadores lógicos - E ($\wedge$), OU ($\vee$), Negação ($!$);
- comparadores relacionais - Igualdade ($\equiv$) e Diferença ($\neq$);
- atribuição;
- chamada de métodos
- literais.

Foram implementadas as classes "*Integer*", "*Character*", "*Boolean*"e "*String*", disponibilizando classes básicas ao programador.

3.2 Restrições

Os ítens mencionados abaixo não fazem parte do escopo da linguagem nessa primeira versão, porém poderão ser adicionados na linguagem em trabalhos futuros:

- inner classes;
- herança;
- polimorfismo;
- *for*;
- pacotes;
- variáveis globais (não existe esse conceito no meta-modelo da *Virtuosi*);
- interfaces e classes abstratas;

[3]Especificação feita pela IBM

- vetores (arrays);
- pacotes;
- tratamento de excepção;
- “design by contract”[5].

Uma das maiores perdas nas restrições da linguagem é a retirada da herança entre objetos. Pois com isso, uma das características mais importantes do paradigma orientado a objetos não será abrangida: a extensibilidade ou especializações entre os relacionamentos das classes.

Nessa primeira versão da linguagem ela abrange apenas tipos numéricos inteiros, não trabalhando com tipos primitivos como ponto flutuante.

4 Elementos Léxicos

Um elemento léxico é um caractere ou um grupo de caracteres que aparece no código fonte, onde esse elemento tem um significado. Um elemento léxico geralmente é chamado de *token* da linguagem. Esta seção discute sobre os elementos básicos da estrutura léxica da linguagem.

Os programas em Aram são escritos usando um conjunto de caracteres ASCII (American Standard Code for Information Interchange).

Tokens:

A linguagem Aram pode ter os seguintes tipos de *tokens*:

- Operadores;
- Palavras-reservadas;
- Literais;
- Identificadores.

Esses tipos de elementos léxicos são discutidos nas sub-seções (§4.1), (§4.2), (§4.3) e (§4.4), respectivamente.

4.1 Operadores Lógicos

A linguagem Aram é composta apenas de cinco operadores, sendo três deles lógicos e dois relacionais. Esses operadores estão divididos em dois grupos: operadores in-fixados e pré-fixados. Que podem ser vistos nas Tabelas 1 e 2.

Operador	Descrição
&&	Operador lógico *E* ($\wedge$).
||	Operador lógico *OU* ($\vee$).
==	Operador relacional *igual*.
!=	Operador relacional *diferente*.

Tabela 1. Operadores in-fixados

Operador	Descrição
!	Operador lógico *NOT* ($\neg$).

Tabela 2. Operadores pré-fixados

Diferente das linguagens C++ e Java, que contêm um número elevado de operadores (como, por exemplo, 37 da linguagem Java), para deixar a linguagem rigorosamente orientada a objetos, Aram procurou definir um número mínimo de operadores básicos, obrigando o programador implementar métodos para resolver operações entre elementos de uma mesma classe.

Um exemplo prático seria o uso da operação de maior entre dois números inteiros (em C++ seria usado o operador $>$ para verificar essa condição). Em Aram, procurando seguir a programação orientada a objetos, o programador teria que fazer um operação de desvio condicional (§5.3.5) para fazer essa comparação, obrigando-o usar um método de uma classe em vez de um operador. Os códigos da Figura 1 exemplificam esse caso.

4.2 Palavras-reservadas

Palavras-reservadas[4] são identificadores especiais reservados da linguagem para uso especial.

[4] Do inglês keyword.

Código escrito em C++:

```
int x = 3;
int y = 11;

if(x > y) {
      ...
}
```

Código escrito em Aram:

```
Inteiro x = Inteiro.make(3);
Inteiro y = Inteiro.make(11);

if(x.maior(y)) {
      ...
}
```

Figura 1. Diferença do uso de operadores entre C++ e Aram.

As seqüências de caracteres mostrada na Tabela 3 são reservadas para serem usadas como palavras-reservadas da linguagem e não podem ser usadadas como nome de idendificadores (§4.4).

action	export
association	if
class	literal
composition	method
constructor	null
datablock	return
default	skip
else	this
enum	void
execute	while

Tabela 3. Palavras-reservadas da linguagem

4.3 Literais

Um literal é uma representação no código fonte de um valor que pode ser do tipo numérico (§4.3.1), caractere (§4.3.2) ou *String* (§4.3.3).

