Anexo A

Metamodelo da VIRTUOSI

Este documento especifica os conceitos de orientação a objeto suportados pela Virtuosi através da formalização do metamodelo da Virtuosi. Para auxiliar o entendimento dos conceitos explicados ao longo desse documento, serão utilizados trechos de código fonte de uma aplicação de software orientado a objeto escritos na linguagem Aram.

A.1 Especificação

Os conceitos de orientação a objeto suportados pela Virtuosi são formalizados através de um diagrama de classes da UML chamado de metamodelo da Virtuosi. O metamodelo da virtuosi possui classes e associações que representam os elementos encontrados em uma linguagem de programação orientada a objeto que suporte aos conceitos suportados pela Virtuosi. Por isso, as classes do metamodelo podem ser chamadas de meta-classes. Por exemplo, uma classe possui atributos; o metamodelo da Virtuosi, portanto, possui uma meta-classe para representar uma classe de aplicação, uma meta-classe para representar um atributo e através de uma associação, explicita se uma classe possui zero ou muitos atributos. O metamodelo da Virtuosi pode ser entendido como um diagrama de classes que descreve conceitos de orientação a objeto e a forma como tais conceitos se relacionam entre si.

A.1.1 Literais

A.1.1.1 Valor Literal

Um valor literal é uma seqüência de caracteres sem semântica definida. Um valor literal pode existir como:

1. parâmetro real em comandos de invocação;
2. origem da atribuição em comandos de atribuição de variáveis enumeradas;
3. segundo elemento em comandos de comparação de valor em variáveis enumeradas.

A Figura A.1 mostra as situações possíveis para o uso de valores literais.

```
class Principal
{
   constructor iniciar( ) exports all {
   ...
   Integer massa = Integer.make( 70 );// 70 é um valor literal
   }
   ...
}
...
class Boolean {
   enum { true, false } value = false;
   ...
   method void flip( ) exports all
   {
       if ( value == true )
           value = false;
       else
           value = true;
   }
   ...
}
```

Figura A.1: Código fonte em Aram mostrando os possíveis uso de um valor literal

A.1.1.2 Referência a Literal

Uma meta-classe que representa uma **referência a literal** se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como parâmetro formal;
2. como parâmetro real;
3. como origem de atribuição em um comando de atribuição de variável enumerada;
4. com uma das possibilidades para o segundo elemento de uma comparação de valor de variável enumerada.

Deve-se notar que uma referência a literal no papel de parâmetro real não pode sofrer atribuição – uma vez que não existe uma meta-classe que represente o comando de atribuição para referência a literal. Também não é possível declarar uma variável local do tipo referência a literal, visto que não existe uma meta-classe que represente o comando para declarar variáveis do tipo referência a literal.

A Figura A.2 mostra o relacionamento das meta-classes que representam valores literais e referências a literal com outros componentes do metamodelo da VIRTUOSI.

O uso de um valor literal e de uma referência a literal é detalhado na Seção A.1.7, especificamente na discussão sobre comandos de declaração de variáveis.

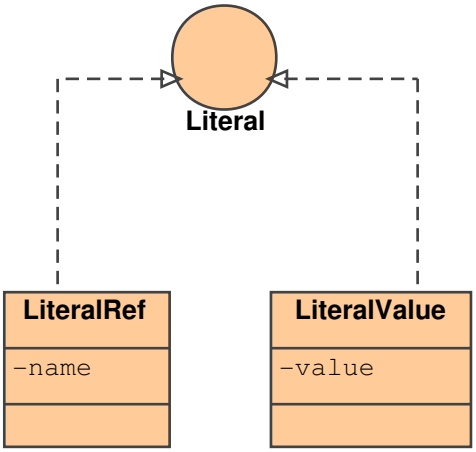

Figura A.2: Relacionamento das meta-classes que representam valores literais e referências a literal com outros componentes do metamodelo da VIRTUOSI

A.1.2 Bloco de Dados e Índice

A.1.2.1 Referência a Bloco de dados

Uma **referência a bloco de dados** consiste em uma referência para uma seqüência contígua de dados binários em memória. Esse tipo de referência é geralmente utilizado para a construção de classes pré-definidas (*Integer*, *Real*, *Boolean*, *Character* e *String*) utilizadas para compor outras classes. A referência a bloco de dados é discutida em detalhe na Seção A.1.4.

A meta-classe que representa uma referência a bloco de dados se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como parâmetro formal;
2. como parâmetro real;
3. como atributo de uma classe;
4. como atributo em um acesso a atributo bloco de dados (Este componente é explicado na Seção A.1.7, especificamente na discussão sobre o comando para atribuição de referência a objeto);
5. na comparação entre duas referência a bloco de dados;
6. na comparação entre uma referência a bloco de dados e uma referência nula;
7. como alvo da atribuição em um comando de atribuição de referência a bloco de dados;
8. como variável local;

9. como alvo da atribuição em um comando de atribuição de referência nula a bloco de dados;

10. como alvo de uma referência a índice;

11. como alvo dos muitos comandos de sistema para manipulação de blocos de dados de classes pré-definidas. Os comandos de sistema são detalhados na Seção A.1.8.1;

12. como alvo dos muitos testáveis especiais para manipulação de blocos de dados de classes pré-definidas.

A Figura A.3 mostra o relacionamento da meta-classe que representa a referência a bloco de dado com outros componentes do metamodelo da VIRTUOSI.

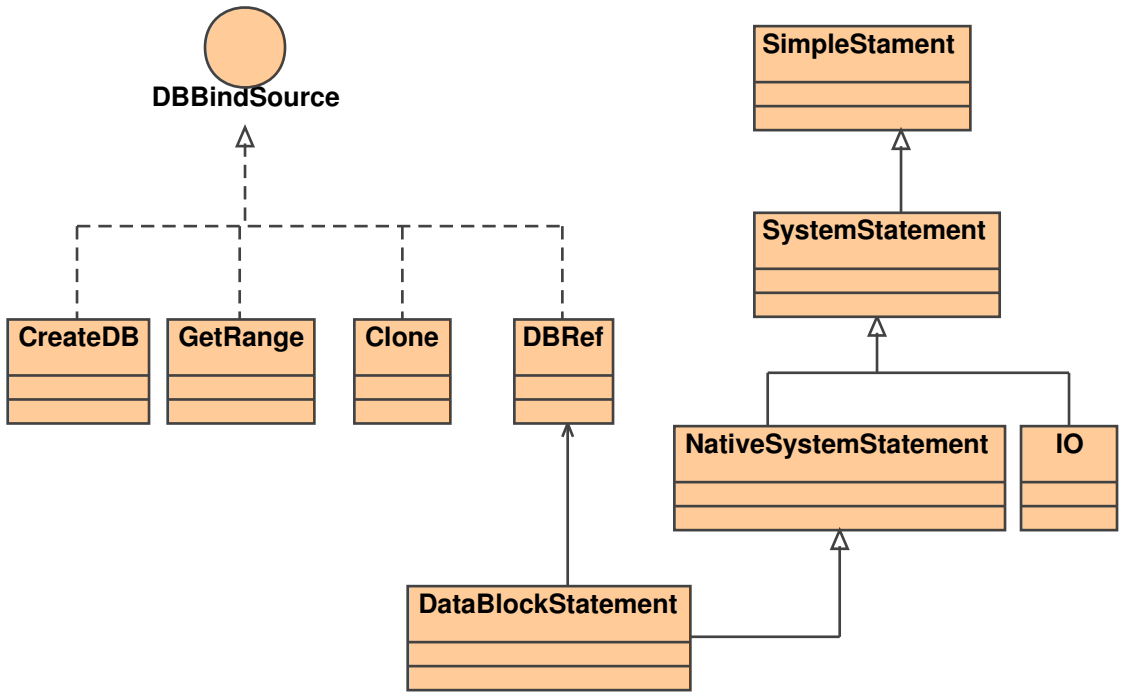

Figura A.3: Relacionamento da meta-classe que representa a referência a bloco de dado com outros componentes do metamodelo da VIRTUOSI.

Devido ao grande número de situações em que uma referência a bloco de dados existe, é preferível analisa-la separadamente em cada caso durante esta Seção.

A.1.2.2 Referência a Índice

Para a manipulação de um bloco de dados existe uma referência especial chamada **referência a índice**, ou simplesmente **índice**. Um índice pode ser obtido a partir de uma referência a bloco de dados. Quando isso acontece o índice passa a estar associado ao bloco de dados, ou seja, o índice passa a apontar para uma posição da seqüência contígua de

dados binários e, a partir desse momento, seu valor pode ser entendido como um número inteiro limitado entre zero e o tamanho do bloco de dados menos um. Um índice possui comandos tais como ir para frente, ir para trás, ir para determinada posição, etc. Também possui métodos de adição, subtração, multiplicação, etc. Estes métodos permitem o índice navegar na seqüência de dados binários.

O metamodelo da VIRTUOSI formaliza a relação entre um índice e uma referência a bloco de dados, conforme mostra a Figura A.4.

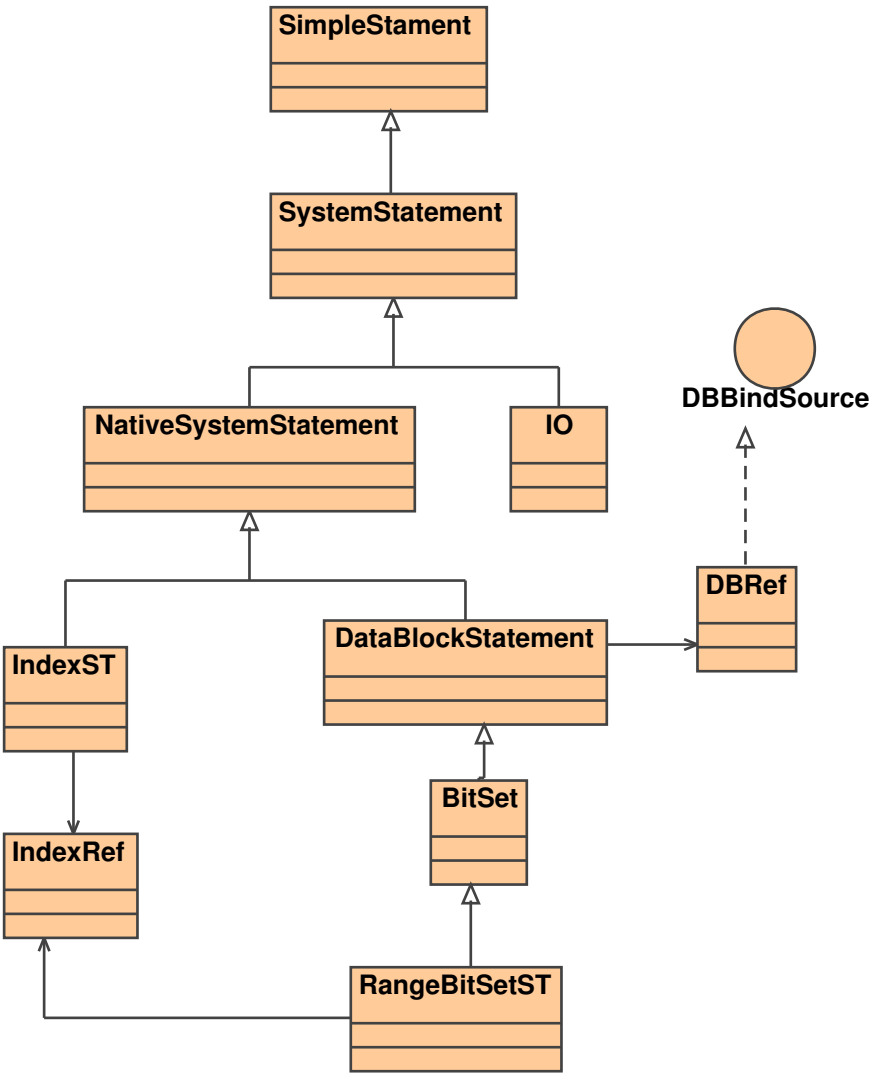

Figura A.4: Relação entre um índice e uma referência a bloco de dados

A meta-classe que representa uma referência a índice se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como parâmetro formal;
2. como parâmetro real;
3. como índice de uma referência a bloco de dados;

4. na comparação de referência a índice;
5. na comparação de referência a índice nula;
6. variável local;
7. como alvo da atribuição em um comando de atribuição de referência a índice;
8. com uma das possibilidades para a origem da atribuição em um comando de atribuição de referência a índice;
9. como parâmetro em alguns comandos de sistema para manipulação de blocos de dados de classes pré-definidas;
10. como parâmetro em alguns testáveis especiais para manipulação de blocos de dados de classes pré-definidas.

A Figura A.5 mostra o relacionamento da meta-classe que representa a referência a índice com outros componentes do metamodelo da VIRTUOSI, excetuando-se os comandos e testáveis especiais.

Devido ao grande número de situações em que uma referência a índice existe, é preferível analisa-la separadamente em cada caso durante esta Seção.

