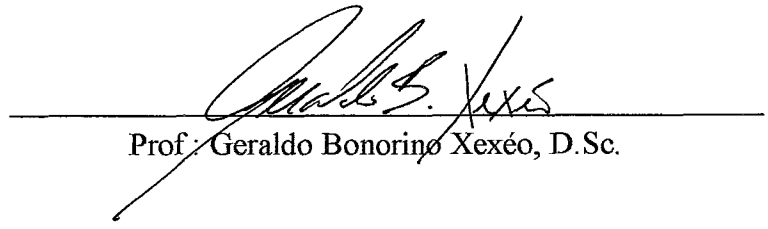
COMPARAÇÃO DIFUSA DE CLASSES BASEADA NO COMPORTAMENTO

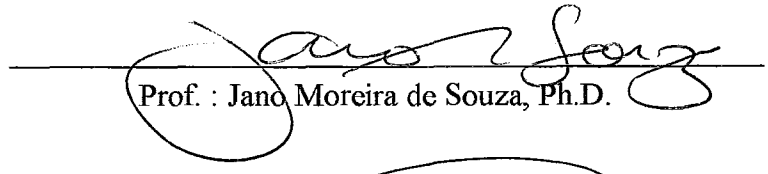
Márcio Luiz Ferreira Duran

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

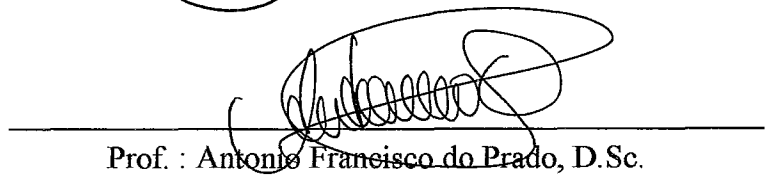
Aprovada por:



Prof.: Geraldo Bonorino Xexéo, D.Sc.



Prof. : Jano Moreira de Souza, Ph.D.



Prof. : Antonio Francisco do Prado, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 1999

DURAN, MÁRCIO LUIZ FERREIRA

Comparação difusa de classes baseada no comportamento [Rio de Janeiro] 1999

VII, 99 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Comparação de Classes
2. Busca por Comportamento
3. Integração de Esquemas
4. Reutilização de Componentes

I. COPPE/UFRJ II. Título (série)

AGRADECIMENTOS

Agradeço a Deus pela saúde e a minha família pelo carinho e compreensão durante a conclusão de mais uma etapa da minha vida.

Ao Marco, pelo apoio e por ter aberto e desbravado os caminhos da minha vida.

Agradeço à Erika pelo carinho e compreensão durante estes anos de estudo.

Ao meu orientador, pelos conselhos e idéias, essenciais à elaboração deste trabalho.

Ao professor Jano, pelas oportunidades proporcionadas e por possibilitar a iniciação no mundo acadêmico.

Ao professor Esperança e, novamente, ao Xexéo, pelo auxílio e atenção na resolução dos problemas encontrados nos laboratórios.

Agradeço ao Lúcio pela amizade e pelos ensinamentos profissionais proporcionados.

Às funcionárias da secretaria, à Patrícia Leal e à Mercedes, sempre prontas para auxiliar e agilizar os problemas burocráticos.

À CAPES, pelo apoio financeiro e o incentivo à pesquisa neste país.

Obrigado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

COMPARAÇÃO DIFUSA DE CLASSES BASEADA NO COMPORTAMENTO

Márcio Luiz Ferreira Duran

Dezembro/1999

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

A comparação de classes é uma importante etapa em qualquer processo de integração de esquemas e também na recuperação de componentes reutilizáveis de uma biblioteca de componentes.

Este trabalho apresenta uma abordagem para determinar a similaridade entre classes utilizando o seu comportamento, aperfeiçoando as técnicas tradicionais as quais usam somente informações sintáticas.

A teoria de conjuntos difusa proporciona a base matemática necessária para comparar, agregar resultados e ordenar classes de acordo com as suas similaridades.

Os resultados desta tese são uma estratégia de comparação, uma arquitetura extensível e flexível para um sistema de comparação e um protótipo implementado em um banco de dados orientado a objetos, juntamente com testes que demonstram a aplicação da abordagem porposta.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

FUZZY COMPARISON OF CLASSES BASED ON BEHAVIOR

Márcio Luiz Ferreira Duran

December/1999

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

The comparison of classes is an important task in any process of schema integration and also in the retrieval of reusable components from a software library.

This work presents an approach to determinate the similarity between classes using their behavior, improving over the traditional techniques which use syntactic information.

Fuzzy set theory provides the required mathematical backgrounds to compare, aggregate results and order class according to their similarity.

This thesis results are a comparison strategy, an extensible and flexible architecture for a comparison system and a prototype implemented in an oriented-object database, together with tests that demonstrate the applicability of the approach.

I. Introdução

I.1. Objetivo da Tese

O objetivo desta tese é propor uma técnica de comparação capaz de identificar a similaridade entre componentes. A técnica de comparação foi elaborada para o modelo orientado a objetos, onde os componentes são representados por classes, e pode ser utilizada tanto para identificar classes similares com o intuito de reutilização, quanto para facilitar o mapeamento entre classes de esquemas distintos (atividade essencial para a construção de mediadores e integração de esquemas).

Para isso, este trabalho propõe a incorporação da comparação baseada no comportamento e da lógica difusa no processo de comparação e identificação de classes similares, com o objetivo de capturar melhor a semântica existente em cada componente. Também são apresentados uma arquitetura extensível e flexível para a implementação da estratégia proposta, e um protótipo implementado em um banco de dados orientado a objetos, além de testes que demonstram a aplicação da estratégia.

I.2. Motivação

Um dos principais fundamentos para a utilização de banco de dados é a possibilidade de representar todos os dados gerenciados em uma empresa ou organização de maneira unificada e sem redundância. Aliado a isso, as corporações estão reconhecendo o valor estratégico da informação (CANNATA, 1991) e, com a expansão e ampliação da Internet e o desenvolvimento da tecnologia de redes, surge a necessidade de aplicações que processem todas as informações disponíveis de uma forma integrada.

Neste contexto, a integração de esquemas está crescendo em importância. A integração de esquemas tem como objetivo produzir um esquema global e unificado a partir dos esquemas de diversos bancos de dados. Um esquema unificado é necessário para a execução de consultas globais sobre várias bases de dados, provavelmente heterogêneas, além de ser indispensável para gerenciar banco de dados distribuídos (ÖSZU *et al.*, 1994), consultar bancos de dados federados (SHETH e LARSON, 1990) e na construção de mediadores (WIEDERHOLD, 1996) para estabelecer o

mapeamento entre o esquema do mediador e os esquemas locais às diversas fontes de informação.

Outro aspecto que tem estimulado a pesquisa na área de integração de esquemas durante os anos, foi a utilização desses conhecimentos no desenvolvimento de grandes aplicações. Neste tipo de situação, é comum encontrarmos diversas equipes de analistas trabalhando em paralelo, sendo necessário consolidar periodicamente os resultados obtidos. Neste caso, a complexidade da integração depende da liberdade que cada equipe possui durante a modelagem do sistema (SOUZA, 1986a, PRESSMAN, 1997). Um controle centralizado pode ser introduzido para evitar ou simplificar o processo de integração (TUCHERMAN *et al.*, 1985), porém diminui a execução concorrente de tarefas.

Em relação ao desenvolvimento de aplicações, outro aspecto relevante é a reutilização de componentes. A reutilização tem ganho força graças ao alto custo de desenvolvimento de software (os preços dos equipamentos caem dia após dia, mas o custo para se desenvolver um software permanece praticamente constante). Além disso, a reutilização pode acelerar o processo de desenvolvimento (diminuindo o seu custo) e aumentar a confiabilidade dos sistemas (os componentes reutilizáveis já foram testados e aprovados).

