Uma Proposta para Tratamento de Exceções em uma Linguagem Orientada a Objeto

Isa Haro Martins
Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente, 225
22453 Rio de Janeiro, RJ, Brasil
isa@inf.puc-rio.br
PUCRioInf - MCC17/93

Abstract

This article discusses a proposal and specification for Exception Handling (EH) in Object Oriented Languages. The idea is to adapt conventional mecanisms of EH to the basic principles of Object Orientation. The proposal has two main golas. One of them is to respect the philosophy of classes and methods so that the EH resource can be easily absorbed by the user. Another goal is to provide flexibility in what refers to the scope of the handler (generic or specific), to the level of coupling between handler and code (instructions or methods), and to the moment of definition of the handler (in modeling time of the object or in programming time).

Keywords: exception handling, object oriented, programming language, reuse, encapsulation.

Resumo

O presente artigo propõe uma discussão e uma especificação para o tratamento de exceções em uma Linguagem Orientada a Objetos. O trabalho busca adaptar mecanismos de tratamento de exceções já conhecidos na programação convencional, a uma Linguagem Orientada a Objetos, respeitando princípios básicos desta orientação. A proposta tem como meta dois pontos principais. Um deles é respeitar a filosofia de trabalho com classes e métodos já absorvida pelo usuário. O objetivo é que o recurso de tratamento de exceções seja naturalmente incorporado pelo usuário, por fazer parte do arcabouço conceitual de uma linguagem que ele já domina. O segundo ponto é, dentro dos princípios já citados, prover a linguagem de flexibilidade. Esta flexibilidade se refere ao escopo do tratador (genérico ou específico), ao nível de associação do tratador ao código de programa (instruções ou métodos), e à definição do tratador (em tempo de modelagem do objeto ou em tempo de programação).

Palavras Chave: tratamento exceções, orientação objeto, linguagem programação, reuso, encapsulamento

SUMÁRIO

Resumo 2

Introdução 4

1. As Bases da Linguagem Orientada a Objeto 4

1.1. Mecanismos de Abstração da Linguagem Orientada a Objetos 4

1.2. Princípios Básicos da Linguagem Orientada a Objetos 6

2. Análise do Problema 7

2.1. Diferentes Abordagens Conceituais ao Tratamento de Exceções 8

2.2. A Busca do Tratador 9

2.3. Unidade Responsável pela Exceção 10

2.4. Associação do Tratador à Ocorrência da Exceção 12

2.5. O Acesso do Tratador ao Contexto da Exceção 12

3. Proposta Semântica 12

3.1. Semântica de Decisões sobre o Tratador de Exceções 13

3.2. Semântica de Operação dos Tratadores de Exceção 17

4. Proposta Sintática 19

4.1. Sintaxe para Declaração do Tratador de Exceções 19

4.2. Sintaxe para Associação do Tratador de Exceções 20

5. Comentários sobre Casos Especiais 21

Conclusões 22

Referências 23

ANEXO 1 24

Introdução

Este trabalho propõe uma especificação para o tratamento de exceções em uma Linguagem de Programação Orientada a Objetos (LOO). Foi feita uma adaptação, para a LOO, de mecanismos de tratamento de exceções já conhecidos para linguagens convencionais. Ele foi desenvolvido tomando como referência o TOOL [SPA92], uma linguagem de programação orientada a objetos e dirigida a eventos. O objetivo não foi definir o tratamento de exceções para essa linguagem, mas através dela discutir a adequação do tratamento de exceções à filosofia de uma LOO.

Para tanto será feito um breve resumo dos princípios da linguagem, dos mecanismos de abstração que ela oferece, e uma discussão sobre a incorporação do tratamento de exceções a esse contexto, tirando partido das peculiaridades de uma LOO. A seguir será apresentada uma proposta sintática e semântica a ser usada numa implementação, e serão discutidos alguns casos especiais.

A proposta apresentada poderá ser adequada a outras linguagens que obedeçam os mesmos princípios, devendo ser adaptada à sua implementação.

1. As Bases da Linguagem Orientada a Objeto

Uma Linguagem Orientada a Objeto busca dar suporte a uma metodologia de programação voltada a objetos e eventos. Para tanto ela provê mecanismos de abstração como classes (classes), métodos (methods) e manipuladores de mensagens (message handlers). Não é nossa proposta questionar os mecanismos de abstração de uma LOO, nem mesmo seus princípios básicos, apenas trabalharemos a partir deles. No entanto sua observação é de extrema importância, uma vez que este trabalho se propõe a incorporar um novo recurso, o tratamento de exceções. Para preservação da coerência da linguagem é fundamental que este novo recurso seja discutido a nível conceitual, metodológico e operacional, obedecendo rigidamente os princípios estabelecidos.

O conceito de classe em uma LOO não é claramente definido. A literatura sobre o assunto oferece diferentes definições sobre o que é uma classe e seu conteúdo. Faz-se assim necessário definir o que se entende, neste trabalho, por classe e método.

1.1. Mecanismos de Abstração da Linguagem Orientada a Objetos

Falaremos brevemente sobre esses mecanismos, apenas o suficiente para embasar o raciocínio que determina a solução apresentada neste trabalho. Em [SPA92] pode ser encontrada informação mais detalhada sobre o assunto.

- Classes

A classe é um agrupamento de informações que modela um objeto. Um objeto tem basicamente um conteúdo e um comportamento. A classe agrupa a definição dos dados que compõem um objeto e as operações que podem ser executadas sobre eles, além de constantes e assertivas referentes a esse objeto. A *declaração de classes* feita através da LOO modela objetos da parcela de mundo real sobre a qual se vai construir um sistema. A *declaração de um objeto* modelado pela classe cria uma instância desse objeto para o sistema.

A característica mais forte da classe é a herança. As classes podem ser definidas hierarquicamente, isto é, classes podem ser definidas a partir de outra classe já existente, herdando todos os seus componentes, sucessivamente. A classe herdeira é chamada subclasse e a classe pai é chamada superclasse.

A hierarquia de classes permite que a modelagem dos objetos seja feita a partir de objetos simples. A essa definição vão sendo acrescentados novos componentes, dados e operações, criando novas classes que modelam objetos mais complexos. Cada uma das classes intermediárias pode ser inúmeras vezes reutilizada na modelagem de novos objetos, desde que o objeto que ela modela seja parte do novo objeto a ser modelado. Vide exemplo no Anexo 1, onde a classe Professor contém somente os dados específicos do objeto Professor, o restante foi herdado da classe anterior. Portanto, numa árvore hierárquica de classes o que caracteriza um objeto é o somatório dos componentes das classes, no caminho entre a raiz e o objeto em questão.

Redeclarações podem existir nesse caminho. Uma classe pode conter uma redeclaração de comportamento de um objeto, declarado numa superclasse. Nosso modelo não permite redeclarações de dados.

A característica de herança de classe é uma coluna de apoio da metodologia de LOO e será fortemente utilizada no tratamento de exceções.