A.1.3 Classes

A meta-classe que representa uma **classe** se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como possuidora de atributos referência a objeto em um relacionamento associação;
2. como possuidora de atributos referência a objeto em um relacionamento composição exclusiva;
3. como possuidora de atributos referência a bloco de dados em um relacionamento composição exclusiva;
4. como possuidora de atributos de variáveis enumeradas em um relacionamento composição exclusiva;
5. como tipo de uma referência a objeto;
6. como possuidora de invocáveis;
7. como cliente da lista de exportação de um invocável;

A Figura A.6 mostra o relacionamento da meta-classe que representa uma classe de aplicação com outros componentes do metamodelo da VIRTUOSI, excetuando-se os relacionamentos com meta-classes descendentes.

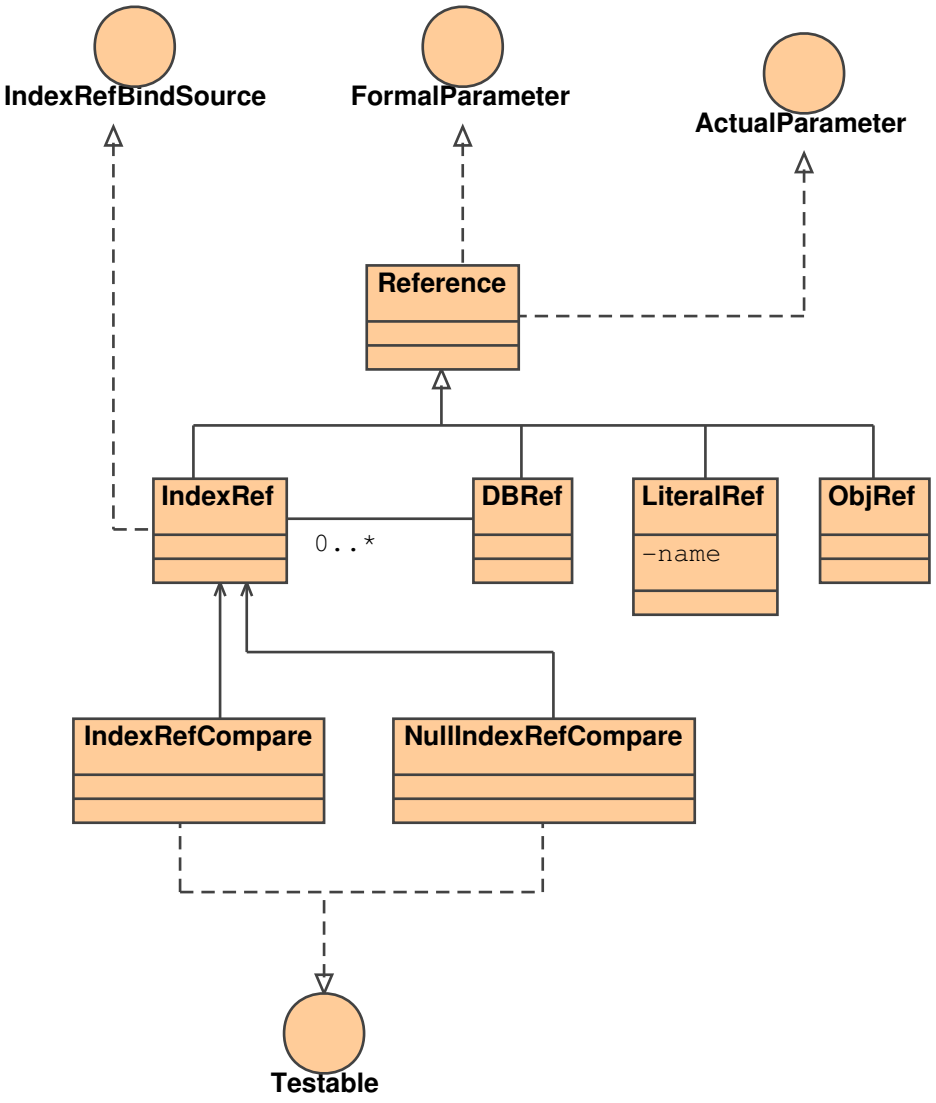

Figura A.5: Relacionamento da meta-classe que representa a referência a índice com outros componentes do metamodelo da VIRTUOSI

A.1.3.1 Herança

Duas classes podem estabelecer um relacionamento de herança entre si, tal que os invocáveis da classe herdeira podem acessar tanto o estado quanto o comportamento (métodos e ações) definidos pela classe ancestral, sem qualquer restrição. Em outras palavras, todas as definições de estado e comportamento existentes na classe ancestral são válidas para a classe herdeira. Segundo o metamodelo da VIRTUOSI, uma classe pode ter apenas uma classe ancestral direta[1], mas pode ter muitas classes herdeiras recursivamente. Entretanto, uma classe não pode ser direta ou indiretamente ancestral de si própria. Assim, um conjunto de classes pode ser organizado como um grafo acíclico dirigido no

[1]Essa propriedade é normalmente denominada herança simples, em contra-partida à herança múltipla, quando uma classe pode ter muitas ancestrais diretas.

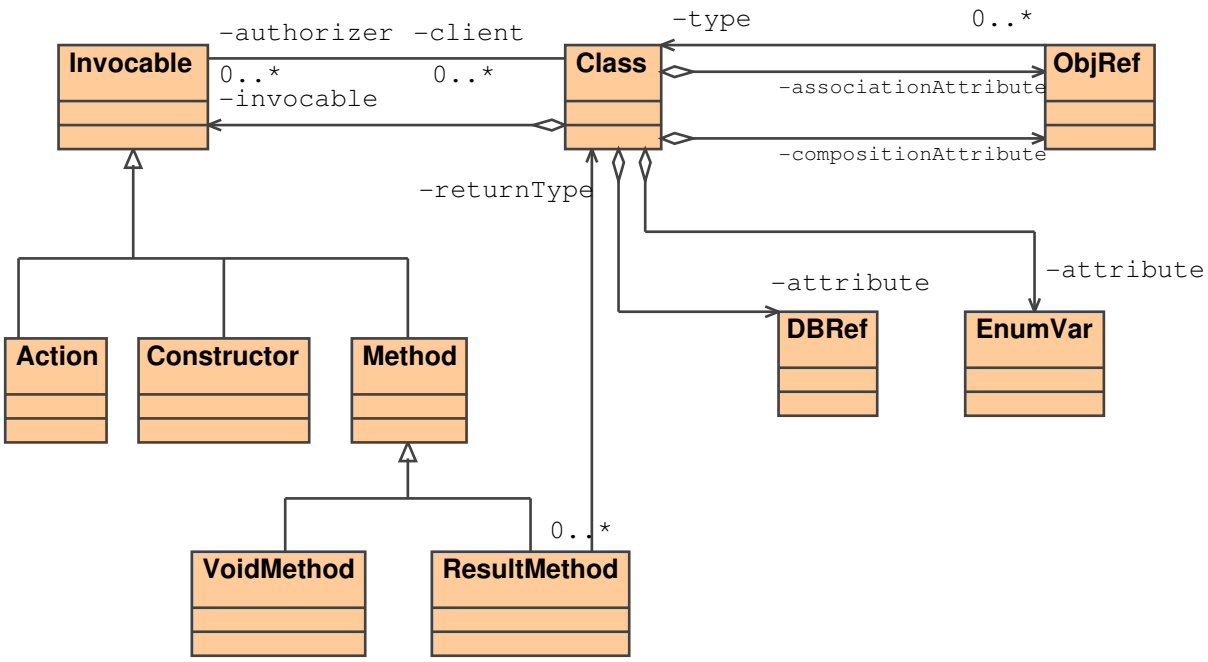

Figura A.6: Relacionamento da meta-classe que representa uma classe de aplicação com outros componentes do metamodelo da VIRTUOSI

qual a propriedade de herança é transitiva, isto é, uma classe herdeira assimila as definições de estado e de métodos de ancestrais diretas ou indiretas.

Uma conseqüência da transitividade da propriedade de herança é que uma referência de uma certa classe pode ter como alvo instâncias de distintas classes, desde que estas sejam herdeiras (diretas ou indiretas) da classe que define o tipo da referência, caracterizando assim a propriedade de polimorfismo.

O metamodelo da VIRTUOSI formaliza a relação de herança entre as classes, conforme mostrado na Figura A.7.

Existem dois tipos de classe, a **classe de aplicação** e a **classe raiz**. Toda classe da aplicação possui uma classe ancestral, sendo que esta pode ser uma outra classe de aplicação ou a classe raiz.

A.1.3.2 Classe Raiz

A classe raiz representa a classe ancestral – direta ou indireta – de todas as classes de aplicação, por este motivo não possui ancestral. Ela é única em toda a hierarquia de classes.

A.1.4 Atributos

O ambiente VIRTUOSI disponibiliza uma biblioteca de classes pré-definidas (*Integer*, *Real*, *Boolean*, *Character* e *String*) utilizadas para compor novas classes, as classes de aplicação.

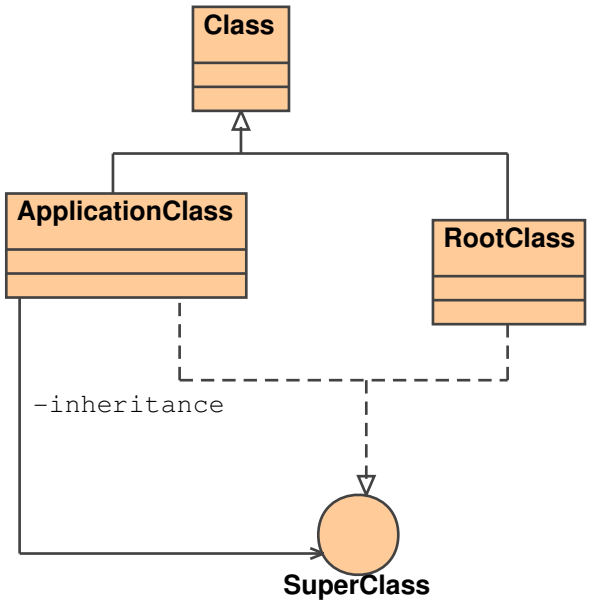

Figura A.7: Relação de herança entre classes segundo o metamodelo da VIRTUOSI

Os atributos de uma classe segundo o metamodelo da VIRTUOSI podem ser de três tipos, a saber:

- referência a objeto;
- referência a bloco de dados;
- variável enumerada.

O metadomelo da VIRTUOSI formaliza os três tipos de atributo que uma classe conforme mostra a Figura A.8.

A.1.4.1 Referência a Objeto

A meta-classe que representa uma **referência a objeto** se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como parâmetro formal;
2. como parâmetro real;
3. como atributo de uma classe por associação;
4. como atributo de uma classe por composição;
5. como sendo um tipo de classe;
6. como objeto em um acesso a atributo variável enumerada;

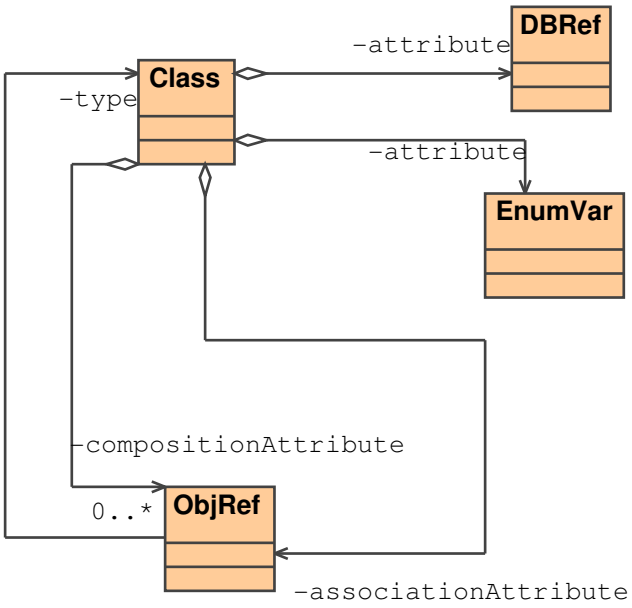

Figura A.8: Os três tipos de atributos possíveis em uma classe segundo o metamodelo da VIRTUOSI

7. como objeto em um acesso a atributo objeto (Este componente é explicado na Seção A.1.7, especificamente na discussão sobre o comando para atribuição de referência a objeto);

8. como atributo em um acesso a atributo objeto (Este componente é explicado na Seção A.1.7, especificamente na discussão sobre o comando para atribuição de referência a objeto);

9. como objeto em um acesso a atributo bloco de dados (Este componente é explicado na Seção A.1.7, especificamente na discussão sobre o comando para atribuição de referência a bloco de dados);

10. na comparação entre duas referência a objeto;

11. na comparação entre uma referência a objeto e uma referência nula;

12. como alvo da atribuição em um comando de atribuição de referência a objeto;

13. como alvo da atribuição em um comando de atribuição de referência nula a objeto;

14. com uma das possibilidades para a origem da atribuição em um comando de atribuição de referência a objeto;

15. com uma das possibilidades para o testável associado a um comando de desvio condicional (Tanto o componente testável quanto o comando de desvio condicional são explicados na Seção A.1.7, especificamente na discussão sobre o comando desvio condicional);

16. como referência retornada por um invocável;
17. como variável local;
18. como alvo de invocação de método;
19. como alvo de invocação de uma ação.

Devido ao grande número de situações em que uma referência a objeto existe, é preferível analisa-la separadamente em cada caso durante esta Seção.

A Figura A.9 mostra o relacionamento da meta-classe que representa a referência a objeto com outros componentes do metamodelo da VIRTUOSI.