Mas aqueles que pretendem aproveitar ao máximo a reutilização enfrentam muitas dificuldades, entre elas o problema de busca e recuperação de componentes. Muitas das técnicas existentes para a recuperação de código foram adaptadas da recuperação de textos através de palavras chaves e se mostram ineficazes na recuperação de componentes.

A comparação de classes (ou componentes) é uma etapa essencial, tanto para o processo de integração de esquemas quanto para a reutilização de componentes, porém as técnicas existentes não consideram o comportamento das classes. No modelo orientado a objetos, o comportamento é responsável por grande parte das regras de negócio e, portanto, da semântica das classes comparadas, sendo importante sua inclusão no processo de comparação.

I.3. ESTRUTURA DA TESE

O capítulo II apresenta a referência bibliográfica desta tese, introduz os conceitos principais das áreas em que este trabalho pode ser aplicado e aborda algumas técnicas de comparação existentes.

O capítulo III aborda os conceitos básicos da lógica difusa que foram utilizados para auxiliar a identificação e classificação de classes similares durante o processo de comparação proposto.

No capítulo IV, é mostrado o trabalho proposto nesta tese, com a descrição da técnica de comparação e o seu funcionamento.

O capítulo V descreve a arquitetura de implementação construída com o objetivo de permitir a extensão da técnica de comparação através da incorporação de novas técnicas. Este capítulo descreve também um protótipo da técnica de comparação implementado em um banco de dados orientado a objetos.

No capítulo VI, são apresentados alguns exemplos de aplicação da estratégia de comparação proposta e analisados os resultados obtidos.

Finalmente, o capítulo VII relata as conclusões finais do trabalho e indica extensões e realizações futuras.

II. Métodos de Comparação

II.1. Introdução

Este capítulo apresenta algumas soluções propostas na literatura para o problema de identificação de componentes similares. O objetivo é mostrar diversos tipos de soluções para o problema de comparação de componentes, de maneira que as várias estratégias abordadas exemplifiquem os métodos de comparação que são essenciais para a identificação de conflitos e equivalências. Estes métodos são típicos de problemas associados a integração de esquemas e reuso de componentes, e algumas das técnicas apresentadas por eles são utilizadas na comparação proposta nesta tese.

II.2. Métodos de Integração de Visões Tradicionais

Esta Seção apresenta alguns métodos de integração de visões tradicionais, ou seja, métodos aplicados aos modelos conceituais (como o modelo Entidade e Relacionamentos ou suas extensões). Em cada técnica, são destacados os aspectos relacionados diretamente à comparação entre os elementos dos modelos, em detrimento às técnicas de resolução dos conflitos identificados e a integração efetiva dos esquemas.

Antes de iniciar o estudo de algumas estratégias, é importante analisar o processo de comparação de esquemas do ponto de vista do número de componentes comparados.

Todo processo de integração envolve dois ou mais esquemas, existindo então duas estratégias possíveis para a execução da integração. A primeira, integração binária, permite a integração de dois esquemas de cada vez e pode ser realizada de duas maneiras diferentes: cada esquema é integrado com um esquema intermediário resultante dos passos anteriores (chamado de *ladder strategies* por BATINI *et al.* (1986), ou estratégia acumulativa), ou os esquemas são divididos em pares e integrados simetricamente de uma forma balanceada.

A segunda estratégia, permite a integração de n esquemas de cada vez ($n > 2$). A estratégia n-ária é considerada direta, se todos os esquemas são integrados em um único passo. Caso contrário é chamada de interativa.

A vantagem das estratégias binárias é a diminuição da complexidade das atividades de comparação e conformação dos esquemas a cada passo de integração. A complexidade aumenta proporcionalmente ao número de esquemas envolvidos. Em geral, a integração de n esquemas é da ordem de n^2 (BATINI *et al.*, 1986) e portanto é desejável manter n baixo. As desvantagens da integração binária são o aumento do número de operações para integração (maior número de passos) e a necessidade de uma análise final para adicionar propriedades globais.

As vantagens das estratégias n-árias são a grande análise semântica antes de realizar a integração (evitando assim a necessidade de ajuste e transformações no esquema global), e a diminuição do número de passos para a integração (NAVATHE *et al.*, 1984). Porém, devido à redução da complexidade, a estratégia binária foi adotada por quase todas as metodologias desenvolvidas.

II.2.1. Batini e Lenzerini (1984)

Em um dos primeiros trabalhos sobre integração de esquemas, BATINI e LENZERINI (1984) escreveram "A Methodology for Data Schema Integration in the Entity Relationship Model". Neste artigo, eles apresentam uma metodologia para a integração de visões que utiliza a transformação de esquemas para resolver os conflitos detectados. Por ser uma metodologia de integração de visões, um modelo de dados conceitual foi escolhido pois permite representar melhor a semântica do universo de discurso.

Assim, o modelo de dados adotado foi uma extensão do Modelo Entidade e Relacionamentos proposto originalmente por CHEN (1976). O meta-esquema do modelo é apresentado na Figura II.1. Entre as modificações em relação ao modelo original, aparecem propriedades úteis durante a integração, como, por exemplo, um conjunto de sinônimos para cada conceito. O artigo possui uma descrição mais detalhada sobre o modelo.

O objetivo principal da metodologia é estabelecer um procedimento a ser seguido pelo projetista ou integrador durante a integração de visões. Neste sentido, foram definidas algumas estratégias e etapas que caracterizam o processo de integração. Quanto ao número de esquemas integrados em cada etapa do processo, a estratégia adotada foi a integração binária e acumulativa, ou seja, a cada passo do

processo de integração, dois esquemas são integrados, sendo um deles um esquema parcial obtido no passo anterior (veja a Figura II.1).

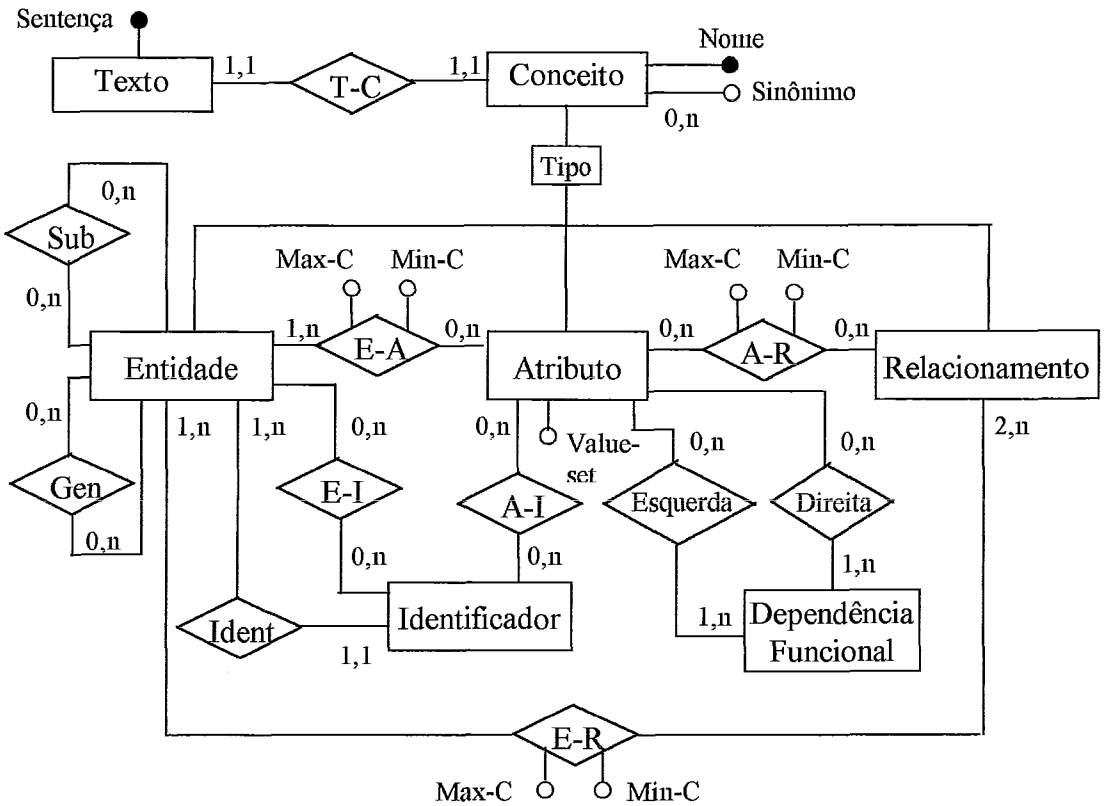


Figura II.1 - Modelo utilizado por BATINI e LENZERINI (1984)

BATINI e LENZERINI dividiram o processo de integração em três etapas: análise de conflitos, junção e, por último, restruturação e enriquecimento final.