4.3.1 Literais Inteiros

Os literais inteiros são números cardinais com 32-bits. Os literais inteiros podem ser expressados na forma decimal (base 10). Nessa primeira versão não serão trados inteiros na forma hexadecimal (base 16) ou octal (base 8).

Um numeral decimal pode ser um simples caractere ASCII 0, representando o inteiro zero, ou consistir de um dígito ASCII 1 até 9, opcionalmente seguido de um ou mais dígitos ASCII de 0 até 9. representando um inteiro positivo.

O valor máximo que um literal inteiro decimal pode ter é 2147483647 (2^{31}). Ou seja, todo literal pode ter um valor entre 0 e 2147483647. O bit de número 32 está reservado para trabalhos futuros como operador de sinal.

Exemplos de Literais Inteiros:

```
0    2    372    2004
```

4.3.2 Literais Caracteres

Um caractere literal é expressado por um caractere ou uma seqüência de escape, circundado do caractere plica.

Caracteres de escape:

Pode ser representado qualquer tipo de caractere usando uma seqüência de escape. Esse tipo de caractere é usado para colocar caracteres que não podem ser impressos em caracteres ou literais do tipo *String* (§4.3.3). Por exemplo, pode-se usar uma seqüência para colocar o caractere de tabulação, retorno de carro e retorno de espaço em um fluxo de saída de vídeo.

Uma seqüência de escape contém o símbolo de "barra invertida"[5] (\) seguido por um caractere de escape. Observando que a continuação de uma linha não é uma seqüência de escape.

Os tipos de seqüência de escape que podem existir na linguagem são mostrados na Tabela 4.

[5] Do inglês *Backslash*.

Escape	Caractere correspondente
\b	Apagar caractere anterior.
\f	Nova página.
\n	Nova linha.
\r	Retorno de carro.
\t	Tabulação horizontal.
\'	Plica.
\"	Aspas.
\\	Barra Invertida.

Tabela 4. Caracteres de Escape

O valor de uma seqüência de escape representa um membro de um conjunto de caracteres usados em tempo de execução; Essa seqüência é traduzida para o código de caractere ASCII correspondente. Por exemplo, o caractere de escape `\n` é traduzido para o caractere de número 13 da tabelas ASCII de símbolos.

Exemplos de Literais Caracter:

```
'a'    '1'   '_'   'Z'   '\t'
```

4.3.3 Literais *String*

Um literal *string* consiste em zero ou mais caracteres ou seqüências de escape circundados pelo caractere aspas.

Exemplos de Literais *String*:

"10" "Isto é um texto" ''\"

4.3.4 Literal null

Para que uma referência de um objeto ou de um *datablock*(§5.2.3) possa ser nula (vazia), é preciso acrescentar à linguagem o literal *null*. O literal nulo contém apenas um valor e é único na linguagem.

4.4 Identificadores

Um identificador é uma seqüência ilimitada de letras e dígitos, porém o primeiro caracter obrigatoriamente tem que ser uma letra. Um identificador não pode ser uma palavra chave (§4.2) da linguagem, ou o literal *null*(§4.3.4).

Os identificadores são sensíveis às letras maiúsculas e minúsculas. Podem conter os caracteres alfanuméricos e dígitos e também "*undescore*".

Não serão tratados caracteres do tipo Unicode[7]. Sendo assim, idiomas que usam esse tipo de caractere (como Japonês, Chinês etc) não serão tratados nessa primeira versão da linguagem.

Exemplos de Idenficadores Válidos:

```
isLetterOrDigit    i3    MAX_VALUE
```

5 Elementos Gramaticais

Essa seção descreve detalhadamente os elementos da gramática da linguagem. Será usado o código de exemplo da Figura 2 para demonstrar a sintaxe e as características que a linguagem Aram possuí.