A Figura A.10 mostra uma classe implementada em Aram – segundo a definição do metamodelo da VIRTUOSI – com três atributos do tipo referência a objeto. O primeiro e o segundo atributo são instâncias de uma classe pré-definida (*String*). O terceiro atributo é instância de uma classe de aplicação, no caso `Pessoa`.

Um atributo do tipo referência a objeto pode estar associado a classe por **composição** ou **associação**.

Composição: Observando a Figura A.10 nota-se que o primeiro atributo – uma *String* de nome *name* – possui como parte de sua declaração a palavra *composition*. A existência da palavra *composition* indica um relacionamento de **composição exclusiva** entre uma classe e um atributo. A Figura A.11 mostra um código fonte com uma relação de composição exclusiva entre classe e atributo e ilustra a semântica correspondente utilizando os objetos envolvidos. Em uma relação de composição exclusiva o objeto representado pelo atributo está contido no objeto representado pela classe. Como conseqüência do relacionamento de composição exclusiva um objeto contido somente pode ser referenciado por ele próprio, pelo seu contentor direto ou por outro objeto que seja contido no mesmo objeto contentor.

Associação: Observando-se novamente a Figura A.10 nota-se que o segundo atributo – uma *String* de nome *address* – possui como parte de sua declaração a palavra *association*. A palavra *association* indica um relacionamento de **associação** entre uma classe e um atributo. A Figura A.12 mostra um código fonte com uma relação de associação entre classe e atributo e ilustra a semântica correspondente utilizando os objetos envolvidos. Em uma relação de associação o objeto representado pelo atributo não faz parte do objeto representado pela classe, simplesmente está associado a ele. Como conseqüência do relacionamento de associação um objeto associado pode ser referenciado por qualquer outro objeto.

A Figura A.13 mostra como o metamodelo da VIRTUOSI formaliza as relações de composição e associação entre uma classe e seus atributos.

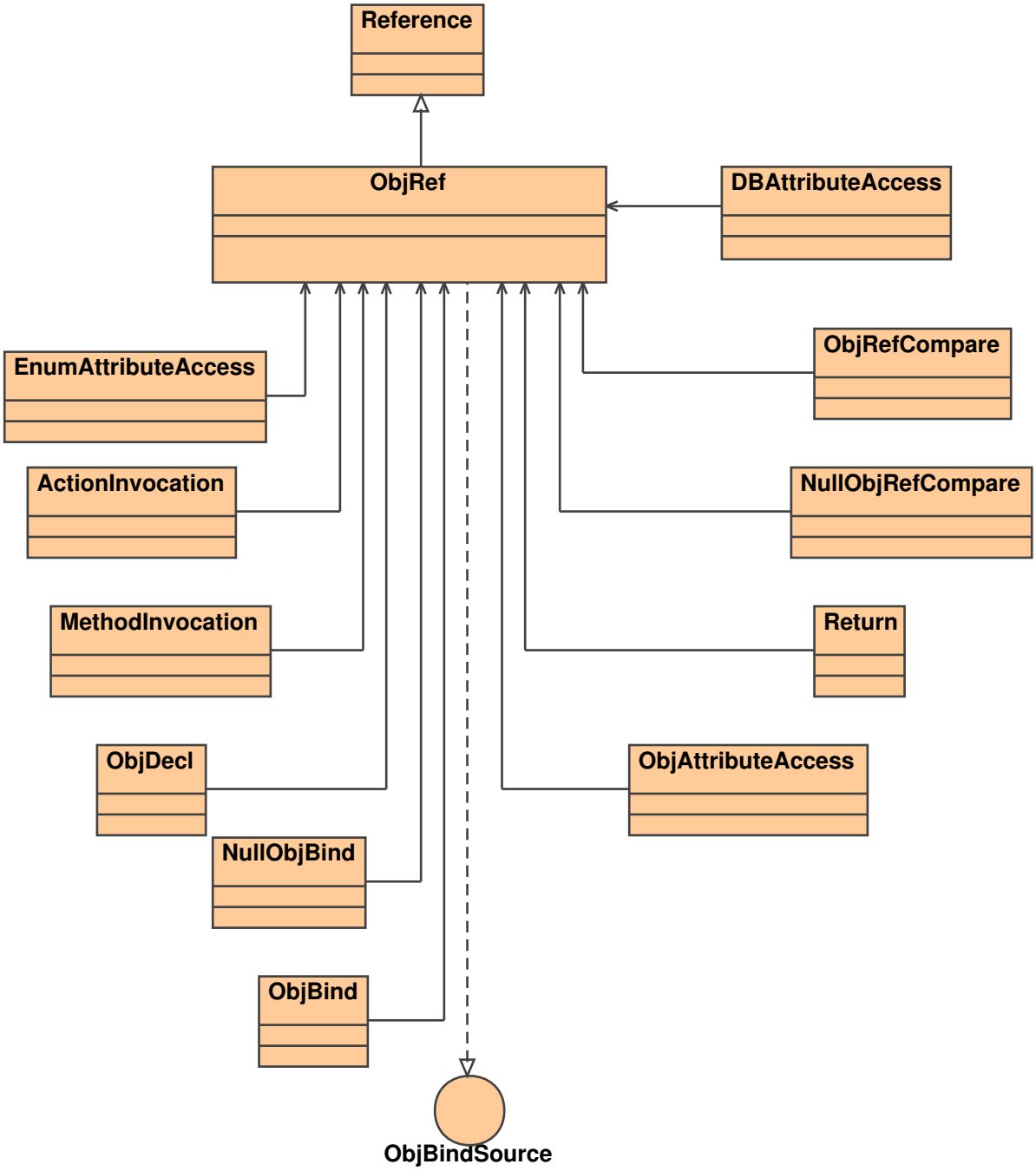

Figura A.9: Relacionamento da meta-classe que representa a referência a objeto com outros componentes do metamodelo da VIRTUOSI

A.1.4.2 Referência a Bloco de Dados

Segundo o metamodelo da VIRTUOSI, um programador tem a possibilidade de criar novas classes que não dependam de nenhuma outra classe pré-existente. Para tanto, um atributo pode referenciar um bloco de dados. Esse tipo de referência é chamada referência a bloco de dados. Uma referência a bloco de dados consiste em uma referência para uma

```
class Pessoa {
   composition String nome;     //classe pré-definida
   association String endereco;  //classe pré-definida
   association Person esposa;     //classe de aplicação
}
```

Figura A.10: Atributos em uma classe segundo o metamodelo da VIRTUOSI

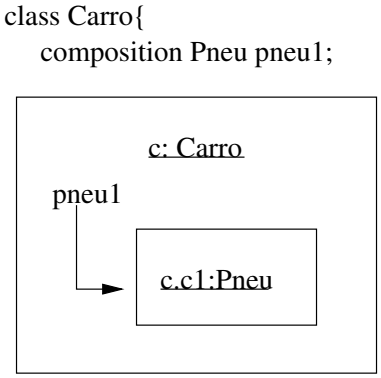

Figura A.11: Código fonte em Aram e diagrama de objeto de uma relação de composição entre uma classe e um atributo

seqüência contígua de dados binários em memória. Um atributo do tipo referência a bloco de dados obrigatoriamente tem uma relação de composição exclusiva com a classe que o possui. Esse tipo de referência é geralmente utilizado para a construção de classes pré-definidas. Cada classe pré-definida é responsável por dar o significado de sua seqüência de dados binários através de suas operações. Por exemplo, um objeto do tipo básico *Integer* pode armazenar um valor inteiro utilizando um bloco de dados de qualquer tamanho, nesse caso uma operação para adicionar um outro valor inteiro (armazenado em outro objeto do tipo *Integer* também utilizando um bloco de dados) ao valor inteiro deste objeto, deve conhecer a convenção utilizada na representação binária de ambos as seqüências. A Figura A.14 mostra o código fonte de uma classe que possui um atributo do tipo bloco de dado e uma ilustração de uma instância desta classe contendo o bloco de dados.

A.1.4.3 Variável Enumerada

Uma classe implementada segundo o metamodelo da VIRTUOSI, pode ainda, ter um atributo do tipo variável enumerada. Um atributo do tipo variável enumerada possui um conjunto de valores possíveis definidos na construção da classe. Os valores possíveis de uma variável enumerada não são objetos de nenhuma outra classe, são simples valores literais. Esse conjunto de valores possíveis de uma variável enumerada chama-se **enumerado**. Um atributo do tipo variável enumerada recebe um valor inicial durante sua declaração. A Figura A.15 mostra o código fonte de uma classe que possui um atributo

```
class Pessoa{
    composition String nome;
    enum {masculino, feminino} sexo = masculino;

    constructor make(String pNome, literal pSexo) exports{all}
    {
        nome = pNome;
        sexo = pSexo;
    }
    ...
} class Aviao{
    association Pessoa passageiro;

    constructor make() exports{all}
    {
    }

    method reservar(Pessoa pPessoa) exports{all}
    {
        passageiro = pPessoa;
    }
    ...
}
```

Figura A.12: Código fonte em Aram de uma relação de associação entre uma classe e um atributo

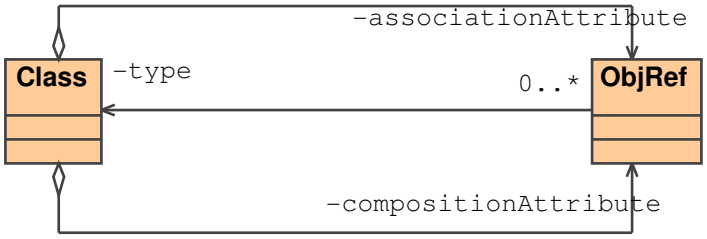

Figura A.13: As duas maneiras em que uma referência a objeto representa o papel de atributo segundo o metamodelo da VIRTUOSI

variável enumerada.

A.1.5 Referências

Existem quatro tipos de referência suportadas pelo metamodelo da VIRTUOSI, a saber:

- referência a objeto;

```
class Inteiro{
    datablock valor;

    constructor make(literal pValor) exports{all}
    {
        valor = datablock.make(32);
        valor.storeInteger(pValor);
    }
    constructor make(Inteiro pValor) exports{all}
    {
        this.set(pValor);
    }
    method void set(Inteiro i) exports{all}
    {
        datablock k = i.valor;
        valor = k.clone();
    }
    ...
}
```

Figura A.14: Um exemplo de atributo do tipo bloco de dados e uma representação de um objeto correspondente – código fonte em Aram

```
class Carro {
    enum { true, false } conversivel = false;
    constructor make() exports{all}
    {
    }
    ...
}
```

Figura A.15: Um exemplo de atributo do tipo enumerado – código fonte em Aram

- referência bloco de dados;
- referência a índice;
- referência a literal.

A Figura A.16 mostra o relacionamento de herança entre as referências na VIRTUOSI e mostra também que qualquer referência pode ser passada como parâmetro real e fazer parte dos parâmetros formais de um invocável. Deve-se notar que a meta-classe referência é abstrata, e é concretizada pelos quatro tipos de referência.

Observando-se a Figura A.16 nota-se que existe uma meta-classe chamada `This` herdeira da meta-classe que representa uma referência a objeto. Essa meta-classe representa uma referência para o objeto corrente durante a interpretação de um método. Um

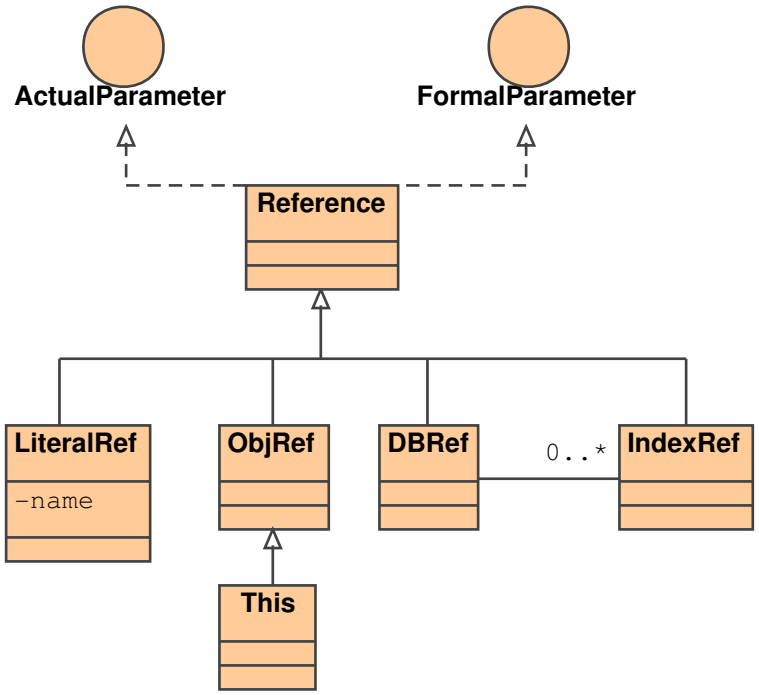

Figura A.16: Relacionamento entre as meta-classe que definem os tipos de referência na VIRTUOSI

método sempre é invocado através de uma referência a objeto sobre o objeto o qual ela aponta. Uma referência do tipo `This` é utilizada quando dentro de um método deseja-se invocar um método da própria classe e sobre o próprio objeto corrente.