A análise de conflitos pode ser considerada a etapa principal do processo de integração e corresponde a comparação de dois esquemas com o objetivo de identificar as diferenças entre eles. Dois tipos de comparações são identificadas:

- Comparação de nome: comparação e análise dos nomes de cada conceito dos dois esquemas, identificando sinônimos e homônimos.
- Comparação de estrutura de esquema: durante esta tarefa, as representações escolhidas nos dois esquemas para um mesmo conceito são comparadas.

Para evitar uma explosão no número de comparações a serem realizadas, podem ser usadas algumas heurísticas que verificam um conjunto de propriedades e restrições dos conceitos e de seus vizinhos. Ao comparar conceitos com um mesmo tipo, são verificados o domínio, se os conceitos forem atributos. No caso da

comparação envolver entidades, são verificados atributos e relacionamentos, e, finalmente, para a comparação de relacionamentos, são considerados os seus atributos e as entidades envolvidas no relacionamento. Para conceitos com tipos diferentes são usadas outras heurísticas. Na comparação de uma entidade com um atributo, são analisados o domínio do atributo e o domínio dos identificadores das entidades. Para comparar uma entidade com um relacionamento, são considerados os atributos do relacionamento e o domínio dos identificadores.

Após a comparação dos dois esquemas e a identificação dos ponto conflitantes, tem início a etapa de análise dos conflitos. Durante esta etapa, para resolver os conflitos de nome, a transformação de alterar o nome de um dos conceitos é adotada. Em seguida, ocorre a análise dos conflitos de estrutura de esquemas que pode ser subdividida em duas partes.

Durante a primeira parte, são analisados os conceitos com o mesmo nome e tipos diferentes. Para resolver esta inconsistência é realizada uma restruturação sintática dos conceitos, aplicando uma ou mais transformações nos esquemas. BATINI e LENZERINI (1984) definiram cinco transformações atômicas, ou seja, transformações que têm como entrada conceitos atômicos (entidade, relacionamento ou atributo).

Na segunda parte, denominada análise de compatibilidade, todos os conceitos que correspondem a uma mesma classe do universo de discurso possuem o mesmo nome e o mesmo tipo. Nela é analisada a compatibilidade das representações de um mesmo conceito nos dois esquemas. Segundo BATINI e LENZERINI (1984), uma definição formal de compatibilidade depende do modelo conceitual utilizado. Neste artigo, quatro incompatibilidades foram identificadas para o modelo adotado:

1. Cardinalidades diferentes para um mesmo atributo ou entidade (participante de um relacionamento),
2. Um identificador em um esquema não é identificador em outro esquema,
3. Uma entidade é um subconjunto de outra entidade em um esquema e o contrário acontece no outro, e
4. Dependências funcionais diferentes são definidas para os mesmos atributos de uma entidade em diferentes esquemas.

Existem duas soluções possíveis para estas incompatibilidades. Na primeira, uma representação é escolhida em detrimento de outra. Na segunda, uma

representação comum é construída de tal maneira que as restrições existentes nos dois esquemas estejam presentes. O integrador é responsável pela escolha de uma ou outra solução.

A etapa de junção dos esquemas corresponde à etapa mais simples do processo e ocorre com a simples sobreposição dos dois esquemas, já que os conflitos foram solucionados na etapa anterior.

A última etapa de restruturação e enriquecimento final possui três tarefas principais: análise de propriedades entre esquemas, análise de caminhos redundantes e a restruturação do esquema global. A análise de propriedades entre esquemas permite acrescentar, por exemplo, relacionamentos entre entidades que estavam em esquemas diferentes. A análise de caminhos redundantes corresponde à eliminação de ciclos formados por relacionamentos entre entidades. A restruturação do esquema global procura melhorar a qualidade do esquema conceitual final, melhorando a sua expressividade, clareza e simplicidade, utilizando algoritmos de normalização, por exemplo.

Concluindo, esta metodologia apresentou um conjunto de técnicas e heurísticas para a comparação de esquemas, um conjunto de transformações que resolvem os conflitos encontrados durante a comparação, além de definir um procedimento que serve como roteiro para o integrador. BATINI e LENZERINI (1984) propuseram uma "maneira de pensar" ao invés de um conjunto de regras para a integração. Esta metodologia não apresenta uma ferramenta automatizada para ajudar na integração, mas ela propõe a utilização de ferramentas, como glossários e diagramas, para auxiliar o processo de integração que depende exclusivamente da interação com o integrador. Por ser uma metodologia de integração de visões, vários conflitos típicos de integração de banco de dados, como por exemplo conflitos de escala e precisão, não foram considerados. O modelo adotado também não possui a semântica de comportamento associada aos conceitos, portanto esses conflitos também foram desconsiderados.

II.2.2. Souza (1986a)

O trabalho "Software Tools For Conceptual Schema Integration" apresentado por SOUZA (1986a) constitui uma das pesquisas mais completas realizadas sobre integração de esquemas da época. No método proposto para a integração de visões, algumas etapas do processo de integração são automatizadas por uma ferramenta e a

comparação dos esquemas identificou conflitos que são resolvidos através de mapeamentos.

Segundo SOUZA (1986a), para integrar esquemas em um ambiente heterogêneo é necessário criar mapeamentos entre eles ou expressá-los em um modelo comum, denominado modelo canônico. O objetivo principal por trás de uma representação em modelo canônico é minimizar o número de possíveis representações de um conceito. Neste sentido, foi adotado o modelo conceitual ACS (Abstract Conceptual Schema) (STOCKER e CATIE, 1983) como modelo comum a todos esquemas. Portanto, assim como BATINI e LENZERINI (1984), este trabalho adotou um modelo semântico como modelo para os esquemas a serem integrados. A Tabela II.1 mostra as estruturas principais existentes no modelo ACS.

Tabela II.1 – Estruturas do modelo ACS

Estrutura	Descrição
DataItem	Item de dado unitário e indivisível que possui um nome e um tipo básico. Não aceita a definição de restrições de valores ou domínios.
Function	Retorna um valor de um DataItem quando aplicada a um Element Set e a um DataItem.
Property	Uma função que, quando aplicada a um Element Set, retorna o valor de um DataItem com o mesmo nome.
Identifier	Retorna o elemento cujo(s) DataItem(s) definido(s) como identificador(es) possui(em) um(ns) determinado(s) valor(es).
Element Set	Conjunto de elementos obtidos pela aplicação sucessiva de um Identifier para todos os valores do domínio no qual o Identifier está definido.
Value Set	Conjunto de n-árias tuplas obtidas com a aplicação das Property(ies) participantes de sua definição sobre cada elemento em um Element Set.
Derived Value Set	Conjunto de n-árias tuplas obtidas com a aplicação de uma operação (agregação ou operador da lógica relacional, por exemplo) sobre um Value Set.
Value Set Constraint	Usado para expressar as restrições de integridade. É uma expressão contendo o nome de dois Value Sets e um operador de restrição (por exemplo, IS-EQUAL-TO) que deve ser sempre verdadeira.
Location	Utilizado para representar um relacionamento IS-A, criando hierarquias de generalização.