5.1 Classe

Uma declaração de uma classe especifica um novo tipo abstrato de dados, possibilitando ao programador criar novos tipos de dados. Uma classe pode ser composta de zero ou vários elementos. Esses elementos podem ser atributos, métodos ou ações. Onde esses itens são especificados nas Seções (§5.2), (§5.3) e (§5.3.5), respectivamente.

O Identificador na declaração de uma classe especifica o nome que essa classe possui.

É importante observar, que nesse primeira versão da linguagem, não devem existir classes com o mesmo nome em um mesmo código fonte, devido ao fato que a linguagem "ainda" não trabalha com idéia de agrupamento de classes em pacotes.

Um exemplo de código que faz a declaração de classe pode ser visto nas linhas 1, 25 e 39 da Figura 2.

5.2 Atributo

Um atributo é uma propriedade de uma classe que define uma relação entre dois elementos. Os tipos de atributos possíveis na linguagem Aram são especificados nas Seções (§5.2.1), (§5.2.2), (§5.2.3), (§5.2.4).

5.2.1 Associação

Um associação é um tipo de relação entre o objeto e o seu atributo, onde o atributo não faz parte dele. Um exemplo seria a relação entre uma pessoa a um carro, onde uma pessoa tem um carro, porém o carro não faz parte da pessoa.

Um exemplo de código pode ser visto na linha 26 da Figura 2, onde um carro pode transportar uma pessoa, porém a pessoa não faz parte desse carro.

5.2.2 Composição

Uma composição é uma relação entre o objeto e o seu atributo, onde fica explícito que o atributo desse objeto faz parte dele (está contido), diferentemente de uma associação. Esses qualificadores das relações entre os objetos e os seus atributos foi definido para poder ter uma semântica maior ao se escrever o código fonte, ficando explicito qual é o tipo de relação que eles possuem.

Um código de exemplo de declaração de uma composição pode ser vista na linha 2 da Figura 2.

5.2.3 Bloco de Dados

Um bloco dados ("*data block*") consiste em um uma referência para um conjunto de dados binários em memória que pode ser manipulado pelas chamadas de sistema. ("*system calls*").

5.2.4 Enumeração

Uma enumeração consiste em uma associação entre literais, onde define-se um conjunto de literais que fazem parte de um domínio e qual literal pode-se associar aos elementos desse conjunto. Diferente de enumeração de C++ e Java, que define um novo tipo, em Aram uma enumeração apenas possibilita fazer verificações de relações entre literais. Pois, uma vez definido o domínio pode-se saber se uma atribuição entre literais é válida, por que são conhecidos os elementos que ele pode se relacionar.

Um código de exemplo de declaração de uma enumeração pode ser vista na linha 3 da Figura 2.

5.3 Método

Um método é uma operação em um objeto que implementa um serviço que pode ser invocado por qualquer objeto, desde que sua classe tenha acesso definido na sua "lista de exportação" (§5.3.3) desse método.

Um código de exemplo de declaração de um método pode ser vista nas linhas 21 e 30 da Figura 2.

5.3.1 Lista de Parâmetros do Método

Na linguagem Aram um método pode não ter parâmetro algum ou possuir uma lista de um ou mais parâmetros, sendo que esses parâmetros só podem ser referências para classes ou literais. Não será possível definir qualquer outro tipo além das referências. Isso para obrigar o programador a sempre usar classes nas mensagens entre os objetos.

Um código de exemplo de declaração dos parâmetros de uma operação pode ser vista na linha 4 da Figura 2.

5.3.2 Tipo de Retorno

Existem dois tipos possíveis de retorno em um método, um vazio ("*void*") onde é definido assim um procedimento e o outro tipo é o retorno de uma referência para um objeto, definindo assim um método.

Um código de exemplo de declaração de uma operação que o tipo de retorno é vazio ("*void*") pode ser vista na linha 21 da Figura 2.

5.3.3 Lista de Exportação

Diferente das definições de encapsulamento de Java e C++ que são as mais utilizadas hoje em dia e conhecidas (métodos privados, públicos e protegidos). Procurou-se definir encapsulamento através de lista de exportações como as de Eiffel, por serem mais versáteis que o outro tipo de encapsulamento e não conter várias restrições que ele tem. Como por exemplo não poder especificar apenas uma classe de acesso a operação do objeto.