A.1.6 Invocáveis

Segundo o metamodelo da VIRTUOSI, uma operação de uma classe pode ser implementada de duas maneiras, a saber:

- como um **método**;
- como uma **ação**;

Essas duas implementações descrevem o conjunto dos serviços que uma classe disponibiliza. Além das operações, uma classe precisa implementar um método especial utilizado para criar novas instâncias da classe, esse tipo de método especial é chamado de **construtor**.

O metamodelo da VIRTUOSI formaliza a relação entre uma classe e as possíveis implementações de suas operações e construtores, conforme mostra a Figura A.17.

Um método, um construtor ou uma ação podem sofrer invocação e, por isso, o metamodelo da VIRTUOSI os formaliza como **invocáveis**[2].

[2]Do inglês ***invocable*** (o termo **invocável** ainda não está registrado nos dicionários da língua portuguesa).

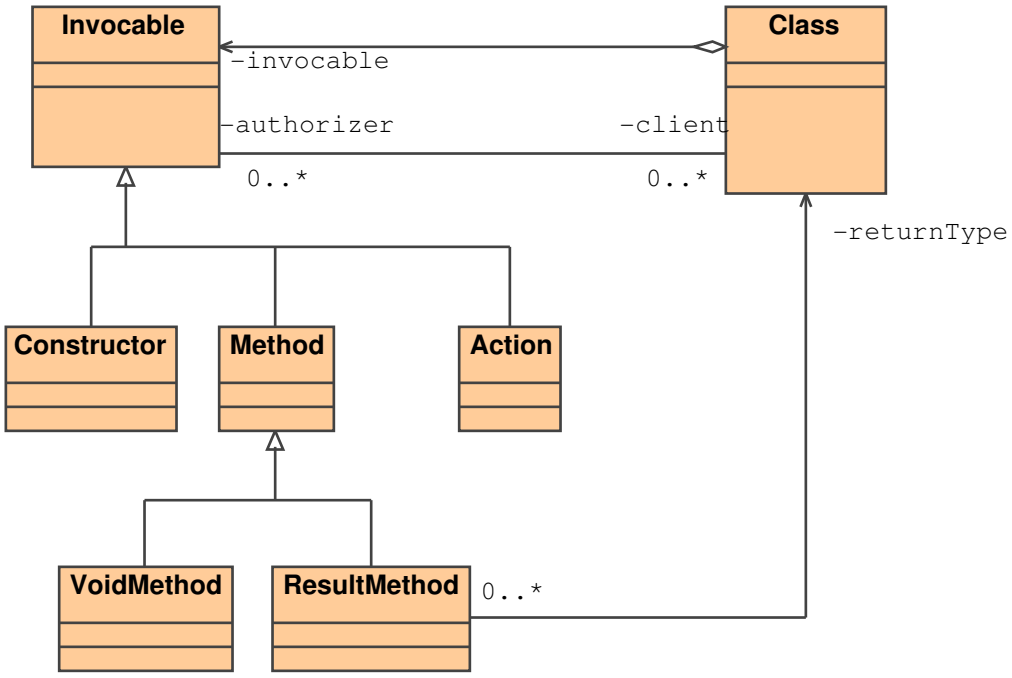

Figura A.17: Relação entre uma classe e as possíveis implementações de suas operações

A Figura A.18 mostra o relacionamento da meta-classe Invocável com outros componentes do metamodelo da VIRTUOSI.

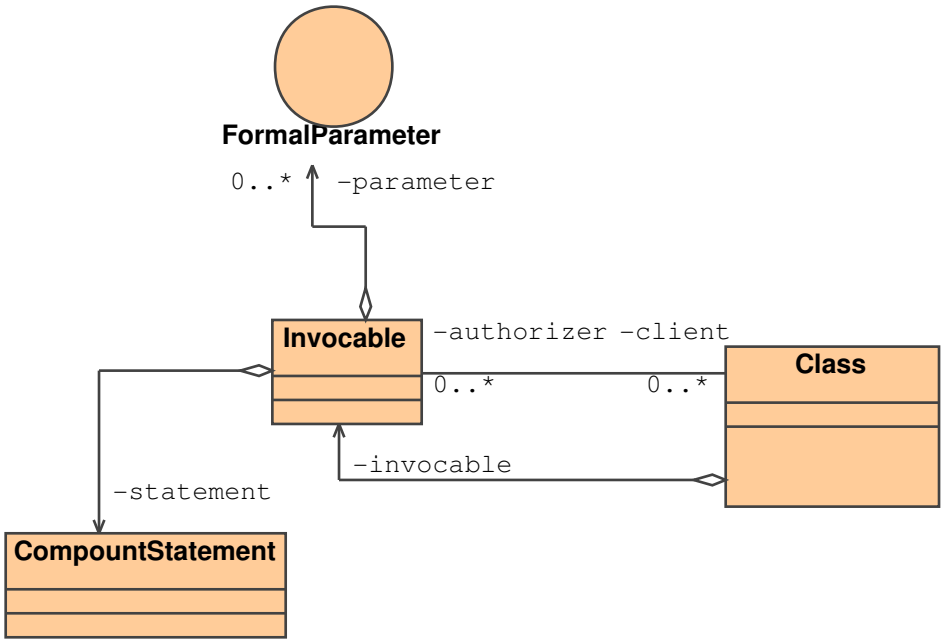

Figura A.18: Relacionamento da meta-classe Invocável com outros componentes do metamodelo da VIRTUOSI

A meta-classe que representa um invocável – independente do seu tipo (construtor, método ou ação) – se relaciona através de associações com outros componentes do metamodelo da VIRTUOSI nas seguintes situações:

1. como possuidora de parâmetros formais;
2. como possuidora de comandos;
3. como pertencente a uma classe;
4. como fornecedora para meta-classes que tem autorização para invocá-la;

Parâmetros:

Um invocável pode receber parâmetros. Por isso um invocável define uma seqüência de parâmetros que deve ser provida durante sua invocação (ou chamada).

Um parâmetro pode ser visto sob duas perspectivas diferentes. A primeira ocorre durante a construção de um invocável onde o parâmetro tem o papel de **parâmetro formal**[3], ou seja, ele define o nome, o tipo e a posição que determinado parâmetro possuirá na seqüência dos parâmetros formais daquela invocação. A segunda ocorre no momento da chamada do invocável, onde um parâmetro é uma referência a um objeto já existente, tendo assim, o papel de **parâmetro real**[4].

O conjunto de parâmetros formais de um invocável pode ser vazio ou composto de referências. Qualquer tipo de referência pode ser um parâmetro formal, inclusive uma referência a literal[5] utilizada para receber os parâmetros reais do tipo valor literal.

A Figura A.19 mostra um exemplo de uma classe de aplicação `Taxi` com dois métodos, um com uma lista de parâmetros vazia (`sairPassageiro`) e outro com uma lista contendo um parâmetro formal do tipo referência a objeto (`entrarPassageiro`). A Figura mostra também a classe de aplicação `Principal`, que invoca os dois métodos da classe de aplicação `Taxi`.

O metamodelo da VIRTUOSI formaliza a relação entre um invocável e seus parâmetros, conforme mostra a Figura A.20.

Lista de Exportação: Um invocável define explicitamente quais as outras classes cujos invocáveis podem realizar invocações sobre o invocável em questão. Para tanto, um invocável possui uma **lista de exportação**, ou seja, uma lista das classes cujos invocáveis podem invocá-lo.

Alguns exemplos do uso da lista de exportação podem ser observados na Figura A.21. A Figura mostra três casos distintos: o método *exportedToAllMethod* – exportado

[3]Do inglês ***formal parameter***.

[4]Do inglês ***actual parameter***.

[5]Embora um parâmetro seja utilizado pelo invocável da mesma forma que uma variável local, no caso de parâmetros do tipo referência a literal, o parâmetro não pode sofrer atribuição, visto que, não existe um comando de atribuição para referência a literal. Portanto, uma referência a literal sempre tem seu valor literal atribuído por um comando de invocação de invocável

```
class Principal
{
   constructor iniciar() exports { all }
   {
       Taxi corsa = Taxi.instanciar();
       ...
       Boolean entrou = corsa.entrarPassageiro(andrea);
       ...
       corsa.sairPassageiro();
       ...
   }
   ...
}
class Taxi {
   ...

   // método sem parâmetros
   method void sairPassageiro( ) exports { Principal }
   {
       ...
   }
   // método com parâmetros
   method Boolean entrarPassageiro( Pessoa p ) exports { Principal }
   {
       ...
   }
}
```

Figura A.19: Código fonte em Aram contendo um método sem parâmetros e um método com parâmetros

para toda e qualquer classe –, o método *nonExportedMethod* – não exportado para nenhuma classe – e o método *exportedToBandC* exportado para as classes B e C. Deve-se observar que nos dois primeiros casos foram utilizadas palavras reservadas da linguagem ao invés de uma lista de nomes de outras classes. Existem duas palavras reservadas que podem ser utilizadas no lugar de uma lista de exportação: *all* e *none*. A palavra *all* implica que o invocável em questão pode ser invocado a partir de invocáveis de toda e qualquer classe, enquanto que a palavra *none* implica que o invocável em questão somente pode ser invocado pelos invocáveis pertencentes a mesma classe que o possui. O uso da palavra *none* permite que invocáveis sejam acessados sem restrição por qualquer invocável da própria classe.

O metamodelo da VIRTUOSI formaliza a relação entre um invocável e sua lista de exportação, conforme mostra a Figura A.22.

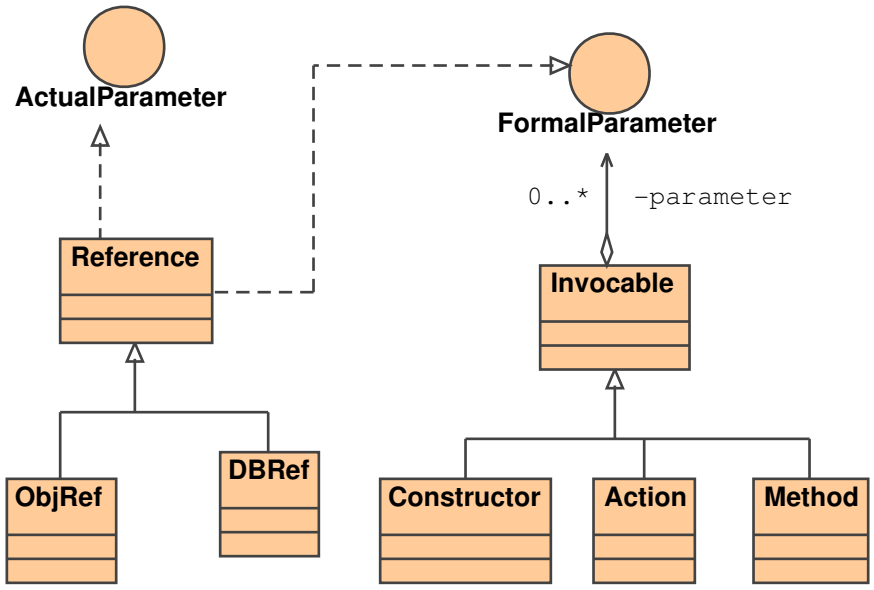

Figura A.20: Relação de um Invocável e seus parâmetros

```
class A {
   ...

   // método exportado para toda e qualquer classe
   method void exportedToAllMethod() exports all
   {
       ...
   }

   // método não exportado para nenhuma classe
   method void nonExportedMethod() exports none
   {
       ...
   }
   method void exportedToBandC() exports { B, C }
}
```

Figura A.21: Métodos com diferentes listas de exportação – Código fonte em Aram

Construtor: Um construtor não faz parte das operações definidas por um TAD pois não interfere no comportamento dos objetos representados pelo TAD. Porém, têm fundamental importância na implementação de uma classe, visto que, um objeto sempre é criado através de sua interpretação. O retorno da interpretação de um construtor é sempre um novo objeto, uma nova instância de uma classe.

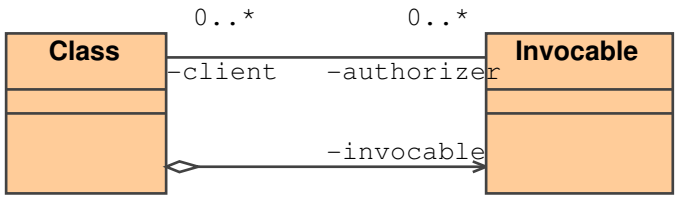

Figura A.22: Relação de um Invocável sua lista de exportação

Método: Um método é a maneira mais comum de implementar uma operação definida por um TAD. Existem dois tipos de métodos, a saber: **método sem retorno** – muitas vezes chamado de procedimento – e **método com retorno** – muitas vezes chamado função – conforme formalizado pelo metamodelo da VIRTUOSI e mostrado na Figura A.23.

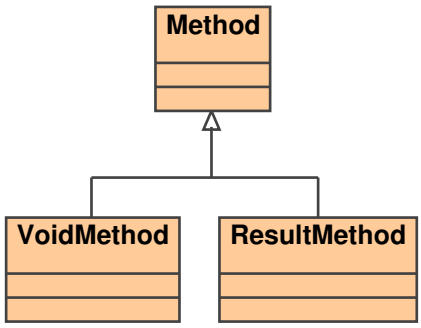

Figura A.23: Métodos com retorno e métodos sem retorno

A diferenciação entre métodos com e sem retorno se dá em parte pelo uso da palavra que fica entre a palavra *method* e o nome do método. Caso essa palavra seja *void* trata-se de um método sem retorno. Caso a palavra seja o nome de uma classe trata-se de um método com retorno. Outra diferença consiste no fato de um método com retorno sempre possuir ao menos um comando retorno. A Figura A.24 mostra um exemplo de código fonte em Aram contendo um método sem retorno e um método com retorno. O comando retorno é um comando simples e é discutido na Seção A.1.7.