- Método

O método, comparativamente a uma linguagem convencional, é o equivalente a uma função ou subrotina. Ele pertence a uma classe e é um agrupamento de operações sobre o objeto que essa classe modela. O método representa o comportamento do objeto. Vale lembrar que parte desse comportamento pode ser herdado.

- Manipulador de Mensagens

O manipulador de mensagens também se assemelha a uma função ou subrotina de uma linguagem convencional, no sentido de ser um agrupamento de operações. A diferença, com relação ao método, está em dois pontos fundamentais. O primeiro é o fato do manipulador de

mensagens ser assíncrono, a unidade de programa que o aciona continua sua execução, independente de resposta. A segunda é com relação ao retorno de informações, diferentemente do método, que pode retornar para a unidade chamadora o resultado de uma operação ou outra informação qualquer, ele não retorna informações.

A necessidade de um mecanismo assíncrono na linguagem surge da proposta do TOOL de permitir ao usuário uma interface de mais alto nível na utilização das bibliotecas XLib. A utilização dessas bibliotecas se dá por meio de mensagens assíncronas. Na prática esse mecanismo também se mostra útil na otimização do processamento, em situações onde o sincronismo não é necessário, o uso do manipulador de mensagens mostra-se mais vantajoso que o uso do método.

Em resumo, as classes e os métodos traduzem o mecanismo de abstração básico da linguagem, sendo os manipuladores de mensagens uma opção surgida de uma necessidade específica. Portanto para efeito de captação do modelo da linguagem podemos concentrar nossa análise no conceito de classes, na questão da herança, e na força da ligação entre os dados e as operações, existente nesse modelo.

1.2. Princípios Básicos da Linguagem Orientada a Objetos

Os princípios básicos sobre os quais uma linguagem se assenta refletem a necessidade que orientou sua geração. É fundamental portanto que todo recurso incorporado à linguagem respeite esses princípios. Neste trabalho consideramos os seguintes princípios: reutilização (reuso), extensibilidade, encapsulamento, proteção. Falaremos sobre cada um deles e durante a análise apontaremos de que forma as eventuais soluções se alinham ou ferem alguns desses princípios. A análise foi desenvolvida no sentido de dotar o recurso de tratamento de exceções da maior flexibilidade possível, dentro dos limites dos princípios da linguagem.

Os princípios de extensibilidade e reutilização caminham juntos. Um dos problemas da programação moderna é a complexidade e o tamanho dos sistemas, e consequentemente o tamanho das equipes de trabalho. Por isso torna-se essencial a possibilidade de reaproveitamento de código de programa, tanto pela divisão do trabalho em equipes como pela reutilização de código propriamente dita. Essa reutilização exige flexibilidade na estrutura do código de programa, isto é, o programa deve ser codificado em unidades que sejam favoráveis à reutilização. Essas unidades podem estar disponíveis em bibliotecas permitindo compor programas a partir de código já existente.

A reutilização do código seria bastante restrita se se limitasse à possibilidade de composição de programas a partir dessas unidades. A idéia de extensibilidade vem quebrar essa limitação. A extensibilidade refere-se ao conteúdo semântico dessas unidades de programa. Todo código de programa modela algum tipo de ocorrência no mundo real. Extender uma especificação significa utilizar uma unidade de programa já existente extendendo a ocorrência por ela modelada. Portanto, uma linguagem que oferece extensibilidade permite reutilizar as unidades de programa extendendo sua modelagem.

A classe é o mecanismo que permite a reutilização e a extensibilidade. Ela é uma unidade de código de programa que modela um objeto do mundo real, por isto nós a chamamos de unidade lógica de programa. Através do mecanismo de herança ela permite a extensão dessa modelagem, ou seja, uma classe definida pode herdar o conteúdo de uma já existente, acrescentando o seu próprio, e com isto modelando um novo objeto.

O mais importante da classe é que ela não é apenas um agrupamento de operações como uma rotina ou uma função em uma linguagem tradicional, que atenderiam em menor escala à necessidade de reutilização. A classe encapsula dados e operações sobre esses dados, numa unidade de programa. Ela modela por completo um objeto, seu conteúdo e seu comportamento. No nosso entender, esse encapsulamento é um dos princípios de apoio da linguagem mais importantes. Na verdade é o encapsulamento que perrmite a especificação de unidades de programa com um conteúdo e uma função tão bem definidas. Essa distinção e clareza permitem a reutilização e extensão do código de maneira tão eficiente.

Em função do encapsulamento a classe pode ser vista pelo usuário como uma entidade abstrata. Não importa para o usuário de classes ter domínio sobre *como* uma determinada operação é realizada mas *quais* operações se aplicam sobre um determinado objeto.

Para preservar o encapsulamento a linguagem provê mecanismos de proteção para a classe. Esses mecanismos só permitem o acesso aos dados de uma classe através de métodos pertencentes a ela. Uma operação só pode ser executada sobre os dados se ela está declarada como pertencente à classe desses dados. Essa proteção gera uma forte ligação entre os dados e seu comportamento. A vantagem dessa forte ligação entre dados e operações existe a nível de programa, como segurança, mas existe principalmente a nível conceitual e de compreensão do usuário sobre o comportamento do objeto.

Uma vez que o usuário tenha absorvido a filosofia de trabalho com classes, em uma LOO, o domínio sobre a programação é muito maior. Por um lado porque trabalhando com entidades abstratas ele se despreocupa de detalhes, ele tem garantias do que pode ser executado sobre o objeto que aquela entidade representa. Por outro lado, na situação em que ele tenha necessidade de ter conhecimento detalhado, o encapsulamento dos dados e operações em uma unidade de programa provê o acesso a essas informações com maior clareza.

Através deste resumo pretendemos mostrar a importância destes princípios e, consequentemente, porque o tratamento de exceções deve respeitá-los. No modelo proposto o tratador de exceções representa o comportamento de exceção do objeto, da mesma forma que o método representa o comportamento normal. Para que não se quebre a filosofia de modelagem do objeto, essa forte ligação entre seu conteúdo e seu comportamento deve ser preservada.

2. Análise do Problema

Com base nos princípios citados da LOO, avaliaremos alguns pontos chave do tratamento de exceções. Falaremos sobre as diferentes formas possíveis de fazer o tratamento de exceções, qual delas se adapta melhor à filosofia da linguagem, oferecendo melhores resultados. A discussão será feita sob o enfoque mais genérico possível. A partir daí apresentaremos a proposta de solução.

2.1. Diferentes Abordagens Conceituais ao Tratamento de Exceções

Segundo Ghezzi [G&J82] a exceção é uma ocorrência difícil de caracterizar. Exceções podem ocorrer na codificação de instruções, por exemplo uma instrução tipo CASE onde apareça um valor no campo de referência para o qual não tenha sido especificada uma saída; ou em função de um dado, por exemplo uma divisão por zero; ou por uma falha no acesso ao hardware; e outras. Na verdade o procedimento de exceção não caracteriza necessariamente uma situação de erro, mas uma situação que foge ao comportamento normal.