Para exportar uma operação possibilitando que todas as classes de um programa tenham acesso é utilizada a palavra chave "*all*".

Para restringir que apenas instâncias da mesma classe possam ter acesso a uma determinada operação é utilizada a palavra chave "*none*".

Um código de exemplo de declaração da lista de exportação pode ser vista nas linhas 27 e 40 da Figura 2.

5.3.4 Construtor

Um construtor é uma operação que possibilita a instanciação de objetos da classe.

Um código de exemplo de declaração de construtores pode ser vista nas linhas 4, 27 e 40 da Figura 2.

5.3.5 Ações

Uma ação é um tipo especial de operação que define um desvio condicional de uma sentença. Ela é o único tipo de operação que pode ser chamada em uma avaliação de expressão. Ficando assim, a cargo da instância do objeto saber se o estado dele é verdadeiro ou não, preservando assim o encapsulamento do mesmo.

Um código de exemplo de declaração de uma ação pode ser vista na linha 8 da Figura 2.

6 Sentenças da Linguagem

Todos as sentenças que podem existir em um código fonte escrito na linguagem Aram são discutidos nessa seção.

6.1 Composição de Sentenças

Para que se possam ter vários comandos em um bloco de código nas operações de uma classe é preciso introduzir na linguagem a composição de sentenças.

6.2 Declação de Variável

Podem ser declaradas como variáveis locais: classes e blocos de dados, o escopo dessas declarações fica restrito ao corpo da operação.

Um código de exemplo de declaração de variável pode ser vista nas linhas 9, 41, 42, 43 e 44 da Figura 2.

6.3 Instanciação

Parar instanciar um objeto na linguagem Aram é preciso que na declaração ou na atribuição seja invocado um construtor de uma classe.

Um código de exemplo de instanciação de uma variável pode ser vista nas linhas 41, 42 e 43 da Figura 2.

6.4 Atribuição

Pode ser feita a atribuição da referência através de identificadores, acesso a atributos(§6.5) de instâncias de outros objetos da mesma classe, invocações de métodos e no caso especial fazer uma referência nula.

Um código de exemplo de atribuição de uma variável para uma referência vazia pode ser vista nas linhas 5, 28 e 47 da Figura 2.

6.5 Acesso a Atributo

Existem situações, como por exemplo, a clonagem de objetos e seus atributos é preciso que objetos de instâncias diferentes da mesma classe possam acessar os atributos um do outro. Porém para preservar

a consistência da classe da qual estão sendo obtidas as informações, só podem se acessadas do lado direito de uma atribuição de referência (somente leitura).

6.6 Referência Nula

Para poder atribuir a uma declaração uma referência nula (vazia) é utilizado o literal *null* (§4.3.4). Com isso, um retorno de um método, uma variável local e um atributo podem referenciar o vazio.

6.7 Retorno das Ações

Em uma ação podem existir dois tipos de resultados, uma que define que a ação é verdadeira e deve ser executada e outra que a condição da ação não é verdadeira. Para isso existem duas palavras-chave que são *execute* e *skip* para tomar essas ações. Essa sentença é restrita ao escopo das ações.

Um código de exemplo de retorno de uma ação vazia ser visto nas linhas 17 e 19 da Figura 2.

Observação: Em uma ação sempre deve ser tomada pelo menos uma decisão.

6.8 Invocação de Método

Podem existir dois tipos de invocação de métodos, um que tenha um retorno, sendo assim uma função e outro que não tenha retorno algum, definindo um procedimento. Pode ser utilizado o qualificador *this* para invocar métodos da própria instância.

Um código de exemplo da invocação de um método pode ser vista na linha 46 da Figura 2.

6.8.1 Expressão

Uma avaliação de expressão só pode ser feita para comparar referências e enumerações e invocação de ações. Também podem ser utilizados os operados especificados na Seção 4.1.

6.9 Desvio Condicional

A sentença *if* permite fazer uma execução condicional desde que a avaliação de uma expressão seja verdadeira. Fazendo uma escolha entre duas sentenças, executando uma ou outra e nunca as duas.