Ação: A segunda maneira de implementar uma operação definida por um TAD é através de uma ação. Uma ação pode ser vista como uma operação cujo retorno é utilizado para a tomada de decisão referente à um comando de desvio. Em outras palavras, o retorno de uma ação permite o comando de desvio decidir qual dentre duas seqüências de comandos deve ser interpretada. Uma ação não retorna uma referência a objeto. Diferente de um método com retorno ou um construtor – onde uma referência é retornada – o retorno de uma ação é um comando simples chamado: **comando resultado de teste**. Tanto o comando de desvio quanto o comando resultado de teste são detalhados na Seção A.1.7.7.

```
class Taxi {
   ...

   // método sem retorno
   method void sairPassageiro( ) exports { Principal }
   {
       ...
   }
   // método com retorno
   method Boolean entrarPassageiro( Pessoa p ) exports { Principal }
   {
       ...
       return resultado;
   }
}
```

Figura A.24: Diferenciação entre métodos com retorno e métodos sem retorno

A Figura A.25 mostra um exemplo de código fonte com uma declaração de uma ação, segundo o metamodelo da VIRTUOSI.

```
class Pessoa {
   ...

   // ação
   action casado() exports all
   {
       ...
   }
}
```

Figura A.25: Código fonte em Aram contendo uma declaração de uma ação

A.1.7 Comandos

No ambiente VIRTUOSI, toda computação é realizada através da interpretação dos comandos que compõem um invocável. Esses comandos podem ser invocações de outros invocáveis, comandos responsáveis por controlar o fluxo da interpretação, comandos para manipular referências a objetos ou ainda comandos de sistema para a manipulação de referência a bloco de dados e manipulação de referência a índice. Esses comandos são interpretados a partir do momento que um invocável é invocado.

A.1.7.1 Composição de Comandos

Um invocável define uma seqüência de comandos. Comandos podem ser simples ou compostos. Um **comando simples**[6] pode ser uma atribuição, uma invocação de operação, um desvio condicional, conforme detalhado no restante dessa Seção. Um **comando composto**[7] é uma seqüência de comandos simples ou compostos, recursivamente.

Uma seqüência de comandos, simples ou compostos, pode ser agrupada em um comando composto. Esse relacionamento é mostrado na Figura A.26.

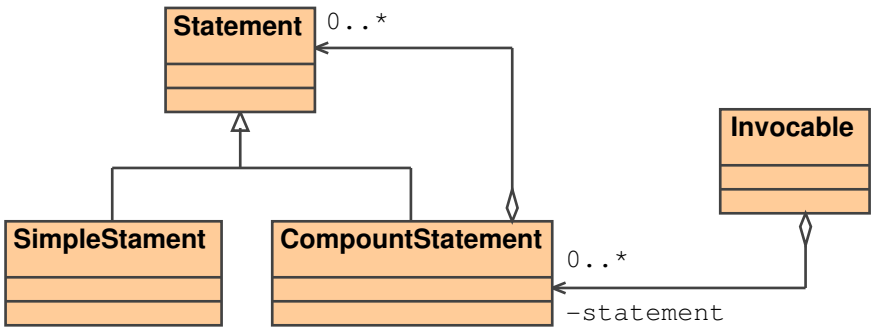

Figura A.26: Relacionamento de herança entre as meta-classes comando, comando simples e comando composto

A.1.7.2 Declaração de Variáveis

Para se invocar um invocável é preciso possuir uma referência para um objeto da classe que o define(com exceção de construtores que são invocados a partir do nome da classe). Segundo o metamodelo da VIRTUOSI, uma referência existe sob três formas, a saber:

1. atributo;
2. parâmetro;
3. **variável local**.

Ao contrário dos atributos e parâmetros que são definidos durante a construção da classe e seus respectivos invocáveis, uma variável local não existe até que seja declarada. Portanto, existem instruções para a declaração de alguns tipos de referência utilizadas como variáveis locais. A Figura A.27 mostra a hierarquia de classes que determina os comandos simples disponíveis para a declaração de variáveis locais, segundo o metamodelo da VIRTUOSI.

Conforme mostra a Figura A.27, é possível declarar variáveis locais do tipo:

[6]Do inglês ***simple statement***.
[7]Do inglês ***compound statement***.

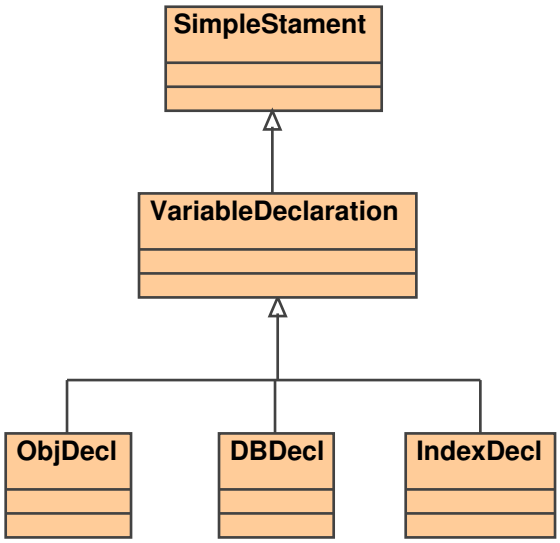

Figura A.27: Tipos de declarações para variáveis locais

- referência a objeto;
- referência a bloco de dados;
- referência a índice.

Observando novamente a Figura A.27 nota-se que não existe declaração de variável local do tipo referência a literal. Isto ocorre porque uma referência a literal somente é permitida como parâmetro de um invocável, ou seja, no momento da invocação o que é passado como parâmetro é um valor literal, e não uma referência a literal. A Figura A.28 mostra uma invocação de um método passando um valor literal e a declaração desse mesmo método contendo um parâmetro formal do tipo referência a literal.

Declaração de Variáveis do Tipo Referência a Objeto: A Figura A.29 mostra a declaração de uma variável local do tipo referência a objeto. Após a interpretação desse comando a referência não possui um alvo, ou seja, não está apontando para algum objeto em memória. Uma referência sem alvo é chamada de referência nula.

Declaração de Variáveis do Tipo Referência a Bloco de Dados: A Figura A.30 mostra a declaração de uma variável local do tipo referência a bloco de dados. Após a interpretação desse comando a referência não possui um alvo, ou seja, não está apontando para alguma seqüência de dados binários em memória. Da mesma forma que uma referência a objeto, uma referência a bloco de dados sem alvo também é uma referência nula.

Declaração de Variáveis do Tipo Referência a Índice: Além da declaração de variável local do tipo referência a objeto e do tipo referência a bloco de dados existe a

```
class Person {
   ...
   constructor create()
   {
       Integer age;
       age = Integer.make(26); // '26' é um valor literal
   }
}
...
class Integer {
   ...
   constructor make ( Literal charSequence){
       ...
   }
}
```

Figura A.28: Invocação de método passando como parâmetro um valor literal e a declaração do método invocado – código fonte em Aram

```
class Person {
   constructor make() exports all
   {
       // declaração de variável local inicialmente nula.
       String name;
       ...
   }
   ...
}
```

Figura A.29: Declaração de uma variável local do tipo referência a objeto – código fonte em Aram

declaração de variável local do tipo referência a índice. A Figura A.31 mostra a declaração de uma variável local do tipo referência a índice. Após a interpretação desse comando a referência não possui um alvo, ou seja, não está apontando alguma seqüência de dados binários em memória, ou seja, é uma referência nula.

A.1.7.3 Atribuição

Uma referência nula não permite uma invocação a partir dela. Isso é verdade tanto para variáveis locais quanto para atributos e parâmetros. Portanto, é preciso fazer com que uma referência aponte para um alvo, ou seja, uma referência a objeto precisa apontar para um objeto em memória, uma referência a bloco de dados precisa apontar para uma seqüência de dados binários em memória e uma referência a índice precisa apontar

```
class Integer {
   method void add( Integer i ) exports all
   {
       // declaração de variável local inicialmente nula.
       datablock d;
       ...
   }
   ...
}
```

Figura A.30: Declaração de uma variável local do tipo referência a bloco de dados – código fonte em Aram

```
class Integer {
   method void add( Integer i ) exports all
   {
       // declaração de variável local inicialmente nula.
       index i;
       ...
   }
   ...
}
```

Figura A.31: Declaração de uma variável local do tipo referência a índice – código fonte em Aram

para uma posição na seqüência de dados binários em memória. Isso ocorre através da interpretação de um comando de atribuição. Esse comando é identificado no código fonte pelo uso do caracter '=' chamado de caracter de atribuição. O que estiver a esquerda do caracter de atribuição é chamado **alvo da atribuição**, e o que estiver a direita do caracter de atribuição é chamado **origem da atribuição**.

Segundo o metamodelo da VIRTUOSI, os comandos de atribuição existentes são os seguintes:

- atribuição de referência a objeto;
- atribuição de referência nula a objeto;
- atribuição de referência a bloco de dados;
- atribuição de referência nula a bloco de dados;
- atribuição de referência a índice;
- atribuição de variável enumerada.

Atribuição de Referência a Objeto: No caso da atribuição de referência a objeto o alvo da atribuição sempre é uma referência a objeto, enquanto que a origem da atribuição pode ser:

- uma referência a objeto;
- um **acesso a atributo objeto**, ou seja, um acesso – somente de leitura – a um atributo do tipo referência a objeto, de outro objeto instância da mesma classe que implementa o invocável onde a atribuição ocorre;
- um comando de invocação de construtor (Este componente é explicado nesta Seção, especificamente na discussão sobre o comando de invocação de construtor);
- um comando de invocação de método com retorno (Este componente é explicado nesta Seção, especificamente na discussão sobre o comando de invocação de método com retorno).

Atribuição de Referência Nula a Objeto: Um comando de atribuição de objeto nulo, faz uma referência a objeto voltar a ser uma referência nula.

A Figura A.32 mostra o relacionamento das meta-classes que representam os dois comandos de atribuição de referência a objeto com outros componentes do metamodelo da VIRTUOSI.

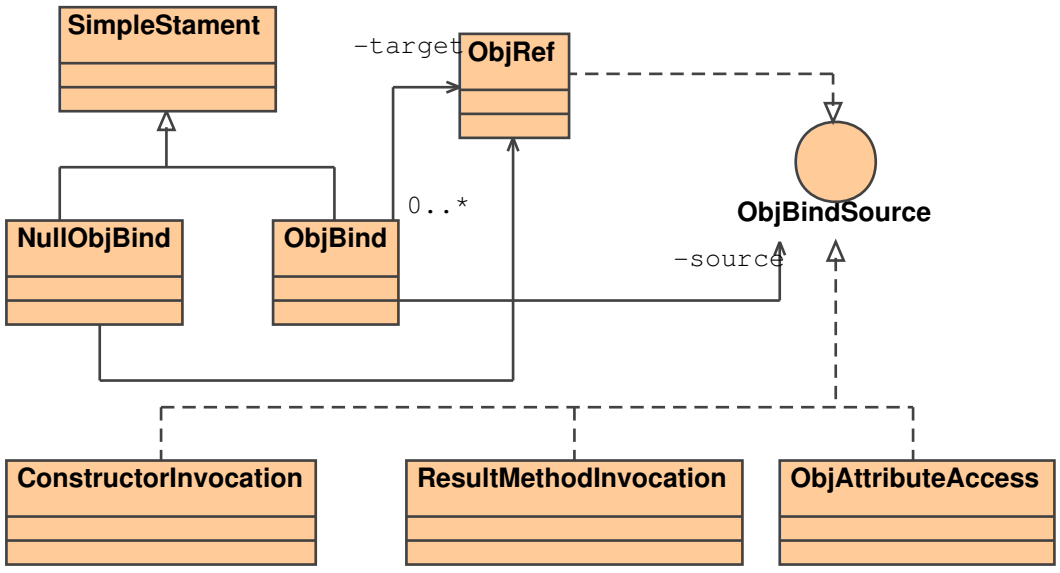

Figura A.32: Relacionamento das meta-classes que representam os comandos de atribuição de referência a objeto com outros componentes do metamodelo da VIRTUOSI

Atribuição de Referência a Bloco de Dados: No caso da atribuição de referência a bloco de dados o alvo da atribuição sempre é uma referência a bloco de dados, enquanto que a origem da atribuição pode ser:

- uma referência a bloco de dados;
- um acesso a atributo bloco de dados, ou seja, um acesso – somente de leitura – a um atributo do tipo referência a bloco de dados, de outro objeto instância da mesma classe que implementa o invocável onde a atribuição ocorre.

Atribuição de Referência Nula a Bloco de Dados: Um comando de atribuição de bloco de dados nulo, faz uma referência a bloco de dados voltar a ser uma referência nula.

A Figura A.33 mostra o relacionamento das meta-classes que representam os dois comandos de atribuição de referência a bloco de dados com outros componentes do metamodelo da VIRTUOSI.