Em Rodriguez [Rod90] pode ser encontrada uma análise comparativa das várias abordagens ao conceito de exceção, e as vantagens e desvantagens advindas de cada uma delas. Dessa análise alguns pontos nos chamam a atenção. Na abordagem encontrada em PL/I [Nob68], a primeira linguagem a propor o conceito de exceção, parte das exceções definidas referem-se a erros, como por exemplo divisão por zero, e parte se refere ao controle do fluxo para situações de programa esperadas mas de ocorrência imprevisível, como por exemplo ENDFILE. Goodenough [Goo75] define conceitualmente que uma condição de exceção é uma situação que permite o controle, como falha ou como validade de resultados, podendo o invocador monitorar a operação. Levin [Lev77] sugere que o mecanismo de tratamento de exceção pode ser usado para distinguir situações de programa, considerando algumas como normais e outras como exceção.

De modo geral, a abordagem encontrada neste tabalho entende a exceção como um comportamento fora do comportamento normal e previsível, porém no mesmo nível conceitual deste. Nossa abordagem está próxima da abordagem descrita em Levin, no sentido de conceitualização do comportamento de exceção no mesmo nível de abstração do comportamento normal do programa. Essa opção oferece flexibilidade de programação e, no contexto de classes com que estamos operando, ela pode ser absorvida naturalmente pelo usuário. Porque, mantendo todo o comportamento do objeto no mesmo nível ela respeita o arcabouço conceitual da linguagem, já conhecido e dominado pelo usuário. Nossa solução difere daquela apresentada em Dony [Don90] onde é proposta uma classe de exceções para uma LOO. Entendemos que o encapsulamento do comportamento do objeto, normal e de exceção, é mais favorável à compreensão e manipulação deste por parte do usuário.

Lembrando Goodenough, consideramos também a necessidade de mecanismos de controle robustos no sentido da capacidade do invocador monitorar a ocorrência. Essa monitoração será objeto de discussão quanto a mecanismos de sinalização para diferentes cadeias de invocação possíveis.

Diante da situação definida surgem duas perguntas. Nas diferentes situações de exceção a que unidades do programa interessa dar conhecimento da ococrrência da exceção? Existem diferentes tipos de exceção que justifiquem diferentes tipos de tratadores? A primeira pergunta será discutida no próximo item, onde falaremos sobre a busca do tratador. Quanto à segunda, entendemos que não, porque estamos falando de tratadores como um mecanismo abstrato e portanto genérico. O que capacita esse mecanismo genérico a atuar em situações diferentes são fatores como:

- unidade do programa responsável pelo tratamento da exceção
- tipo de busca feita ao tratador
- a flexibilidade de associação do tratador a operações ou unidades do programa
- o acesso do tratador ao contexto de ocorrência da exceção

Esses pontos serão detalhados a seguir. Para distinguir classe de método, enquanto unidade de código de programa, daqui para a frente algumas vezes estaremos nos referindo a classes como unidade lógica de programa e a métodos como unidade de operações.

2.2. A Busca do Tratador

A discussão básica é definir qual unidade do programa é responsável pelo tratamento da exceção. Uma abordagem possível para esse problema é a responsabilidade do tratamento da exceção ser atribuída ao ambiente onde ela foi gerada, o qual chamaremos de unidade de programa sinalizadora da exceção, ou apenas unidade sinalizadora. Outra, é a responsabilidade do tratamento ser atribuída ao ambiente onde foi invocada a unidade de programa na qual ocorreu a exceção, o qual denominaremos unidade chamadora.

Nessa discussão é necessário distinguir duas coisas, embora se vá falar delas ao mesmo tempo: a responsabilidade pelo tratamento da exceção e o caminho a ser seguido na busca do tratador. Em uma linguagem convencional talvez esta questão ficasse mais clara, porque as duas coisas se sobrepõem. No caso, se a responsabilidade é da unidade sinalizadora, ela ativa o tratador. Se a responsabilidade é externa, ou seja, de uma unidade chamadora, existe um caminho através do qual pode ser buscado um tratador, é a cadeia de invocações, que levou à chamada da unidade sinalizadora. Neste caso deve ser questionado o limite de propagação da exceção.

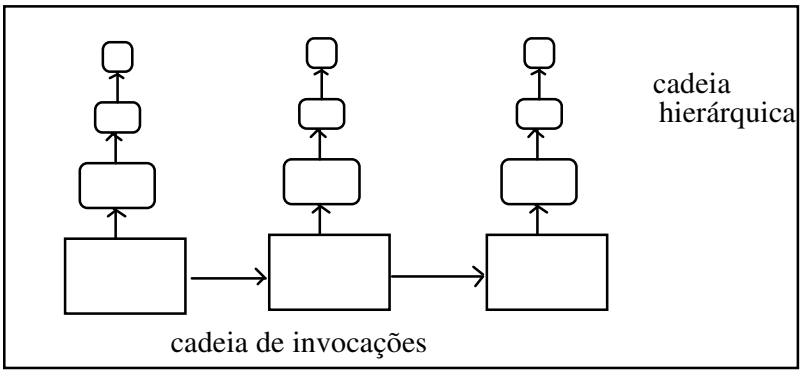


Figura 1. Cadeia de invocações e cadeia hierárquica na execução de um programa

Mas em se tratando de uma LOO, onde há classes e herança, as coisas se confundem, porque existem duas estruturas presentes. Uma delas é a cadeia de invocações semelhante a uma linguagem tradicional, que vai encadeando os métodos atribuídos aos objetos. Outra é a cadeia hierárquica. Cada método atribuído a um objeto pode ter atráz de si uma cadeia hierárquica. Por exemplo, o tratamento de uma interface onde um método CriaJanela é atribuído a um objeto, dentro dele um método TrataBotão é atribuído a outro objeto. Neste caso a cadeia de invocações é MóduloInicial->CriaJanela->TrataBotão, porém o método CriaJanela tem atráz de si uma cadeia hierárquica de objetos e comportamentos, da mesma forma o método TrataBotão tem atráz de si outra cadeia hierárquica de objetos e comportamentos. Uma vez determinado que a responsabilidade pelo tratamento da exceção é da unidade sinalizadora o caminho a ser seguido na busca do tratador é a cadeia hierárquica. Mas se for determinado que a responsabilidade é da unidade chamadora, ambas as cadeias podem ser seguidas. Ou seja, o tratamento de exceções pode ser feito através das unidades lógicas do programa ou das unidades de operações. A Figura 1 ilustra essa situação.

2.3. Unidade Responsável pela Exceção

Dizer que a unidade sinalizadora é responsável pela exceção, significa dizer que esta unidade vai passar o controle para o tratador associado quando ocorrer a exceção. Evidentemente a unidade tem controle sobre qual é o tratador a ser utilizado. Vamos analisar duas abordagens possíveis na busca do tratador. Através da unidade sinalizadora e da unidade chamadora.