SOUZA (1986a) utiliza operadores para realizar o mapeamento entre dois esquemas comparados, como os operadores da álgebra relacional (PROJECT,

UNION, etc.), ou novos operadores (criados para mapear conflitos nos quais um conceito em um esquema aparece como metadados em outro). Os mapeamentos ocorrem após a etapa de comparação dos esquemas.

SOUZA (1986a) destacou a importância de uma ferramenta automatizada para o processo de integração de esquemas. A ferramenta apresentada foi o SIS (Schema Integration System) (SOUZA, 1986b) implementada em PROLOG.

SOUZA (1986a) também analisou cinco estratégias para a comparação de esquemas: comparação sintática, reconhecimento de padrão, teoria da informação aplicada à integração, prova de teorema e comparação de grafos. O algoritmo implementado no SIS contém várias características dessas estratégias e é descrito com mais detalhes abaixo.

O método implementado procura identificar semelhança entre duas classes utilizando uma agregação difusa (*fuzzy*) dos diversos tipos de comparações possíveis. A semelhança entre dois Element Sets, por exemplo, pode ser avaliada sob diversos aspectos como "terem o mesmo nome" ou "terem identificadores com o mesmo nome", etc.. Cada aspecto possui uma função de equivalência F_i, $F: A \times B \in [0, 1]$, onde A e B são os domínios em cada um dos esquemas. Assim, uma função de semelhança de nomes assume o valor 1 quando os nomes forem idênticos, 0 quando são inteiramente diferentes e um valor intermediário (entre 0 e 1) quando possuem alguma diferença não essencial (são relacionados por um dicionário de sinônimos, por exemplo). Os valores intermediários representam o grau de equivalência entre os dois domínios e a função F pode ser considerada uma função de pertinência para um subconjunto difuso A x B.

A equivalência entre duas classes pode ser analisada então sob diversos aspectos, cada qual com uma função de equivalência própria. Para obter o resultado final, as funções correspondentes aos vários aspectos são agregadas. Por sua vez, as funções podem ser definidas por outras funções de equivalências de outros aspectos, formando uma hierarquia de funções. Por exemplo, a semelhança entre um par de DataItems pode ser usada para estimar a semelhança de Properties que, por sua vez, podem ser usadas para estimar a semelhança entre Element Sets.

Existem diversas maneiras de agregar os valores das funções de equivalência. O método adotado por SOUZA (1986a) usa a Equação 2.1 para calcular a função F_x

aplicada a um par de objetos ACS comparáveis, *a* e *b*, usando as funções de equivalência F_q e os pesos associados W_q, $q = 1...m$.

$$F_x = \sum_{q=1}^{m} F_q(a,b)*W_q \qquad 2.1$$

Os pesos podem ser usados para ajustar o algoritmo conforme desejado, e são normalizados para somarem 1. Assim, para destacar a comparação dos nomes entre dois Element Sets em detrimento aos identificadores, podemos aumentar o peso associado a função de equivalência dos nomes e diminuir a participação dos identificadores no resultado final. A Figura II.2 mostra a hierarquia e os valores padrões dos pesos associados a cada função para a comparação de dois Element Sets.

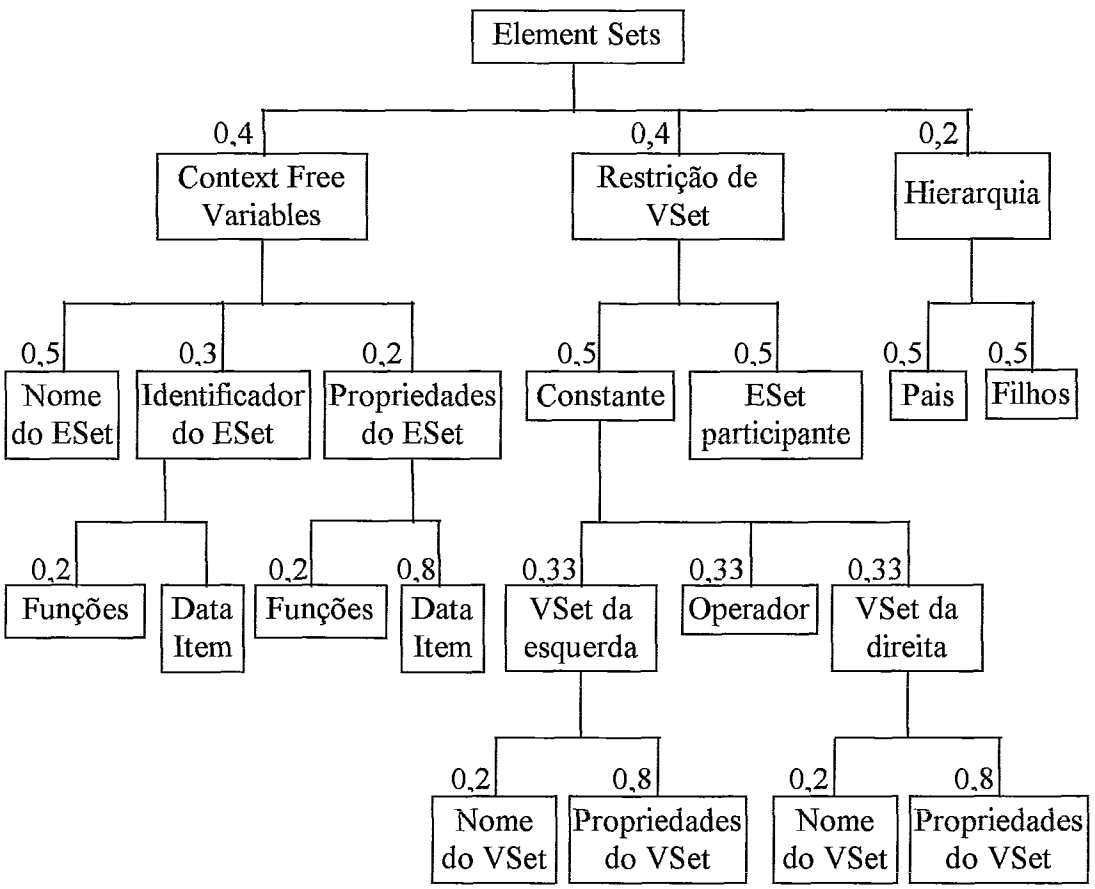


Figura II.2 - Funções de equivalência e pesos para a comparação de Element Sets

SOUZA (1986a) apresentou uma ferramenta automatizada para auxiliar o processo de integração, destacando o processo de comparação de esquemas. O algoritmo de comparação adotado, utilizando agregação difusa de funções de

equivalência, permite comparar diversos aspectos de cada conceito. Além disso, ele torna o método de comparação flexível, permitindo o acréscimo de novas formas e estratégias de comparação, bastando para isso um ajuste nos pesos de agregação.

II.2.3. Larson, Navathe e Elmasri (1989)

Neste artigo, os autores agrupam os seus trabalhos anteriores (NAVATHE, *et al.*, 1984 e NAVATHE *et al.*, 1986) e apresentam uma extensão para os métodos de integração propostos, utilizando a equivalência entre atributos para integrar entidades, relacionamentos e entidades com relacionamentos.