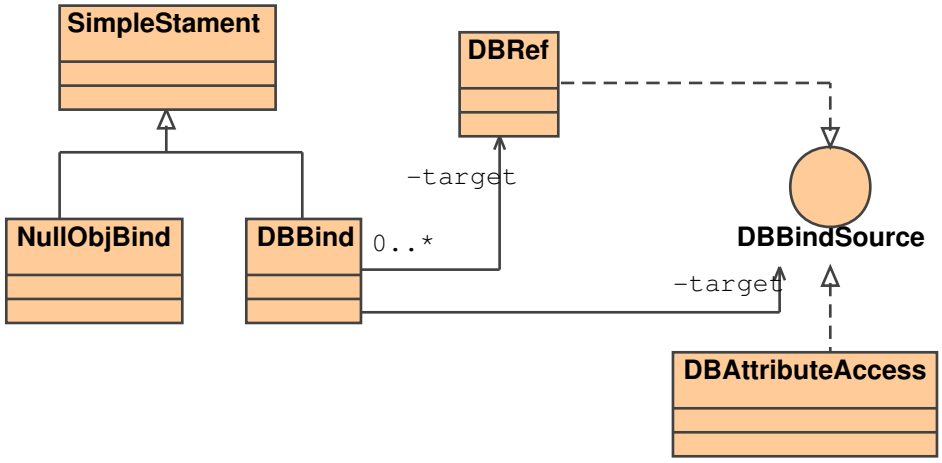

Figura A.33: Relacionamento das meta-classes que representam os dois comandos de atribuição de referência a bloco de dados com outros componentes do metamodelo da VIRTUOSI

Atribuição de Referência a Índice: No caso da atribuição de referência a índice o alvo da atribuição sempre é uma referência a índice, enquanto que a origem da atribuição pode ser:

- uma referência a índice;
- um comando especial para a manipulação de referência a bloco de dados.
- um comando de atribuição de índice nulo, faz uma referência a índice voltar a ser uma referência nula.

A Figura A.34 mostra o relacionamento da meta-classe comando de atribuição de referência a índice com outros componentes do metamodelo da VIRTUOSI.

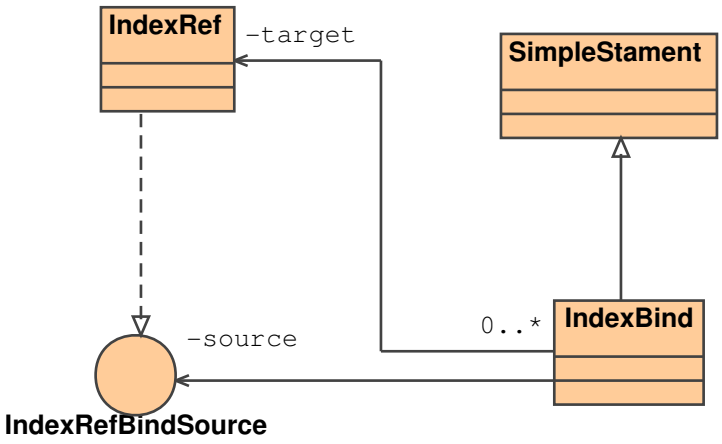

Figura A.34: Relacionamento da meta-classe comando de atribuição de referência a índice com outros componentes do metamodelo da VIRTUOSI

Atribuição de Variável Enumerada: No caso da atribuição de variável enumerada o alvo da atribuição sempre é uma variável enumerada, enquanto que a origem da atribuição pode ser:

- um valor literal;
- uma referência a literal;
- um acesso a atributo variável enumerada, ou seja, ou seja, um acesso – somente de leitura – a um atributo do tipo variável enumerada, de outro objeto instância da mesma classe que implementa o invocável onde a atribuição ocorre;

Um atributo do tipo variável enumerada recebe um valor inicial durante sua declaração e somente é permitido atribuir-lhe um valor literal pertencente ao enumerado definido.

A Figura A.35 mostra o relacionamento da meta-classe comando de atribuição variável enumerada com outros componentes do metamodelo da VIRTUOSI.

A.1.7.4 Retorno de Método

O comando de retorno utilizado por um método é chamado simplesmente de **retorno**, sendo que sempre retorna uma referência a objeto do mesmo tipo definido pelo tipo de retorno do método.

O metamodelo da VIRTUOSI formaliza o relacionamento entre um comando de retorno e uma referência a objeto, conforme mostra a Figura A.36.

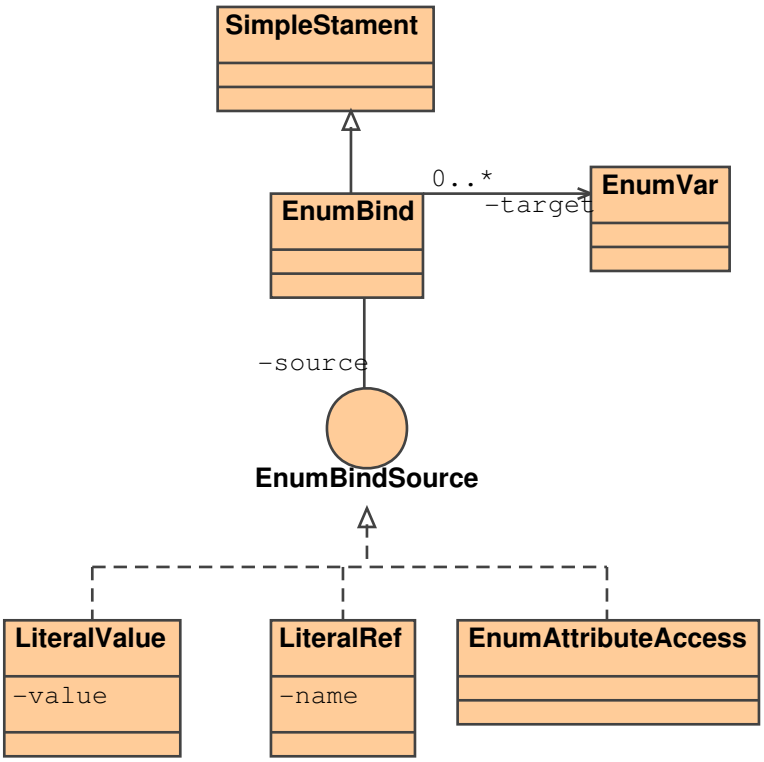

Figura A.35: Relacionamento da meta-classe comando de atribuição variável enumerada com outros componentes do metamodelo da VIRTUOSI

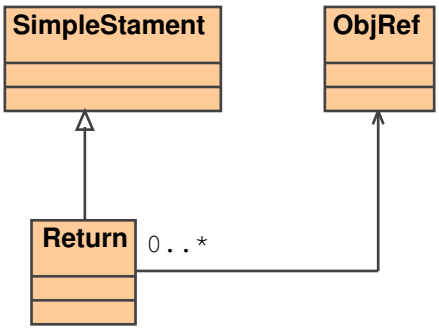

Figura A.36: Relacionamento entre um comando de retorno e uma referência a objeto

A.1.7.5 Retorno de Ação

Conforme explicado na Seção A.1.6, uma ação não retorna uma referência, ao invés disso, tem como retorno um comando *desvie* ou um comando execute. Um comando resultado de teste é o comando retornado por todos os componentes do metamodelo que são testáveis. Tanto o comando resultado de teste quanto os comandos testáveis são detalhados na discussão do comando de desvio condicional nessa Seção.

A.1.7.6 Invocação de Invocáveis

O metamodelo da VIRTUOSI define os comandos para realizar a invocação de construtores, métodos com retorno e métodos sem retorno. A invocação de um construtor é realizada a partir do nome da classe que o possui. Já a invocação de um método, com ou sem retorno, é realizada a partir de uma referência a objeto. Tanto o comando de invocação de método com retorno quanto o comando de invocação de método sem retorno podem ser generalizados como comandos de invocações de método. O comando de invocação de método e o comando de invocação de construtor podem ser generalizados como comandos de invocação.

A Figura A.37 mostra o relacionamento da meta-classe que representa um comando de invocação com sua hierarquia e outros componentes do metamodelo da VIRTUOSI.

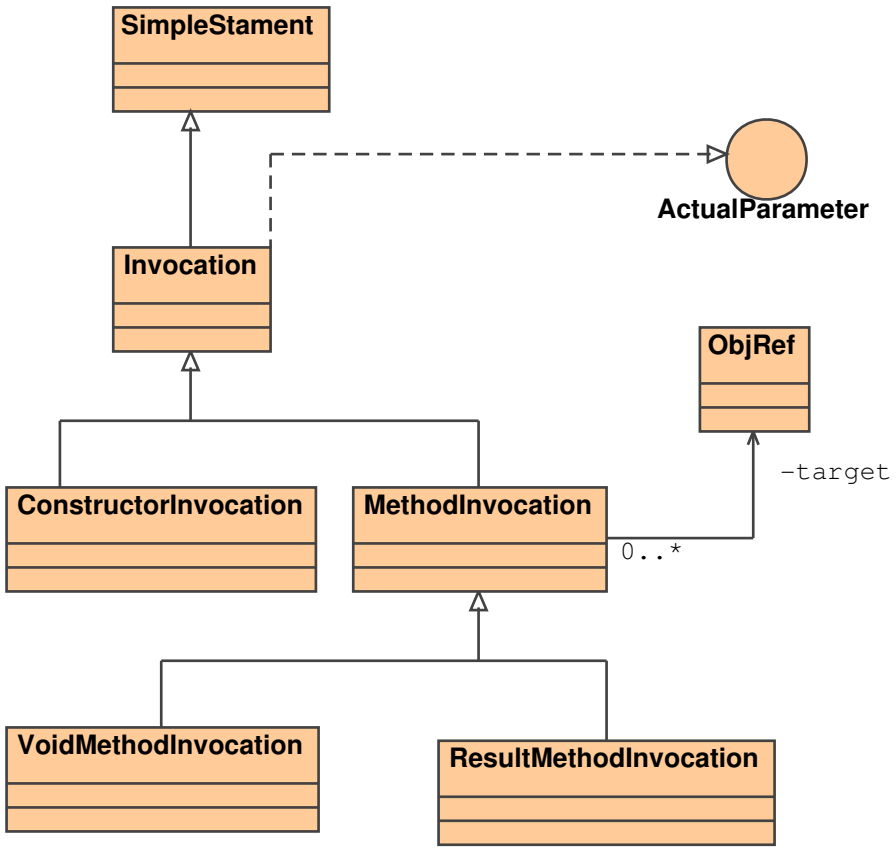

Figura A.37: Relacionamento da meta-classe que representa um comando de invocação com sua hierarquia e outros componentes do metamodelo da VIRTUOSI

O metamodelo da VIRTUOSI formaliza a relação entre os comandos de invocação e os respectivos invocáveis, conforme mostra a Figura A.38.

Conforme a Figura A.38 mostra, um comando de invocação de construtor causa a interpretação da seqüência de comandos definidos em um construtor; um comando de invocação de método sem retorno causa a interpretação da seqüência de comandos definidos em um método sem retorno; um comando de invocação de método com retorno

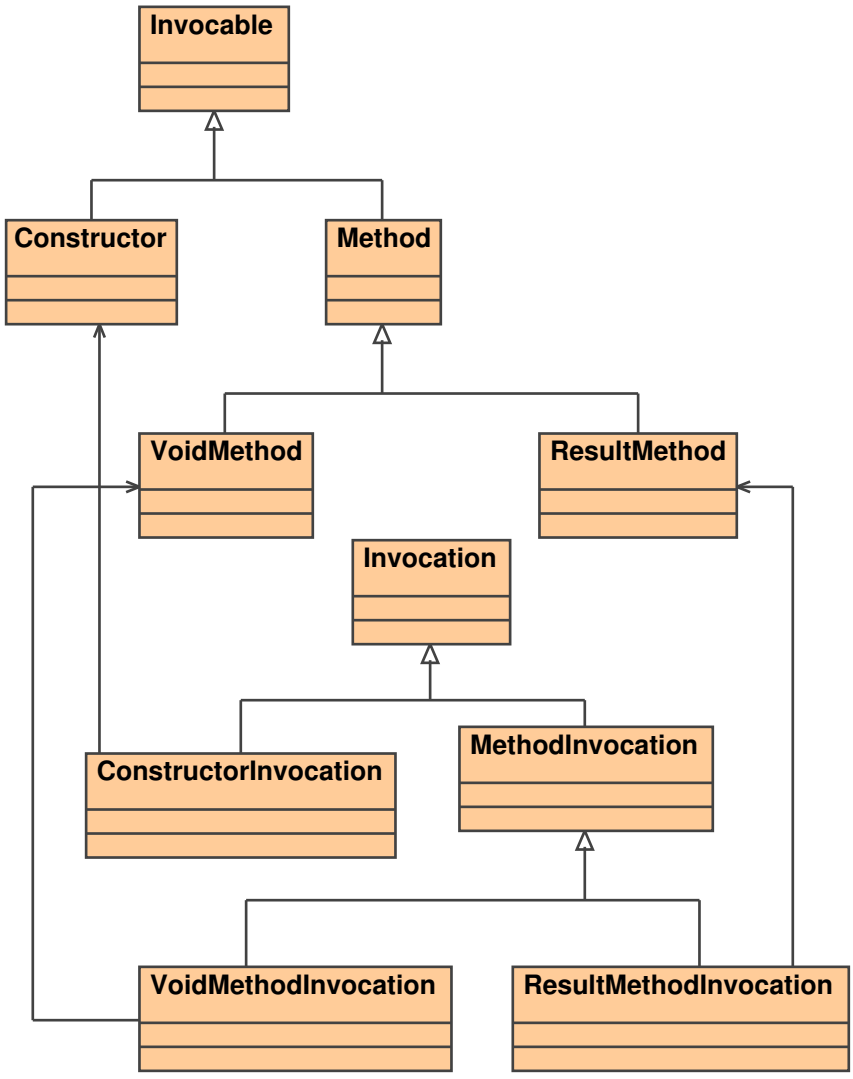

Figura A.38: Relação entre os comandos de invocação e os invocáveis

causa a interpretação da seqüência de comandos definidos em um método com retorno.