2.3.1. Unidade Sinalizadora Responsável pela Exceção

Foi dito que a classe modela um objeto definindo seu conteúdo e seu comportamento. Se abordarmos o problema dentro da filosofia da LOO, a classe deve ter recursos para definir o comportamento normal do objeto, o método, e recursos para definir o comportamento de exceção, o método ou equivalente. Ao modelar o objeto o construtor de classes especifica o

comportamento total do objeto, normal e de exceção. Este comportamento estará encapsulado na classe, ligado aos dados nela definidos.

avendo um tratador definido na classe, valerá para ele toda a semântica de herança e redeclaração, válidas para as classes em geral. Toda classe herda o comportamento de exceção de uma superclasse. Se uma classe contém uma redeclaração de um comportamento de exceção herdado, este é substituído a partir daquele nível hierárquico. Portanto, ocorrendo uma exceção na execução de um método, ele tem a responsabilidade de passar o controle da execução para o tratador, que em princípio será buscado dentro da própria classe. Não havendo um tratador declarado na classe, será utilizado um tratador herdado de uma superclasse, seguindo a linha hierárquica.

A esta abordagem do problema chamaremos encadeamento hierárquico na busca do tratador. Existem grandes vantagens nessa abordagem. A principal delas é a coerência com a filosofia da linguagem, aproveitando os recursos que ela oferece. Outra vantagem é que, pelo fato do módulo sinalizador ser responsável pela passagem de controle ao tratador, este tem acesso ao contexto onde ocorreu a exceção. Outra ainda, é que ela permite que o controle seja passado para o tratador, retornando ao módulo em execução após ação de "saneamento", caso seja essa a opção semântica da linguagem. É interessante observar que nesta abordagem se preserva o princípio de proteção, uma vez que apenas o comportamento declarado na classe está sendo atribuído ao objeto.

Por outro lado, o aspecto de encapsulamento oferecido pelas classes, que é extrema vantagem das LOO, pode ser visto como uma pequena restrição no caso do tratamento de exceções. Em algumas situações o tratamento de exceções definido pelo construtor de classes em *tempo de modelagem do objeto* pode não ser aquele conveniente para o usuário de classes no momento de sua utilização, o qual chamaremos de *tempo de programação.* Isto também ocorre para o método. Um método especificado pelo construtor de classes pode não atender ao usuário de classes, o que o forçaria a uma redeclaração deste método ou declaração de outra classe. Mas este não é um procedimento tão comum uma vez que o método espelha o comportamento do objeto no mundo real. No caso específico do tratamento de exceções, entendemos que a própria natureza do problema induz a uma flexibilidade maior de comportamento. Isto torna desejável a possibilidade de redeclarar *em tempo de programação* o tratamento de exceção de um objeto, conforme o desejo do usuário, sem por isto ter de declarar o comportamento normal, ou seja, o método.

De um modo geral estamos falando na possibilidade do programador interferir no comportamento de exceção estabelecido para o objeto, sem ferir os princípios da linguagem. Optando-se por esta flexibilidade, o importante é manter a responsabilidade do tratamento de exceção com a unidade sinalizadora, no momento da redeclaração deste comportamento, mantendo a abordagem de busca hierárquica do tratador. Neste caso seria necessário um artifício para associar à unidade sinalizadora um tratador não pertencente à classe imediata, nem à cadeia hierárquica.

2.3.2. Unidade Chamadora Responssável pela Exceção

Esta abordagem associa o tratador ao método em tempo de programação. A unidade responsável por transferir o controle de execução ao tratador é a unidade chamadora, por isso agora a definição do tratador não mais é feita pelo construtor de classes e sim pelo usuário de classes em tempo de programação. Neste caso teoricamente poder-se-ia buscar o tratador tanto na cadeia hierárquica da unidade chamadora quanto na cadeia de invocações.

A busca na cadeia hierárquica significaria a ligação do tratador aos dados daquela unidade lógica. Uma vez que os dados a serem tratados pertencem a outra unidade lógica, essa busca não tem muito sentido em termos de uma linguagem, porque embora um tratador da cadeia hierárquica possa eventualmente ser utilizado, essa abordagem não pode ser generalizada. A abordagem que pode ser generalizada é a busca através da cadeia de invocações, que chamaremos encadeamento de uso. Esta é uma abordagem tradicional, ela ignora os princípios da linguagem expostos para o encadeamento hierárquico. Aqui a exceção é propagada através da cadeia, em busca de um tratador.

A vantagem desta abordagem é exatamente a flexibilidade que foi citada acima, a capacidade de tratar a exceção na situação em que o objeto é utilizado, não na situação em que ele é modelado, o que permite tratamentos específicos. As desvantagens são exatamente as vantagens da forma anterior, a indisponibilidade dos módulos da cadeia hierárquica sobre o contexto de ocorrência da exceção no momento em que lhe é devolvido o controle, e a impossibilidade de devolver o controle ao módulo sinalizador após a atuação do tratador.

Esta abordagem quebra o conceito mais puro da LOO, uma vez que parte do comportamento do objeto não está embutido na classe. Mas não quebra a idéia de encapsulamento no sentido de que o objeto se mantém como uma entidade abstrata, não sendo dada visibilidade da classe ao módulo chamador. Apenas é feita uma passagem de valores referentes à exceção para os módulos chamadores.

2.4. Associação do Tratador à Ocorrência da Exceção

Até agora, por facilidade de compreensão, pensamos no tratador associado ao método. Isto é, um mesmo tratador atendendo a qualquer ocorrência de exceção na execução do método. Esta é uma situação bastante genérica que obviamente não atende a todos os casos. É necessário então uma forma de associar mais de um tratador ao método para atender ocorrências diferentes.

Outra solução é a possibilidade de associar o tratador a diferentes níveis de código do programa, por exemplo, associá-lo a instruções ou ao método, priorizando as diferentes associações, permitindo que associações específicas se sobreponham às genéricas. Este mecanismo oferece uma grande flexibilidade, porque permite a criação do tratador genérico, que atende a maior parte dos casos, inclusive ocorrências imprevistas que podem ser orientadas para o tratador automaticamente, e permite criar tratadores específicos, sem que um interfira com o outro.

2.5. O Acesso do Tratador ao Contexto da Exceção

O aspecto de acesso ao contexto de ocorrência da exceção já foi indiretamente comentado nos itens anteriores. A questão que se coloca é a visibilidade, ou não, do contexto da exceção. Dentro da estrutura da LOO, se tratador e método pertencem à mesma classe, ou seja, se ambos modelam o comportamento do mesmo objeto, o tratador tem visibilidade sobre os dados desse objeto. Se o tratador não pertence à classe do objeto ao qual foi atribuído o método, ele não tem visibilidade sobre o contexto da exceção, sendo necessária uma passagem de parâmetros para o tratador.

3. Proposta Semântica

Em função da argumentação de vantagens e desvantagens apresentada e do interesse em oferecer algumas facilidades aos usuários, optamos por uma solução mista, em termos de busca do tratador. Nossa proposta utiliza-se da busca na cadeia hierárquica, complementando essa busca, através de um pequeno artifício, para prover a linguagem de maior flexibilidade na associação de tratadores às ocorrências de exceções.