Assim como SOUZA (1986a) e NAVATHE *et al.* (1984), os autores utilizam um modelo conceitual como modelo canônico a ser usado na integração. O modelo adotado foi uma extensão do Modelo Entidade e Relacionamentos proposto originalmente por CHEN (1976), denominada Entidade, Categoria e Relacionamentos (ECR). A necessidade de representar subclasses motivou a criação deste modelo que possui uma definição formal em ELMASRI *et al.* (1985). Um exemplo de um diagrama do modelo ECR aparece na Figura II.3.

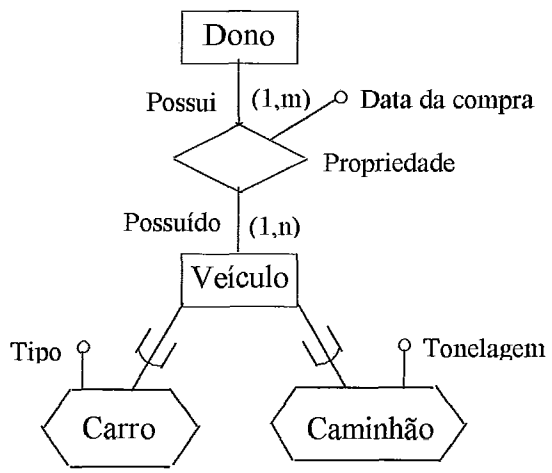


Figura II.3 – Exemplo de um diagrama ECR

A integração das entidades e categorias (representadas por hexágonos na figura), também denominadas classes, e dos relacionamentos está baseada na relação de equivalência dos seus atributos. Para determinar a equivalência entre atributos, oito características são analisadas:

1. Unicidade: restrição que garante que dois objetos pertencentes a uma mesma classe não podem ter os mesmos valores para um conjunto de atributos que identificam unicamente uma instância da classe.
2. Cardinalidade: número de valores para um atributo que uma instância da classe pode ter. Existem limites inferiores e superiores.
3. Domínio: um conjunto de valores que um atributo de uma classe pode assumir.
4. Restrições estáticas e semânticas de integridade: restrições de integridade definidas para um atributo, como, por exemplo, dependências funcionais ou integridade referencial.
5. Restrições dinâmicas e semânticas de integridade: esta característica descreve restrições sobre as atribuições de valores que um atributo deve seguir.
6. Restrições de segurança: restrições de uso (visibilidade pelos usuários) dos valores do atributo.
7. Operações permitidas: coleção de operações permitidas associadas ao domínio de cada atributo.
8. Escala: especifica a interpretação para os valores dos atributos. Por exemplo, para os atributos numéricos, a escala representa a unidade de medida.

Os autores definem então três tipos de equivalência entre atributos: forte, fraca e disjunta. A diferença entre os tipos de equivalência está relacionada com a possibilidade de determinar uma função que possua sete propriedades, chamadas de propriedades básicas de equivalência de atributos. A definição completa da função e as propriedades básicas são apresentadas na Tabela II.2.

Tabela II.2 – Função de equivalência e suas propriedades

Propriedades básicas de equivalência: Seja a_i um atributo de uma classe A e b_i um atributo de uma classe B. Seja D_i o maior subconjunto não nulo do domínio de a_i - DOM(a_i) - e R_i o maior subconjunto não nulo do DOM(b_i), tal que existe uma função de mapeamento f_i: $D_i \rightarrow Ri$ e sua inversa fi^*: $R_i \rightarrow D_i$. As propriedades dessa função são:
1. A função f_i é isomórfica, isto é, existe uma correspondência um-para-um entre D_i e R_i. 2. Cada operação permitida sobre a_i tem uma operação equivalente sobre b_i.

3. Todas as restrições semânticas são asseguradas pelas funções de mapeamento f_i e f_i^*.
4. Todas as transições de estados são asseguradas pelas funções f_i e f_i^*.
5. Todas as restrições de segurança são válidas para f_i e f_i^*.
6. As funções de mapeamento preservam as dependências funcionais.
7. As funções de mapeamento preservam os identificadores únicos das classes.

O primeiro tipo de equivalência entre atributos é a equivalência forte. A equivalência entre dois atributos (*a* e *b*) é considerada forte se existe uma função de mapeamento que obedeça todas as propriedades definidas acima. Este tipo de equivalência possui quatro subtipos que são determinados após a comparação dos domínios dos atributos: igual, contém, está contido e sobreposto. A equivalência forte entre os atributos implica na possibilidade de atualização do atributo integrado *c*, resultado da integração de *a* e *b*. A compatibilidade total existe se os atributos possuírem uma equivalência "forte igual" e, em alguns casos, equivalências dos tipos contém e está contido. A equivalência forte sobreposta não possibilita uma compatibilidade total entre os atributos *a* e *b*.

Os atributos também podem estar relacionados através da equivalência fraca. Este tipo de equivalência também deve seguir as propriedade básicas de equivalência, com exceção das propriedades 3, 4 e 5, porém não necessita de uma função inversa de mapeamento dos domínios dos atributos. Os mesmos subtipos existentes para a equivalência forte também são definidos para a equivalência fraca. A principal diferença entre as equivalências forte e fraca está na incompatibilidade de atualização dos atributos na equivalência fraca, devido a ausência de uma função inversa no mapeamento dos domínios dos atributos.

O último tipo de equivalência é a disjunta. Essa equivalência é definida quando dois atributos não possuem equivalência forte nem fraca e os papéis dos atributos são iguais (representam o mesmo conceito do universo de discurso).

O processo de integração utiliza as equivalências definidas acima para integrar classes (entidades e categorias) e relacionamentos. Este processo é o mesmo para equivalências fortes e fracas, sendo realmente determinante o subtipo da equivalência (igual, contém, está contido e sobreposta). A abordagem escolhida para determinar a

equivalência entre duas classes é através da equivalência dos valores dos atributos identificadores (chaves) das classes.

Assim, existem cinco tipos de equivalência entre classes: igual, contém, está contido, disjunta e sobreposta. Duas classes são ditas iguais se existe uma equivalência do tipo igual entre os atributos que formam suas chaves. As equivalências dos tipos contém, está contido, disjunta e sobreposta são obtidas de maneira análoga.

Os mesmos tipos de equivalência ocorrem para os relacionamentos, observando-se apenas que os atributos analisados são as chaves das classes participantes do relacionamento.

Os autores definiram então um conceito único (a equivalência entre atributos) que pode ser utilizado durante todo o processo de integração. Deste modo eles conseguiram projetar e implementar uma ferramenta capaz de auxiliar ao administrador na tarefa de integração. A equivalência entre atributos também é utilizada em outros trabalhos sobre integração de esquemas (DAI, 1997).

II.3. INTEGRAÇÃO DE ESQUEMAS OO

Esta Seção apresenta algumas técnicas de comparação criadas para a integração de esquemas orientados a objetos. Estas técnicas diferem das técnicas de integração de visões tradicionais apresentadas na Seção anterior pois incorporam ao processo de comparação o comportamento das classes dos diferentes esquemas.

II.3.1. Thieme e Siebes (1993)

Neste trabalho, THIEMES e SIEBES (1993) apresentam um método formal para comparar esquemas orientados a objetos, em especial, hierarquia de classes. O método apresentado utiliza uma função sintática de ordem de subclasse para comparar classes e um operador de *join* para integrar dois esquemas.

O modelo orientado a objetos adotado, expresso na forma de hierarquia de classes, é semelhante ao proposto por LÉCLUSE e RICHARD (1989). A hierarquia de classes é obtida a partir da gramática apresentada na Tabela II.3. Antes, são definidos cinco conjuntos disjuntos: um conjunto denominado CN, cujos elementos são os nomes de classes existentes, um conjunto AN com o nome dos atributos, um conjunto MN com o nome dos métodos, um conjunto LABELS com o nome de variáveis e parâmetros e, por último, um conjunto CONS com as constantes básicas (constantes

de inteiro, cadeia de caracter ou números reais, por exemplo). Observe que os métodos definidos pela gramática são exclusivamente métodos de atualização, isto é, métodos que atribuem valores aos atributos de uma classe.