Um código de exemplo do desvio condicional *if* pode ser visto nas linhas 10 e 16 da Figura 2.

6.10 Repetição

A sentença *while* avalia uma expressão e executa uma sentença repetidamente enquanto a avaliação dessa expressão for verdadeira. Se inicialmente a avaliação não for verdadeira a sentença não é executada nenhuma vez.

Um código de exemplo da sentença de repetição *while* pode ser visto na linha 45 da Figura 2.

6.11 Vazio

Uma sentença vazia que não faz nada pode ser feita no código fonte utilizando o ponto e vírgula.

7 Compilador

Para testar a especificação da linguagem e o mapeamento da mesma para o código objeto do meta-modelo da Virtuosi, foi implementado um compilador usando a linguagem Java e gerador de “parser”e árvores gramáticas ANTLR[1].

Com o auxilio da ferramenta, toda à parte de analise léxica e sintática foi construída a partir das regras gramáticas feitas na especificação formal. Dada a árvore gramátical gerada pelo ANTLR, foi implementados a verificação semântica do código fonte e o mapeamento para o código objeto usado na máquina virtual da Virtuosi.

8 Conclusões

Todas as fases de desenvolvimento de uma linguagem e um compilador foram implementadas, desde a primeira fase de especificação formal até a geração

de código objeto. Desenvolvendo uma linguagem ”puramente”orientada a objetos, com uma sintaxe próxima de Java e C++, com conceitos de Eiffel. Atingindo o objetivo da linguagem destinada ao ensino da programação orientada objetos, pois as linguagens com esse propósito, devem ser próximas das linguagens usadas no ambiente de trabalho.

Mesmo tendo estabelecido várias restrições no escopo da linguagem, o trabalho serve como base para uma pesquisa futura, implementando novas características.

Referências

[1] ANTLT. Another tool for language recognition. URL: http://www.antlr.org/, Maio 2004.

[2] A. Calsavara. Virtuosi: Máquinas virtuais para objetos distribuídos. Trabalho apresentado em concurso para professor titular, Pontifícia Universidade Católica do Paraná, Fevereiro 2000.

[3] J. Gosling. *The Java Language Specification*. Addison-Wesley, second edition, June 2000.

[4] A. C. Leonardo R. Nunes. Estudos sobre a concepção de uma linguagem de programação reflexiva e correspondente ambiente de execução. *Simpósio Brasileiro de Linguagens de Programação*, (5):12, Maio 2001.

[5] B. Meyer. *Object-oriented software construction*. Prentice Hall, 1997.

[6] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1993.

[7] Unicode. The unicode standard: Worldwise character encoding. URL: http://www.unicode.org/, Outubro 2003.

```
 1 class Pessoa {
 2    composition Integer peso;
 3    enum { masculino, feminino } sexo = masculino;
 4    constructor instanciar(Integer peso, literal sexo ) exports { all } {
 5        this.peso = peso;
 6        this.sexo = sexo;
 7    }
 8    action obeso() exports { all } {
 9        Integer valor;
10        if (sexo==masculino) {
11            valor = Integer.make(1);
12        }
13        else {
14            valor = Integer.make(2);
15        }
16        if (massa.gt(h))
17            execute;
18        else
19            skip;
20    }
21    method void emagreca( Integer i ) exports { all } {
22        peso.subtract(i);
23    }
24 }
25 class Taxi {
26    association Pessoa passageiro;
27    constructor instanciar() exports { Principal } {
28        passageiro = null;
29    }
30    method Boolean entrar(Pessoa p) exports { Principal } {
31        if (!p.obeso()) {
32            return Boolean.make(true);
33        }
34        else {
35            return Boolean.make(true);
36        }
37    }
38 }
39 class Principal {
40    constructor iniciar() exports { all } {
41        Taxi corsa = Taxi.instanciar();
42        Integer peso = Integer.make(3);
43        Pessoa andrea = Pessoa.instanciar( peso, masculino );
44        Boolean valor = corsa.entrar(andrea);
45        while(!valor) {
46            andrea.emagreca(1);
47            valor = corsa.entrar(andrea);
48        }
49    }
50 }
```

Figura 2. Código fonte de exemplo