Entendemos como importante que a solução adotada preserve a redigibilidade e a legibilidade da linguagem [Ghezzi82]. Partimos do princípio que a adoção de uma "solução de força", com o objetivo de ganhar flexibilidade, pode representar uma perda mais que um ganho se ferir algum desses aspectos. Portanto, nossa solução foi bastante cuidadosa nesse sentido.

Esta proposta semântica distingue um procedimento mínimo de tratamento de exceção e procedimentos mais complexos. Esse procedimento mínimo comporta: a interrupção da unidade sinalizadora da exceção, o retorno à unidade chamadora e a sinalização de ocorrência da exceção. Ele é obrigatório e deve ser oferecido pela linguagem, ou seja, tratadores de exceção definidos pelo usuário não descartam esse procedimento mínimo. Como a unidade sinalizadora é o método, a unidade de programa interrompida por uma exceção é sempre o método. Abaixo justificaremos esta opção.

Tratadores de exceção mais complexos podem ser definidos. Mais de um tratador pode existir capaz de atender a mesma situação, sendo a linguagem provida de critério de escolha entre eles. Portanto especificamos a semântica de decisão sobre o tratador a ser utilizado e a semântica de operação dos tratadores de exceção.

3.1. Semântica de Decisões sobre o Tratador de Exceções

A semântica de decisão sobre o tratador a ser utilizado considera os seguintes pontos:

- tratadores de diferentes ordens
- associação de tratadores a instruções ou métodos
- associação de tratadores em tempo de modelagem ou tempo de programação
- prioridade entre os tratadores associados
- busca hierárquica dos tratadores.

3.1.1. Ordem do Tratador

Primeiramente vamos esclarecer que não está sendo considerada a possibilidade de um tratamento de exceção definido dentro do próprio método, uma vez que esse procedimento não permitiria a distinção clara entre o comportamento normal e o de exceção. Além disso, o comportamento de exceção ficaria totalmente restrito ao método não havendo garantias da execução dos procedimentos mínimos já citados. Por questão de metodologia optamos por exigir a distinção, no código do programa, entre método e tratador.

Existem tratadores de quatro ordens diferentes, que podem ser acionados através de dois mecanismos de tratamento. Do mais genérico para o mais específico, os tratadores são: tratadores default da linguagem, tratadores específicos da linguagem, tratadores default do usuário e tratadores específicos do usuário. Chamamos de tratador default aquele genérico, tanto da linguagem como do usuário. O tratamento da exceção pode ser acionado através de um mecanismo automático ou assinalado pelo programador. Na Figura 2 encontra-se um quadro elucidativo das diferentes ordens de tratadores e sua ativação automática ou explícita.

Quanto ao tipo de tratamento, entende-se por tratamento automático a situação em que os tratadores existentes não têm uma associação explícita aos possíveis eventos. Nestes casos a linguagem oferece uma busca automática, obedecendo regras de prioridade. É o caso mais genérico, sendo acionado somente na ausência de um tratador explicitamente assinalado pelo programador.

	tratador da linguagem	tratador do usuário
genérico (default)	Automático	Automático
específico	Automático	Explícito

Figura 2. Quadro relacionando diferentes ordens de tratadores e o tipo de ativação

Quanto ao tipo de tratador, a linguagem pode oferecer tratadores específicos, para exceções pré-definidas ou um tratador default. Como exemplo de situações para possíveis tratadores específicos temos o caso de uma divisão por zero ou uma falha de hardware num acesso a um dado (supondo-se que o Sistema Operacional devolva o controle ao programa). Ou ainda, uma instrução do tipo Case onde não exista saída default e o campo de referência seja diferente das opções apresentadas. Na ocorrência de um desses eventos, não havendo o tratador específico, a linguagem ofereceria o tratador default.

Tratadores definidos pelo programador também podem ser genéricos ou específicos. O tratador genérico é chamado tratador default do usuário. Ele é declarado sempre pertencendo a uma classe. Ele é único na classe e por isso não tem nome particular, sendo identificado apenas como um tratador genérico da classe. Não haverá instrução do programador no sentido de associá-lo aos métodos. Ele é automaticamente associado a todos os métodos daquela classe. Sua simples presença na classe faz com que ele seja acionado em caso de ocorrência de uma exceção na execução do método. Obviamente o tratador default é sujeito ao comportamento hereditário das classes, uma classe herda o tratador default de sua superclasse, um tratador default pode ser redeclarado em uma subclasse. Esta é uma situação ainda genérica, mas sob controle do usuário.

Outra ordem de tratador é o específico do usuário. Este é declarado pelo usuário e identificado por um nome. Uma classe pode conter um ou mais tratadores específicos. Através do nome eles podem ser explicitamente associados a uma instrução ou um método. O tratador específico tem prioridade sobre os defaults. Também é sujeito ao comportamento hereditário das classes.

3.1.2. Nível de Associação do Tratador

A associação do tratador pode ser feita em dois níveis de programa: um método ou uma instrução. Associar o tratador ao método é mais genérico, significa associá-lo a um agrupamento de instruções. A ocorrência de uma exceção em qualquer uma delas aciona o mesmo tratador. Essa associação pode ser automática, no caso do tratador default da linguagem ou do default do usuário. Ou pode ser explícita, no caso de um tratador específico do usuário.

A associação ao nível da instrução é mais específica, somente uma exceção na execução daquela instrução aciona o tratador. Esta associação tem prioridade sobre aquela ao nível do método. Obviamente para as instruções sem tratadores associados valerá o tratamento associado ao método.

3.1.3. Momento de Associação do Tratador

Esta é a situação mais delicada no modelo proposto. Na verdade este é um artifício do qual lançamos mão para oferecer ao usuário uma flexibilidade maior. Na seção de discussão do

problema foi levantada a conveniência de poder-se associar um tratador ao método em tempo de programação. Isto é, um tratador que atendesse a uma situação de uso da classe, onde fossem tomadas medidas mais de acordo com a situação, do que aquelas medidas declaradas pelo construtor de classes no momento da modelagem do objeto. Estamos propondo a associação de um tratador específico a um método no momento da atribuição deste método a um objeto.

O que transforma isto numa situação excepcional é o fato desse tratador não pertencer à classe que modela o objeto. Mas embora ele não pertença a essa classe de fato, ele pertence logicamente, na medida em que se propõe a substituir um tratador nela declarado. E por pertencer logicamente a essa classe ele não pode pertencer a outra.