Deve-se notar que uma ação, embora seja um componente invocável, não possui um comando para invocação correspondente. A invocação de uma ação é abordada em detalhe na discussão sobre componentes testáveis.

A.1.7.7 Desvios

Dois comandos simples são responsáveis por controlar o fluxo de interpretação dentro de uma seqüência de comandos, a saber:

- **desvio incondicional**;
- **desvio condicional**.

Desvio Incondicional: Um comando de desvio incondicional quando interpretado faz com que uma seqüência de comandos específica seja interpretada. Nesse contexto essa seqüência de comandos é chamada de **caminho destino**. Um desvio incondicional não aparece explicitamente no código fonte, ele sempre é utilizado em conjunto de um comando de desvio condicional.

Desvio Condicional: Um desvio condicional tem duas seqüências de comandos que podem ser interpretados (exclusivamente). A primeira seqüência é chamada de **caminho destino** e a segunda é chamada **caminho alternativo**.

Testável: Na VIRTUOSI, para se interpretar um comando de desvio, não é necessário avaliar o valor de um objeto da classe *Boolean*, embora isso possa ser feito.

Um comando de desvio condicional está associado a algo que pode ser testado, um **testável**[8]. Um testável, quando interpretado, retorna um dentre dois comandos possíveis, a saber:

- comando **execute** ou ;
- comando **desvie**.

Caso o comando retornado seja o comando execute, isto faz com que o caminho destino seja interpretado. Caso o comando retornado seja o comando desvie, o caminho alternativo é interpretado.

Visto que ambos os comandos – execute e desvie – são os dois resultados possíveis de um testável, diz-se que ambos são comandos resultado de teste.

O metamodelo da VIRTUOSI formaliza os comandos de desvio e como estes se relacionam com as seqüências de instruções, conforme mostra a Figura A.39.

Entre os componentes definidos como testáveis pelo metamodelo da VIRTUOSI, está a invocação de uma ação. Uma ação, embora seja um componente invocável, não possui um comando de invocação correspondente. Uma ação também é invocada a partir de uma referência a objeto, contudo, sua utilização *sempre* está associada a um comando de desvio condicional, ou seja, a invocação de uma ação é realizada a partir de uma referência a objeto somente quando esta invocação for o elemento testável de um comando de desvio condicional.

Deve-se notar a diferença entre uma ação e uma invocação de ação. Uma invocação de ação causa a interpretação da seqüência de comandos definidos por uma ação. Um comando de desvio condicional interpreta um testável, para que este retorne um resultado de teste. O testável em questão, *pode* ser uma invocação de ação. A invocação de ação interpreta cada um dos comandos definidos pela ação até encontrar um comando resultado de teste. Esse resultado de teste é utilizado pelo comando de desvio condicional.

O metamodelo da VIRTUOSI formaliza a relação de um desvio condicional, um testável, uma invocação de ação e uma ação, conforme mostra a Figura A.40.

A Figura A.41 mostra um exemplo de utilização de comando de desvio condicional cujo testável é uma invocação de ação a partir de um objeto da classe pré-definida *Integer*.

[8]Do inglês ***testable*** (o termo testável ainda não está registrado nos dicionários da língua portuguesa).

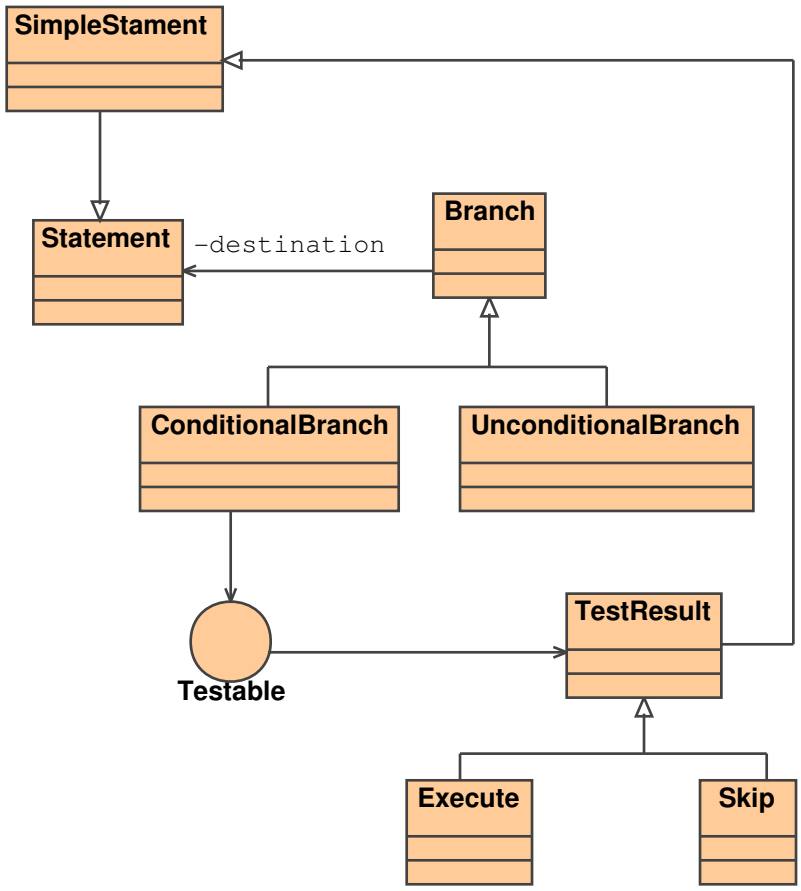

Figura A.39: Instruções de desvio como se relacionam com as seqüências de instruções

A Figura A.42 mostra um exemplo de utilização de comando de desvio condicional cujo testável é uma comparação de valor entre uma variável enumerada e um valor literal.

Essa abordagem é bem diferente da abordagem normalmente utilizada por linguagens de programação, onde um desvio condicional sempre depende da avaliação de uma expressão que retorna um valor verdadeiro ou falso. Isto permite, por exemplo, que qualquer classe defina uma ou mais ações que podem ser utilizadas para a tomada de decisão em um desvio condicional. Além disso, toda classe pode definir uma ação chamada `default` ou ação padrão. Esta ação padrão não precisa ser explicitamente chamada, ou seja, se um comando de invocação tiver como teste simplesmente uma referência a objeto, isto significa que a ação invocada deve ser a padrão. A Figura A.43 mostra o exemplo da definição de uma ação padrão e seu respectivo uso por um comando de desvio. Portanto, embora o funcionamento seja diferente, é possivel utilizar o valor de um objeto da classe *Boolean* como testável de um comando de desvio.

Além de uma invocação de ação, um testável pode ser:

- uma comparação entre duas referências a objeto, chamada **comparação de referência a objeto** – utilizada para verificar se ambas as referências apontam para o mesmo objeto em memória;

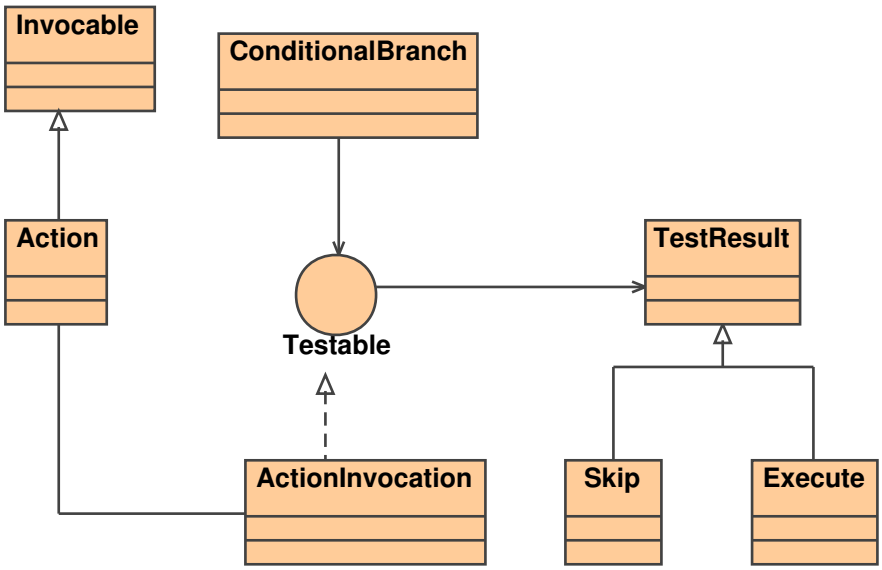

Figura A.40: Relação de um desvio condicional, um testável, uma invocação de ação e uma ação

```
class Pessoa
{
   ...
   action obesa() exports all
   {
       ...
       if (massa_.gt(h))// gt significa greater than
           execute;
       else
           skip;
   }
 ...
}
```

Figura A.41: Desvio condicional com um testável que é uma invocação de uma ação – código fonte em Aram

- uma comparação entre duas referências a bloco de dados, chamada **comparação de referência a bloco de dados** – utilizada para verificar se ambas as referências apontam para a mesma seqüência de dados binários em memória;
- uma comparação entre duas referências a índice, chamada **comparação de referência a índice** – utilizada para verificar se ambas as referências apontam para a mesma posição em uma mesma seqüência de dados binários em memória;
- uma comparação entre uma referência a objeto e uma referência nula, chamada

```
class Boolean
{
   enum { true, false } value = false;

   ...
   method void flip( ) exports all
   {
       if ( value == true )
           value = false;
       else
           value = true;
   }
}
```

Figura A.42: Desvio condicional com um testável que é uma comparação de valor entre uma variável enumerada e um valor literal – código fonte em Aram

comparação de referência nula a objeto- utilizada para verificar se uma referência é nula;

- uma comparação entre uma referência a bloco de dados e uma referência nula a bloco de dados, chamada **comparação de referência nula a bloco de dados** – utilizada para verificar se uma referência é nula;
- uma comparação entre uma referência a índice e uma referência nula, chamada **comparação de referência nula a índice** – utilizada para verificar se uma referência é nula;
- uma comparação de **valor** entre uma variável enumerada e um segundo elemento do tipo variável enumerada, valor literal, referência a literal ou ainda um acesso a atributo variável enumerada, chamada **comparação de valor de variável enumerada** – utilizada para comparar valores;

O metamodelo da VIRTUOSI formaliza todos elementos testáveis que podem ser utilizados por um desvio condicional, conforme mostra a Figura A.44.

A.1.7.8 Repetição

Utilizando um comando de desvio condicional associado a um comando de desvio incondicional (não visível no código fonte) é possível realizar o comportamento de uma estrutura de repetição no estilo *enquanto–faça* ou *faça–enquanto* conforme a Figura A.45 mostra.

```
class Boolean
{
   enum { true, false } value = false;
   ...
   action default() exports all /ação padrão da classe Boolean
   {
       if (value==true) {
           execute;
       }
       else {
           skip;
       }
   }
   ...
}
class Principal
{
   constructor iniciar() exports all
   {
       ...
       Boolean entrou = corsa.entrarPassageiro(andrea);

       if (entrou) {// uso da ação padrão da classe Boolean
           Integer distancia = Integer.make(10);
          ...
       }
       ...
   }
   ...
}
```

Figura A.43: Definição de uma ação padrão e seu respectivo uso por um comando de desvio

A.1.7.9 Vazio

O metamodelo da VIRTUOSI define um comando que não causa nenhum efeito, esse comando é chamado de comando vazio.

A.1.8 Chamadas de Sistema

Conforme citado na Seção A.1.4, o ambiente VIRTUOSI disponibiliza uma biblioteca de classes pré-definidas (*Integer*, *Real*, *Boolean*, *Character* e *String*) para a construção de novas classes chamadas de classes de aplicação.