Tabela II.3 – Gramática da Hierarquia de classes, por THIEMES E SIEBES

Hierarchy	::=	Class +
Class	::=	'Class' CN ['Isa' CN +] ['Attributes' Att +] ['Methods' Meth +] 'Endclass'
Att	::=	AN ':' Type
Type	::=	BasType | SetType | RecType | CN
BasType	::=	'integer' | 'rational' | 'string'
SetType	::=	'{' Type '}'
RecType	::=	'<' FieldList '>'
FieldList	::=	Field | Field ',' FieldList
Field	::=	LABELS ':' Type
Meth	::=	MN ['(' ParList ')'] '=' AsnList | MN '\' MN ['(' ParList ')'] '=' AsnList
ParList	::=	Par | Par ',' ParList
Par	::=	LABELS ':' BasType
AsnList	::=	Assign | Assign ';' AsnList
Assign	::=	Dest ':=' Source | 'insert(' Source ',' Dest ')'
Dest	::=	AN | Dest '.' LABELS
Source	::=	Term | Term '+' Source | Term '-' Source | Term '×' Source | Term '÷' Source
Term	::=	Var | CONS
Var	::=	L | NA | Var '.' LABELS | Var '.' AN

A comparação sintática e semântica de classes utiliza uma função sintática. Esta função é definida em termos de uma relação fraca de subtipo para os atributos e uma relação fraca de subfunção sobre o funcionamento dos métodos. A comparação de classes foi dividida em duas etapas a saber: comparação de atributos e comparação de métodos.

A relação fraca de subtipo é definida em termos do isomorfismo de grafos entre as árvores que representam as classes e seus atributos. A Figura II.4 apresenta a definição da representação de uma classe sob a forma de uma árvore e um exemplo.

Definição: Seja H uma hierarquia de classes bem definida e C uma classe em H. Seja c nome(C) e $\{a_1 : T_1, ..., a_k : T_k\}$ atributos de C. A árvore representando a classe C, denominada struc(C) é definida assim:

struct(d) = struct (D) se $\exists\, D \in H$ [nome(D) = d],

struct(B) tem um e somente um nó, com o rótulo B, se $B \in$ {integer, rational, string},

struct($\{U\}$) consiste de uma raiz com rótulo {}, uma subárvore struct(U) e uma aresta não rotulada da raiz {} para a raiz de struct(U),

struct($<l_1 : U_1, ..., l_n : U_n>$) consiste de uma raiz com rótulo <>, subárvores struct(U_1), ..., struct(U_n), e arestas rotuladas por l_i, uma para cada $i \in \{1, ..., n\}$, da raiz <> para a raiz de struct(U_i).

Exemplo:

Class Pessoa
Attributes
nome : string
dt_nasc : Date
mae : Pessoa
Methods
change (s : string) =
name := s
Endclass

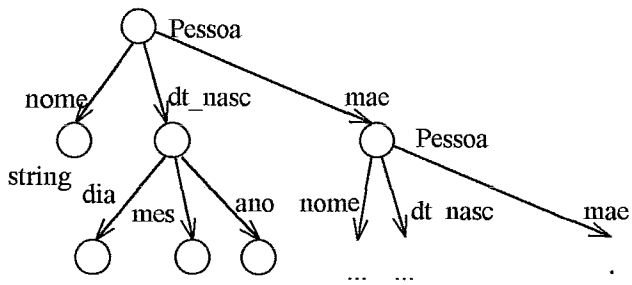


Figura II.4 – Árvore de representação de uma classe

Para uma classe ser um subtipo de outra, deve existir uma função de mapeamento de grafos φ que seja injetiva em relação ao homorfismo de árvores, ou seja: preserve os nomes dos nós com exceção de nomes de classes e atributos e seja exata em relação às classes, isto é, relacione uma classe somente para outra classe. Essas funções de mapeamento são utilizadas então para integrar classes.

A comparação e integração de métodos de duas classes distintas está baseada na igualdade dos comandos, os quais, no modelo adotado, estão limitados às atribuições de valores. Esta igualdade é verificada com o auxílio da função de mapeamento entre atributos de classes φ, vista acima. A definição de especialização de um método é mostrada abaixo:

Definição: Seja spec($m(P) = E$, φ) a condição que, para cada atribuição $d := s$ em E, a existência do destino $\varphi^{-1}(d)$ em C implica na existência da fonte $\varphi^{-1}(s)$ em C, tal que o tipo de $\varphi^{-1}(s)$ é um subtipo do tipo de $\varphi^{-1}(d)$. A generalização de $m(P) = E$ em C denominada por $\varphi^{-1}(m(P) = E)$ é definida como $m'(P') = E'$, onde

1. E' é obtido de E substituindo todas as atribuições $d := s$ em E, tal que o destino $\varphi^{-1}(d)$ existe, por $\varphi^{-1}(d) := \varphi^{-1}(s)$ e removendo todas as outras atribuições.
2. P' é obtida de P removendo os parâmetros que não parecem em E'.

Portanto, para dois métodos serem equivalentes, o primeiro deve ser uma generalização do segundo, ou seja, todas as atribuições realizadas no segundo método devem envolver as mesmas ou mais atribuições do primeiro. Além disso, os atributos e valores envolvidos nas atribuições equivalentes devem ser do mesmo tipo utilizado no primeiro método ou uma especialização desse tipo.

Com a comparação de atributos e métodos definidas formalmente, THIEMES e SIEBES (1993) propuseram um operador de *join* para uma hierarquia de classes. Esse operador é usado para integrar duas ou mais classes, baseando-se nas funções de comparação de métodos e atributos. A integração ocorre então, através de uma transformação dos esquemas originais para um novo esquema global. A Figura II.5 corresponde a um exemplo de integração de classes.

Esquema 1

```
Class Quadrado
  Attributes
    x_left_up :integer
    y_left_up :integer
    width :integer
  Methods
    set (x :integer, y :integer) =
      x_left_up := x; y_left_up := y
    translate (delta_x :integer, delta_y :integer) =
      x_left_up := x_left_up + delta_x;
      y_left_up := y_left_up + delta_y
Endclass

Class SFigure
  Attributes
    nome :string
    primeiro :Quadrado
    segundo :Quadrado
  Methods
    copia1 = primeiro := segundo
    copia2 = segundo := primeiro
Endclass
```

Esquema 2

```
Class Retangulo
  Attributes
    x_left_up :integer
    y_left_up :integer
    width_x :integer
    width_y :integer
  Methods
    set (x :integer, y :integer) =
      x_left_up := x; y_left_up := y
    translate (delta_x :integer, delta_y :integer) =
      x_left_up := x_left_up + delta_x;
      y_left_up := y_left_up + delta_y
    rotate =
      y_left_up := y_left_up + width_x;
      width_x := width_y - width_x;
      width_y := width_y - width_x;
      width_x := width_x + width_y
Endclass

Class RFigure
  Attributes
    code :integer
    esquerda :Retangulo
    direita  :Retangulo
  Methods
    copia_d = direita := esquerda
    copia_e = esquerda := direita
Endclass
```

Esquema integrado

```
Class Retangulo Isa Quadrado
  Attributes
    width_y :integer
  Methods
    rotate =
      y_left_up := y_left_up + width;
      width   := width_y - width;
      width_y := width_y - width;
      width   := width + width_y
Endclass

Class Figura
  Attributes
    primeiro :Quadrado
    segundo  :Quadrado
  Methods
    copia1 = primeiro := segundo
    copia2 = segundo  := primeiro
Endclass

Class SFigura Isa Figura
  Attributes
    nome :string
Endclass

Class Rfigura Isa Figura
  Attributes
    codigo :integer
    primeiro :Retangulo
    segundo  :Retangulo
Endclass
```

Figura II.5 – Exemplo de integração em THIEMES e SIEBES (1993)

Portanto, THIEMES e SIEBES (1993) definiram formalmente um método de integração de classes que considera as características sintáticas de seus atributos e os seus métodos de atualização, abrangendo portanto a forma de manipulação dos objetos. O maior destaque deste método é considerar o comportamento das classes durante o processo de integração.