A "solução de conciliação" adotada foi a exigência de que este tratador seja declarado em uma classe independente, sem superclasse e sem subclasse, uma classe que contenha somente esse tratador. Observe-se o fato de que neste caso embora o tratador seja associado ao método através da unidade chamadora, a responsabilidade pelo tratamento da exceção continua sendo da unidade sinalizadora. Isto significa que embora a associação seja indicada na unidade chamadora, o tratador está logicamente ligado à unidade sinalizadora. O artifício gera uma mudança física na declaração de associação, de uma unidade para outra, mas não uma mudança lógica. Essa ligação lógica do tratador à classe é a garantia de que o artifício está causando um rompimento mínimo na filosofia da linguagem. Os problemas ocorrem com relação à necessidade de passagem de parâmetros para o tratador, uma vez que os dados do contexto da exceção estão protegidos contra métodos que não pertençam àquela classe, e com relação à prioridade entre o tratador mais recentemente associado ao método e os tratadores associados a instruções dentro do método.

Como foi dito, esta é uma solução de conciliação, que se não chega a agredir profundamente a filosofia da linguagem, também não está em total harmonia com ela.

3.1.4. Prioridade entre os Tratadores

Das situações apresentadas com relação a tratadores default e específicos, associação de tratadores a nível de método e instrução, associação de tratadores em tempo de modelagem e programação, percebe-se a necessidade de estabelecer prioridades entre os diferentes tratadores associados, em função das redundâncias.

As seguintes regras devem ser respeitadas pela linguagem. O tratador do usuário tem prioridade sobre o tratador oferecido pela linguagem. O tratador especificado pelo usuário em tempo de programação tem prioridade sobre aquele especificado em tempo de modelagem. A associação a nível de instrução tem prioridade sobre a associação a nível de método. O tratador específico tem prioridade sobre o tratador default.

Uma situação difícil existe com relação ao tratador associado em tempo de programação. Como este tratador é associado ao método, pelas regras ele não tem prioridade sobre um tratador associado a uma instrução daquele método. Isso pode gerar uma situação de conflito, uma vez que o usuário tenta substituir um comportamento estabelecido na modelagem do objeto, mas só

consegue fazê-lo parcialmente. Pois, para as instruções com tratadores associados será mantido o comportamento antigo. Se, ao contrário, for estabelecida uma semântica de prioridade absoluta para o tratador mais recente, associado ao método, fazendo com que se sobreponha às associações mais específicas, isto poderá gerar problemas por substituir tratamentos específicos por tratamentos genéricos.

De qualquer forma, apesar das dificuldades, por julgar que os benefícios dessa flexibilidade são importantes para o usuário, optamos pela segunda solução. Em primeiro lugar porque sendo uma solução de conciliação é mais sensato que ela seja absoluta. Isto significa que é "mais legível" para o usuário compreender que todo comportamento de exceção do método foi substituído e não apenas parte dele. Em segundo lugar porque entendemos que os métodos tendem a ser, por si só, bastante específicos, executando operações bem definidas com relação aos dados. Isso não exigiria o uso em grande escala de tratadores associados a instruções, ao contrário, a expectativa é de que a associação do tratador ao método seja utilizada, tanto para tratadores específicos como defaults.

Se tivéssemos optado por uma semântica de retorno do tratador para a unidade sinalizadora, essa solução seria inadmissível. Porque os tratamentos aplicados às exceções específicas das instruções seriam substituídos por um tratamento genérico, o que poderia causar sérios problemas para a continuidade de execução da unidade sinalizadora.

3.1.5. Regra de Busca do Tratador

Respeitando a característica de herança simples de classes das LOO, todo tratador específico explicitamente associado a um método ou a uma instrução pode estar declarado na classe onde foi feita a associação ou em uma superclasse. Por isso a busca do tratador é feita através da cadeia hierárquica.

Caso existam na hierarquia tratadores específicos com o mesmo nome valem os mais recentes, obedecendo, para tratadores, a mesma capacidade de redeclaração que a linguagem oferece para os métodos. Da mesma forma ocorre com os tratadores default. Todo tratador default definido em um classe tem prioridade sobre os tratadores default herdados. Na ocorrência de exceção em uma classe que não tenha nenhum tratador declarado, será buscado o tratador default mais próximo através da cadeia hierárquica. Exceção a essa busca na hierarquia é o caso do tratador associado no momento de atribuição do método a um objeto. Este caso já foi apresentado.

Graficamente o caminho a ser trilhado na decisão sobre o tratador apresenta-se na forma da Figura 3, com a maior prioridade indicada de cima para baixo.

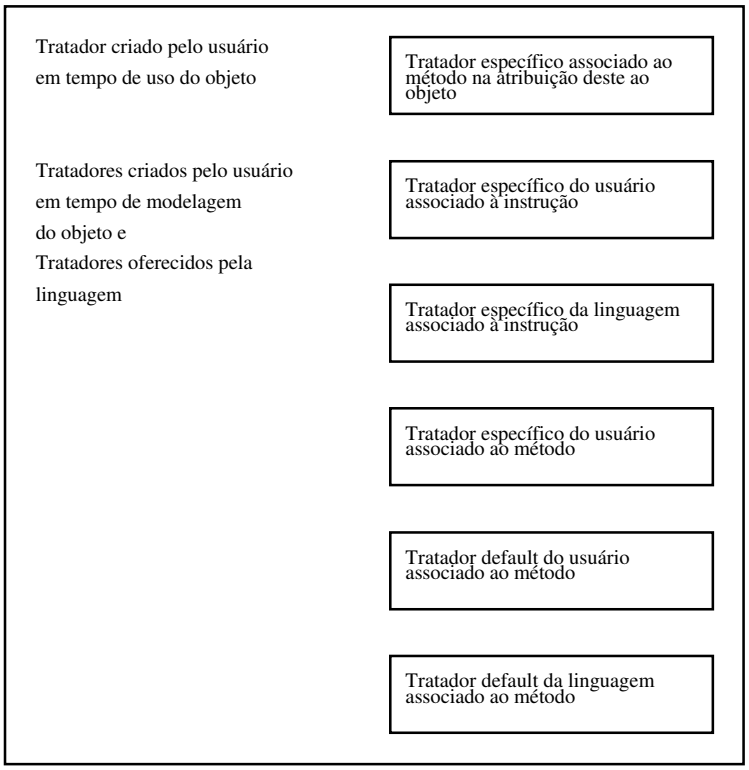


Figura 3. Prioridades entre os tratadores de exceção associados

3.2. Semântica de Operação dos Tratadores de Exceção

A semântica de operação dos tratadores de exceção utilizados considera os seguintes pontos:

- o comportamento obrigatório de um tratador de exceções
- o acesso ao contexto da exceção
- a ocorrência de falha na execução do próprio tratador

3.2.1. Controle de retorno do tratador e sinalização de exceção

Duas operações diferenciam um tratador de exceções de um método, o ponto de retorno após a execução e o código de retorno gerado. De acordo com o modelo adotado, onde o tratador é associado ao método ou à instrução, não havendo necessidade de retorno à unidade chamadora para ativação do tratador, poderia ter sido adotada uma solução de tratamento e retorno ao ponto de sinalização da exceção. Isto é, o tratador de exceções seria responsável por medidas de saneamento das áreas de dados, que permitissem o retorno ao ponto de ocorrência da exceção e a continuidade de execução da unidade sinalizadora. Porém, esta não é a solução proposta.