Contudo, o ambiente VIRTUOSI também disponibiliza comandos simples e testáveis de sistema para a construção de classes que utilizem referências a bloco de dados. As próprias classes pré-definidas disponibilizadas pelo ambiente VIRTUOSI são construídas

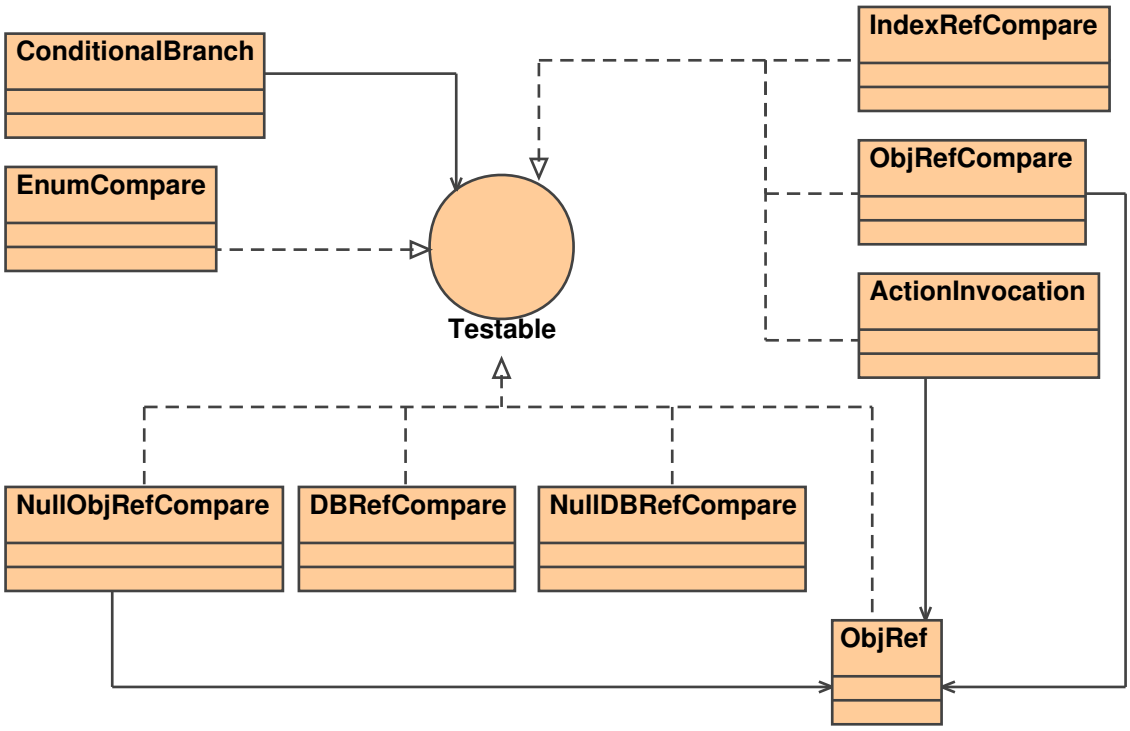

Figura A.44: Relação de um desvio condicional com todas os *testables* possíveis

```
class Principal
{
   constructor iniciar() exports  all
   {
        ...
        Integer t = Integer.make(2);
        while (andrea.obesa()) {
            andrea.emagreca(t);
        }
   }
}
```

Figura A.45: Estrutura de repetição realizada pela combinação de um desvio condicional e um desvio incondicional – código fonte em Aram

com estes comandos e testáveis de sistema.

Para cada uma das classes pré-definidas existe um conjunto definido de comandos de sistema e um conjunto definido de testáveis de sistema.

A.1.8.1 Comandos de Sistema

Os comandos simples disponibilizados pelo ambiente VIRTUOSI são chamados de comandos de sistema. Com o intuito de organizar a hierarquia das meta-classes que representam estes comandos, o metamodelo da VIRTUOSI define uma hierarquia de comandos de sistema, conforme mostra a Figura A.46.

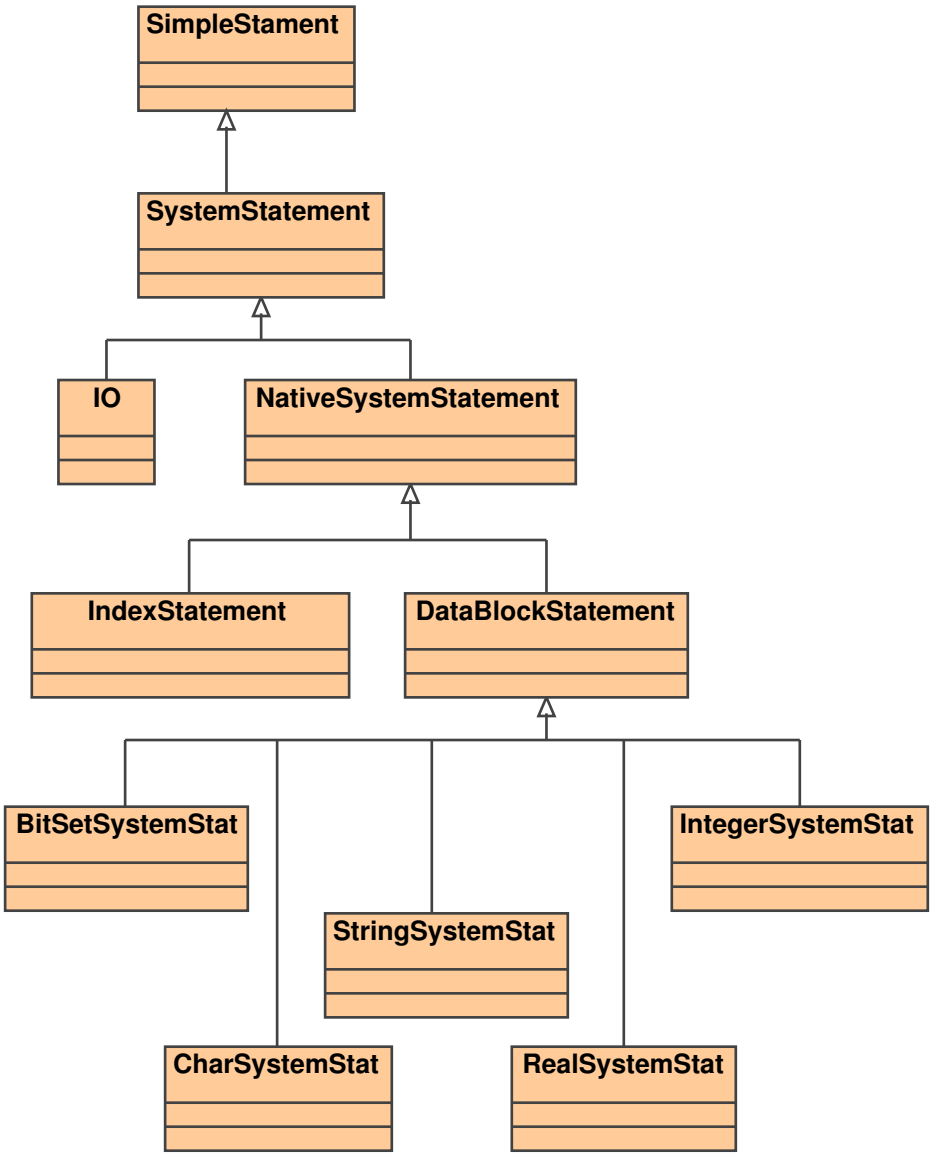

Figura A.46: Comandos de sistema disponibilizados pelo sistema

Observa-se que abaixo do comando de sistema, existem os comandos de entrada e saída e os comandos nativos.

Abaixo dos comandos nativos, existem dois tipos de comando:

- comandos para a manipulação de bloco de dados;
- comandos para manipulação de índices.

Abaixo dos comandos para manipulação de bloco de dados, existem cinco tipos de comando:

- comandos de sistema para seqüência de dados binários, que manipulam dados binários de forma neutra;
- comandos de sistema para valores inteiros;
- comandos de sistema para valores reais;
- comandos de sistema para valores caracter;
- comandos de sistema para valores conjunto de caracter;

A Figura A.47 mostra um exemplo de código fonte de uma classe pré-definida fornecida pelo ambiente VIRTUOSI, no caso uma classe para manipulação de valores inteiros, que utiliza dois comandos de sistema: *createDB* – representado no código fonte pela invocação do construtor make e *storeInteger* – utilizado para armazenar uma seqüência de dados binários no formato apropriado para a representação de números inteiros.

```
class Integer
{
   datablock value;

   constructor make( literal k ) exports all
   {
       value = datablock.make( 32 );
       // 32 é um valor literal utilizado para especificar
       // o tamanho do bloco de dados.
       value.storeInteger( k );
   }
   ...
}
```

Figura A.47: Uso de dois comandos de sistema para facilitar a construção de uma classe pré-definida *Integer* – código fonte em Aram

Observa-se que um comando sistema – disponibilizado pelo sistema – pode retornar uma referência para bloco de dados ou referência para índice. Nota-se também, que através da utilização desses comandos de sistema, o programador pode, por exemplo, definir novas classe que armazenem valores inteiros com bloco de dados de tamanho diferente. Isso é possível porque os comandos de sistema que manipulam bloco de dados com a representação de valores inteiros podem receber como parâmetros um valor literal indicando o tamanho do bloco de dados que deve ser criado.

A.1.8.2 Testáveis de Sistema

Os testáveis disponibilizados pelo ambiente VIRTUOSI são chamados de testáveis de sistema. Com o intuito de organizar a hierarquia das meta-classes que representam estes testáveis, o metamodelo da VIRTUOSI define uma hierarquia de testáveis de sistema, conforme mostra a Figura A.48.

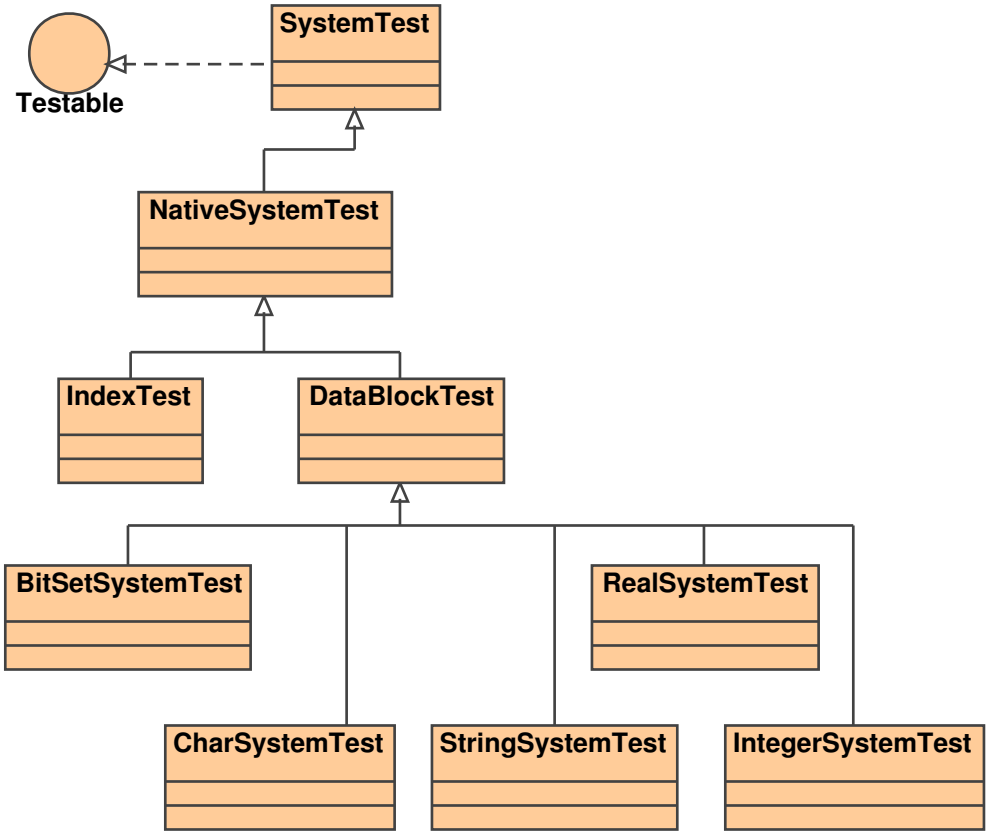

Figura A.48: Testáveis de sistema disponibilizados pelo sistema

Observa-se que abaixo da meta-classe representado os testáveis de sistema, existe somente os testáveis nativos.

Abaixo dos testáveis nativos, existem dois tipos de testáveis:

- testáveis para a manipulação de bloco de dados;
- testáveis para manipulação de índices.

Abaixo dos testáveis para manipulação de bloco de dados, existem cinco tipos de testáveis:

- testáveis de sistema para seqüência de dados binários, que manipulam dados binários de forma neutra;
- testáveis de sistema para valores inteiros;

- testáveis de sistema para valores reais;
- testáveis de sistema para valores caracter;
- testáveis de sistema para valores conjunto de caracter;

A Figura A.49 mostra um exemplo de código fonte de uma classe pré-definida fornecida pelo ambiente VIRTUOSI, no caso uma classe para manipulação de valores inteiros. O exemplo mostra a implementação de duas ações da classe *Integer*. A ação chamada *equals* faz uso de um testável de sistema para seqüência de dados binários chamado *sameBits*. A ação chamada *greaterOrEqual* faz uso de um testável de sistema para valores inteiros chamado *geq*, que retorna um comando execute caso o valor inteiro armazenado no bloco de dados seja maior ou igual ao passado como parâmetro.

```
class Integer
{
   datablock value;
   ...
   action equals( Integer i ) exports { all }
   {
       databalock k = i.value;
       if ( value.sameBits( k ) )
           execute;
       else
           skip;
   }
   action greaterOrEqual( Integer i ) exports { all } // greater or equal
   {
       databalock k = i.value;
       if ( value.geqInteger( k ) )
           execute;
       else
           skip;
   }
}
```

Figura A.49: Uso de testáveis especiais nos comandos de desvio utilizados na construção de uma classe pré-definida *Integer* – código fonte em Aram