Nossa proposta é a interrupção da unidade sinalizadora da exceção. O retorno do tratador é feito para a unidade chamadora. Isto significa que o processamento da unidade sinalizadora é descontinuado. A tarefa a ser executada por aquela unidade é suspensa. Mesmo nos casos em que o tratador é associado às instruções, o que nos induz a pensar mais de imediato num tratamento de recuperação da área de dados afetada, optamos por interromper a execução do método.

A justificativa dessa opção está no fato de que mesmo que o tratador execute medidas de saneamento, na maioria dos casos essas medidas poderão gerar resultados falsos. Por exemplo, que medida de saneamento seria possível no caso de uma divisão por zero, sem comprometer o resultado da operação? Não executar a operação seria a opção mais segura. Há casos em que, sem dúvida, a possibilidade de um tratamento e retorno à execução é bastante atraente. Mas, enquanto proposta de uma linguagem, necessariamente genérica, julgamos ser mais segura a interrupção da execução da unidade sinalizadora, a transferência da execução para o tratador, a sinalização da exceção através de um código de retorno e o retorno à unidade chamadora mais externa.

É obrigatório que a unidade de programa interrompida sinalize para a unidade chamadora a ocorrência da exceção. Porque embora o tratamento esteja vinculado à unidade sinalizadora, a unidade chamadora necessita a informação de que a tarefa solicitada foi executada com falha ou sucesso. Tanto o retorno, quanto o código de retorno são feitos automaticamente pela linguagem, após o uso do tratador.

É importante ressaltar que a solução adotada não foi compulsória, foi uma opção. Com relação à semântica anteriormente estabelecida não haveria problema se se optasse por retornar ao ponto de sinalização após a execução do tratador. O único caso de exceção a essa afirmação é a solução adotada nas regras de prioridade para os tratadores associados ao método em tempo de programação. Mas esta é uma solução para um problema específico, que não tem a ver com a idéia geral de tratamento de exceções na LOO, e que portanto pode ser facilmente substituída.

3.2.2. Acesso do tratador ao contexto da exceção

Temos duas situações possíveis, aquela em que o tratador pertence à mesma cadeia hierárquica do método e aquela em que o tratador é associado ao método no momento de atribuição deste ao objeto. No primeiro caso o tratador "reconhece" o objeto ao qual foi atribuído o método, basta que lhe seja dado acesso à instância declarada desse objeto. Embora o método em questão possa estar tratando com apenas alguns campos do objeto, o tratador declarado na classe tem acesso a todos os campos do objeto, áreas de constantes, áreas de trabalho, e outras.

A segunda situação é aquela em que os dados devem ser passados como parâmetros para o tratador.

3.2.3. Ocorrência de falha na execução do tratador

Em caso de falha na execução do tratador ele deverá ser interrrompido e o controle devolvido ao módulo chamador, sinalizando a falha do tratador. Portanto, o código de retorno da execução de

um método pode sinalizar duas coisas, a ocorrência de uma exceção no método com consequente ativação do tratador e a execução do tratador com sucesso ou falha.

4. Proposta Sintática

Nesta seção será apresentada a sintaxe para declaração do tratador de exceções e associação do tratador, nas diferentes situações possíveis de utilização deste recurso. A sintaxe adotada está próxima da linguagem Tool, utilizada como referência.

4.1. Sintaxe para Declaração do Tratador de Exceções

O tratador de exceções é considerado um operador declarado pelo usuário (user declared operator) tal como os métodos e os manipuladores de mensagens, onde os métodos retornam o controle ao ponto de chamada e podem retornar o valor de um objeto, os manipuladores de mensagens são assíncronos e não retornam valores, e os tratadores não retornam à unidade de sinalização mas à unidade chamadora e devolvem um código de sinalização da exceção. Em função desta distinção optou-se pelo nome Exception para caracterizá-lo.

Os tratadores podem ser declarados como defaults ou específicos. Abaixo um exemplo de declaração de um tratador específico:

```
Exception TrataCalculaSalario (parametros)
        -- declarações locais
begin
        -- comandos
end Exception
```

O tratador default não tem nome, apenas a especificação. Por exemplo:

```
Exception  (parametros)
        -- declarações locais
begin
        -- comandos
end Exception
```

4.2. Sintaxe para Associação do Tratador de Exceções

A associação do tratador de exceções pode ser explícita ou implícita. A associação implícita existe para os tratadores defaults. A associação explícita pode ocorrer a nível de instrução, método ou na atribuição do método a um objeto. A associação será sintaticamente representada por um "dois pontos" duplo.

- Asssociação do Tratador a Nível de Comando

O tratador é associado ao comando como um todo, isto é, se o comando for uma expressão numérica automaticamente o tratador é associado ao total da expressão e não ao último elemento, independente da presença de parênteses. Para tanto é obrigatório que ele seja o último elemento da expressão sintática.

```
comando :: nome-do-tratador
```

Exemplos:
salario_hora = salario / horas_mes :: TrataErroCalcSalario

Switch valor-referência Case 1 Processa1
Case 2 Processa2 :: TrataErroValorRef

- Associação do Tratador a Nível de Método

É obrigatório que a associação seja o último elemento da expressão sintática.

```
Method nome-do-método :: nome-do-tratador
```

Exemplo:
Method CalculaIdade :: TrataCalculaIdade

- Associação do Tratador a Nível de Atribuição do Método a um Objeto

A associação é feita com o método, numa sugestão de que ambos são atribuídos ao objeto. É obrigatório que a associação seja o último elemento da expressão sintática.

```
objeto  <-  nome-do-método :: nome-do-tratador
```

Exemplos:
prof1 <- CalculaIdade :: NovoTrataCalcIdade

prof1 <- AlteraSalario (novo_salario) :: TrataSalario

5. Comentários sobre Casos Especiais

- Tratamento de Exceção para as Built-in Classes

Um caso interessante seria a linguagem prover tratadores defaults para as built-in classes. Um tratador default declarado em uma built-in class estaria dentro da filosofia da linguagem. Na forma como foi especificada nossa proposta, na ausência desse tratador o problema poderá ser contornado através da associação de um tratador à instrução. Mas esta solução apenas contorna o problema, não resolve, uma vez que um tratador associado a uma expressão numérica não vai saber exatamente em que momento do cálculo da expressão ocorreu um erro. Ao passo que um tratador associado à built-in class poderia acusar exatamente a ocorrência do erro.

- Tratamento da Exceção para os Manipuladores de Mensagens

O manipulador de mensagens é parte do código de programa da linguagem Tool. Como foi dito, é uma unidade assíncrona que não retorna valor à unidade que a ativou. No entanto existe a possibilidade de ocorrência de exceção na execução de uma manipulador de mensagens. Não há sentido em que o programa aguarde qualquer código de retorno do manipulador avisando sobre a exceção, porque o assincronismo seria perdido. Por isso, mesmo que houvesse forma de associar-se um tratador de mensagens ao manipulador, o programa principal não poderia ser avisado da ocorrência da exceção e da ativação do tratador. Neste trabalho desconsideramos a possibilidade de associação de um tratador a um manipulador de mensagens. Mas esta não é uma questão esgotada, ela valeria por si só um novo trabalho de pesquisa.

Conclusões

Este trabalho apresentou uma proposta de adaptação do recurso de tratamento de exceções, conhecido para linguagens convencionais, a uma LOO. Um dos objetivos da solução adotada foi manter a clareza da linguagem de programação. Outro objetivo foi dotá-la de flexibilidade para o tratamento de diferentes situações de exceção possíveis. Para tanto foi proposto um tratamento modelando o comportamento de exceção do objeto no mesmo nível em que o método modela o comportamento normal do objeto. O recurso foi dotado de flexibilidade quanto ao escopo do tratador e à associação do tratador à ocorrência da exceção. Foi definido um esquema de prioridades para os casos de mais de um tratador ativo.

Na solução apresentada duas questões são passíveis de redefinição, a interrupção da unidade sinalizadora após a ocorrência da exceção, e a associação do tratador de exceções ao método no momento de sua atribuição ao objeto. A primeira, mudaria o perfil do tratador e sua utilização, sem no entanto afetar a proposta do ponto de vista conceitual. A segunda, pode ser redefinida em função do ônus com relação à metodologia de LOO, acarretado pela flexibilidade oferecida ao usuário. Eventualmente, o projetista de linguagens poderá optar por uma solução menos flexível para o usuário e de maior rigidez com relação à metodologia. As duas questões se relacionam, e se afetam mutuamente. Portanto a redefinição de uma poderá ocasionar a redefinição da outra.

Apontamos como vantagem o fato de que a solução proposta pode ser resolvida em tempo de compilação. A sintaxe adotada manteve-se próxima da linguagem usada como referência, o TOOL. A semântica respeitou os princípios da LOO, a metodologia de trabalho que ela suporta e as vantagens que ela oferece.

Na especificação apresentada soluções intermediárias são possíveis. Há que se medir a validade das opções apresentadas no momento da implementação deste recurso a uma linguagem concreta, quando os objetivos e o público usuário da linguagem estarão claramente definidos.

Agradecimentos

A autora gostaria de agradecer Sergio E.R. Carvalho tanto pela motivação no desenvolvimento do trabalho, como pela participação na discussão das idéias.

Referências

Car92 Carvalho, Sergio E.R. **Exception Handling in Object Oriented Languages: a Proposal.** Monografias em Ciência da Computação, no. 23/92, PUC-Rio, Departamento de Informática, 1992.

Don90 Dony, C. **Exception Handling and Object Oriented.** OOPSLA ECOOP'90 Proceedings, SigPlan Notices, Vol. 25, Num. 10, Oct/90.

G&J82 Ghezzi, Carlo & Jazayeri, Mehdi **Conceitos de Linguagens de Programação**. Editora Campus, 2a. Edição, 1987.

Goo75 Goodenough, J.B. **Exception Handling: Issues and a Proposed Notation**. *Comm. ACM*, 18(2), Dec/75.

Lev77 Levin, Roy **Program Structures for Exceptional Condition Handling**. Pittsburgh, PA, Carnegie-Mellow University, Tese de Doutorado, Jun/77.

Lis79 Liskov, Barbara & Snyder, Allan **Exception Handling in CLU.** *IEEE Transaction on Software Engineering.* SE5(6) 546-558, Nov/79.

Nob68 Noble, J.M. **The Control of Exceptional Conditions in PL/1**. *Proc. IFIP Congress 68* c79-c83, Ago/68.

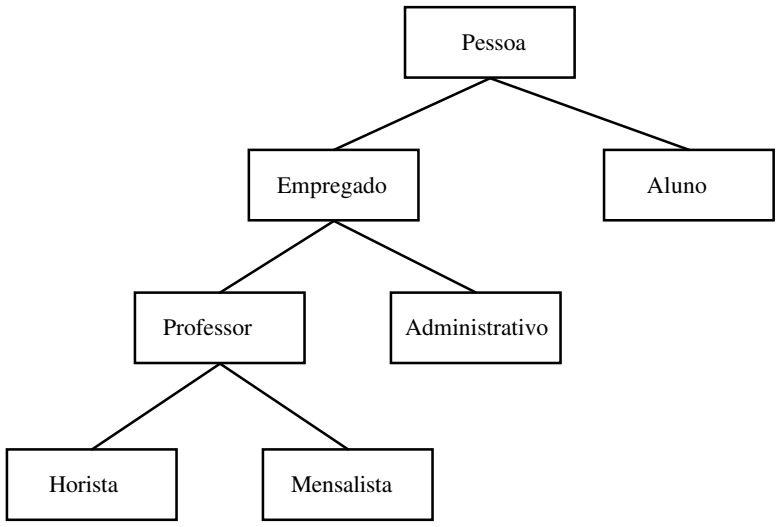
Rod90 Rodriguez, Noemi de La Rocque **Tratamento de Exceções.** Monografias em Ciências da Computação, no. 10/90, PUC-Rio, Departamento de Informática, 1990.

Rod93 Rodriguez, Noemi de La Rocque **Um Sistema de Tipos Orientado por Objetos, Incluindo Persistência**. Tese de Doutorado, PUC-Rio, Departamento de Informática, Jan/93.

Spa92 SPA, Sistemas, Planejamento e Análise. **TOOL for Windows: the Object Oriented Language**, 1992.

ANEXO 1

Pequeno exemplo de uma hierarquia de classes:



O exemplo está codificado semelhante à sintaxe da linguagem TOOL, com a declaração de classes em negrito para facilitar a leitura.

```
Class Pessoa
        representation
                string   nome, endereço, telefone;
                int      cpf, controle, identidade;
                data     nascimento;
        end representation

        method AlteraEndereco (in STRING novo_end)
                -- comandos
        end method

        method CalculaIdade
                -- comandos
        end method
end Class
```

```
Class Empregado from Pessoa
        representation
                real    identificação;
                data    admissão;
        end representation
end Class
```

```
Class Aluno from Pessoa
        representation
                real     matricula;
        end representation
end Class
```

```
Class Professor from Empregado
        representation
                string  areagraduação;
                string  anograduação;
        end representation
end Class
```

```
Class Horista from Professor
        representation

                int     horas_semanais, salário_hora;
        end representation

        method CalcSalario
                -- comandos
        end method

        method AlteraSalarioHora (in INT novo_salario)
                -- comandos
        end method
end Class
```

```
Class Mensalista from Professor
        representation
                int     salario;
        end representation

        method AlteraSalarioMes
                -- comandos
        end method
end Class


Class Administrativo from Empregado
        representation
                data    admissão;
                string  Pis;
        end representation

        method CalcTempoServico
                -- comandos
        end method

        method AlteraPis
                -- comandos
        end method
end Class
```