II.3.2. Damasceno, Ribeiro e Oliveira (1997)

Este trabalho foi apresentado no XII Simpósio Brasileiro de Banco de Dados e mostra como foi realizada a integração de esquemas para possibilitar a interoperabilidade de bancos de dados heterogêneos, auxiliando na elaboração de mediadores (WIEDERHOLD, 1996). A integração é implementada através de uma interface CORBA do padrão OMG. O modelo adotado foi um modelo conceitual orientado a objetos e o processo de integração considera também, além das informações tradicionais de cada classe, o comportamento de seus objetos.

A integração é obtida através do mapeamento entre os esquemas conceituais. Este mapeamento é detalhado por RIBEIRO e OLIVEIRA (1995) e uma taxionomia de conflitos também foi definida em RIBEIRO e OLIVEIRA (1994). A Figura II.6 mostra a arquitetura CORBA e seus quatro componentes principais: ORB (*Object Request Broker*), Serviços de Objetos, Recursos Aplicativos e Objetos de Aplicação. O modelo orientado a objetos utilizado possui também o conceito de papéis (*roles*).

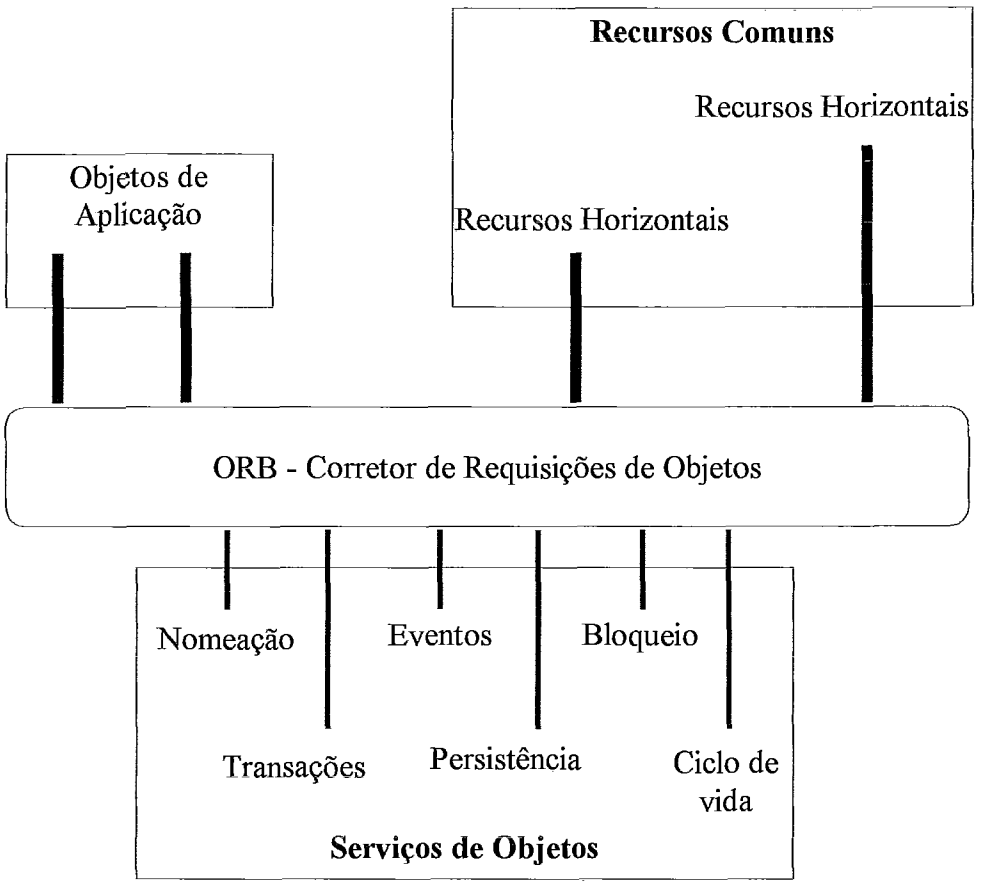


Figura II.6 – Arquitetura de gerenciamento de objetos do OMG

O processo de mapeamento dos esquemas conceituais utiliza quatro tabelas criadas a partir de um esquema de exportação. O esquema de exportação é definido em cima do esquema conceitual local e contém o conjunto de representações correspondentes aos objetos que serão disponibilizados. Os esquemas de exportação são a base para a construção das Tabelas de Objetos Locais (TOL), onde as informações relevantes (nome e estrutura) sobre as classes são armazenadas, acrescidas de uma descrição sucinta sobre o objeto especificado.

Os resultados da comparação dos esquemas conceituais (esquemas de exportação e tabelas de objetos locais) dão origem a duas outras tabelas: Tabela de Equivalência de Objetos (TEO) e a Tabela de Equivalência de Atributos (TEA). A TEO indica o relacionamento entre pares de objetos equivalentes e, para cada entrada, é criada uma TEA, onde os relacionamentos identificados entre os atributos modelados em cada representação são descritos. Um esquema dessa estrutura de tabelas aparece na Figura II.7. O mapeamento propriamente dito é efetuado a partir das informações registradas nestas tabelas de equivalência.

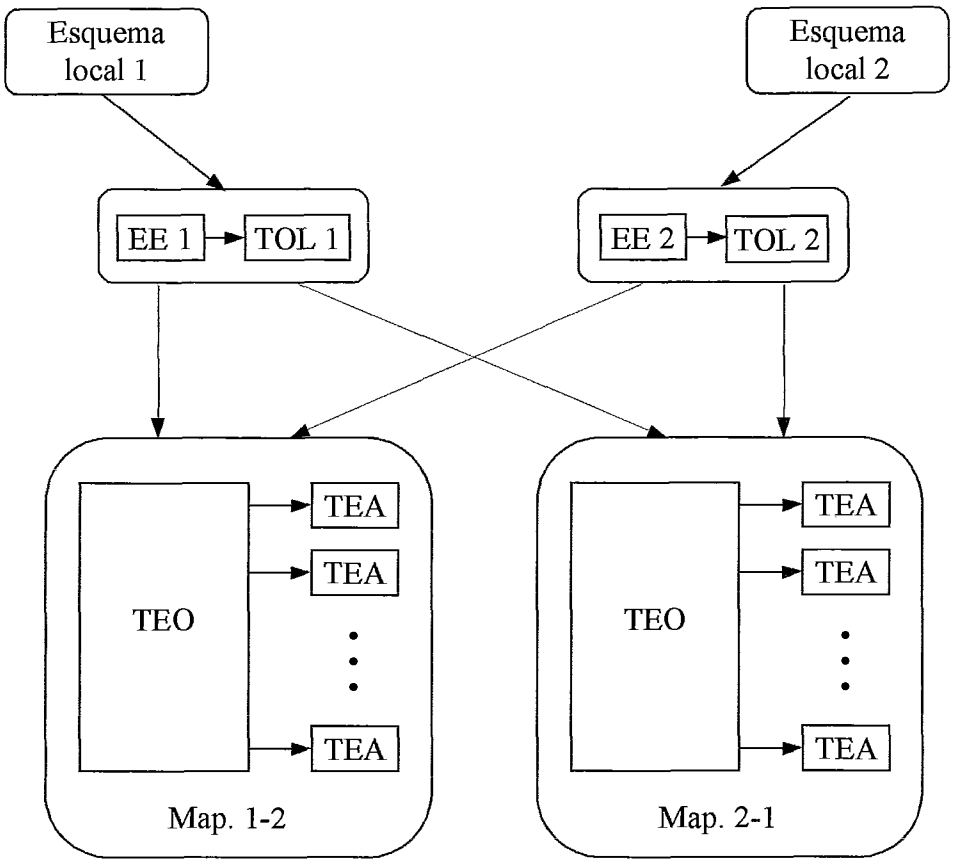


Figura II.7 – Tabelas para o mapeamento de esquemas

A metodologia de mapeamento possui quatro etapas: identificação de equivalências, identificação de conflitos, comparação de comportamento e a definição do mapeamento. As três primeiras etapas efetuam a comparação entre os esquemas, enquanto a última etapa corresponde a integração entre os esquemas. A etapa de identificação de equivalências tem como objetivo descobrir as diferentes representações de um mesmo objeto do mundo real. Os resultados desse processo são registrados na TEO e nesta etapa a TEA começa a ser montada. Esta etapa é composta por quatro passos:

1. Identificação de mesmos nomes de objetos: os nomes associados aos principais elementos estáticos da modelagem são comparados para a identificação de equivalências. A comparação ocorre entre a TOL local e a TOL externa, cadastrando na TEO os objetos equivalentes confirmados pelo usuário.
2. Identificação de mesmo nomes de atributos de objetos: este processo restringe-se à comparação de atributos chaves (do tipo *unique*), permitindo a identificação de objetos equivalentes através de seus atributos identificadores.

3. Identificação dos mesmos componentes nas estruturas: verifica a existência de classes que contenham papéis (*roles*) que já estão na TEO, ou de papéis contidos em classes, ou de papéis irmãos já incluídos na TEO, mas que não tenham sido identificados como equivalentes.
4. Identificação de atributos equivalentes: todo o par de objetos equivalentes relacionados na TEO gera uma TEA, onde seus atributos são cruzados para a identificação de equivalências. Assim, são comparados os nomes dos atributos e os seus domínios.

Após a identificação das ocorrências de sinônimos e homônimos na etapa anterior, a comparação dos esquemas tem continuidade, com a próxima etapa de identificação de conflitos sendo dividida em cinco passos:

1. Identificação de divergências de conjuntos de objetos sendo modelados: a equivalência entre uma classe e um papel, ou a existência de papéis não equivalentes em classes identificadas como tal, podem indicar que existe divergência entre os elementos modelados, caracterizando um conflito de restrição de modelagem. Deve-se incluir uma relação de equivalência "contido" ou "contém" entre os dois objetos na TEO.
2. Identificação de mesmos objetos modelados com diferentes perspectivas: ocorre quando existe um par de objetos na TEO com um número mínimo de atributos equivalentes, refletindo a modelagem complementar nos diferentes esquemas conceituais. A identificação deste tipo de conflito ocorre em paralelo com a identificação de equivalência de atributos.
3. Identificação de mesmos objetos modelados com diferentes construtores: este conflito resulta do uso de diferentes estruturas na representação de objetos equivalentes. A detecção deste conflito é automática, através da comparação da estrutura existente na TOL, referente a cada par de objetos relacionados na TEO.
4. Identificação dos mesmos objetos sendo modelados com diferentes critérios de agregação: no conflito de agregação, as mesmas entidades do mundo real estão agregadas seguindo critérios diferentes. Esta situação é detectada através da ocorrência de um mesmo elemento incluído na TEO repetidas vezes, mas relacionando-se a diferentes representações.

5. Identificação de diferentes domínios para os mesmos atributos: atributos já identificados como equivalentes apresentam divergências quanto ao domínio descrito.

A próxima etapa consiste na comparação do comportamento dos objetos, identificando conflitos através da comparação de mensagens, regras de transição de estados e regras de integridade modeladas. A comparação efetuada ocorre em termos sintáticos e possui os seguintes passos:

1. Identificação de diferenças em mensagens de objetos equivalentes: este conflito auxilia na descoberta de outros conflitos não identificados relativos à parcela do universo de discurso sendo modelada. As mensagens dos objetos são comparadas quanto: ao número de mensagens definidas, aos nomes estabelecidos para as mensagens, ao número de parâmetros definidos para mensagens equivalentes e aos domínios desses parâmetros.
2. Identificação de diferenças em regras de objetos equivalentes: diferença entre regras de transição de estados e de integridade onde conflitos ainda não foram identificados. O processo de comparação envolve o número de regras de transição de estados e de integridade e a comparação de predicados entre as regras do mesmo tipo.

A última etapa do processo de integração corresponde ao mapeamento entre os esquemas conceituais. O mapeamento torna possível o acesso ao conjunto total de instâncias correspondentes a uma mesma entidade do mundo real distribuídas através das bases de dados da federação e é composto por três passos: identificação de objetos e o mapeamento entre identificadores, mapeamento entre atributos equivalentes e mapeamento de domínio de atributos externos.

Para facilitar o processo de integração, foi desenvolvida uma ferramenta que assiste todo o processo e facilita a interação com o usuário (no caso, o responsável pela integração). Esta ferramenta foi construída utilizando a arquitetura CORBA apresentada na Figura II.6.

Este método utiliza a comparação de comportamento durante a integração, realizando a comparação sintática dos métodos de cada classe ou objeto, assim como THIEME E SIEBES (1993). Porém, ambos não consideram o comportamento observável, isto é, não comparam o resultado da execução dos métodos dos objetos

como proposto na técnica de busca por comportamento que será apresentada na próxima Seção.

II.4. MÉTODOS DE BUSCA E RECUPERAÇÃO DE CÓDIGO

O objetivo principal da busca e recuperação de código é facilitar a reutilização de componentes. Entende-se por componentes, procedimentos, funções ou até mesmo classes que estejam armazenados em bibliotecas para a reutilização. Os conceitos de busca em relação a código, e os métodos tradicionais de busca e recuperação de informações, podem ser estendidos a outros elementos do processo de desenvolvimento de software, como a especificação de requisitos, Modelos Entidade e Relacionamentos, etc..

As técnicas de busca e recuperação de código se desenvolveram ao longo dos últimos anos, estimuladas pelo crescimento do número de componentes reutilizáveis e de bibliotecas que armazenam e distribuem esses componentes. A reutilização tem ganho força graças ao alto custo de desenvolvimento de software (o preço dos equipamentos caem dia após dia, mas o custo para se desenvolver um software permanece praticamente constante). Além disso, a reutilização pode acelerar o processo de desenvolvimento (diminuindo o seu custo) e aumentar a confiabilidade dos sistemas (os componentes reutilizáveis já foram testados e aprovados).

Mas aqueles que pretendem utilizar ao máximo a reutilização de código enfrentam muitas dificuldades, entre elas o problema de buscar e recuperar os componentes que implementam a funcionalidade desejada. Muitas das técnicas existentes para a recuperação de código foram adaptadas da recuperação de textos através de palavras chaves. Esta Seção introduz alguns métodos para ilustrar as características, problemas e soluções envolvidas na busca e recuperação de código. Os primeiros métodos e técnicas apresentados (técnicas tradicionais e reconhecimento de padrões) não consideram o comportamento dos componentes. Os dois últimos métodos testam o comportamento durante a comparação de componentes. Uma revisão mais completa sobre este assunto pode ser encontrada em BARROS (1995). Antes, é interessante definir dois termos que são utilizados para avaliar os métodos de recuperação de documentos (documentos são entendidos como sendo qualquer unidade de informação recuperável que o usuário esteja buscando, como por exemplo arquivos, componentes, funções, etc.).

Dada uma consulta, existem um conjunto de documentos com *n* elementos que realmente satisfazem a consulta (documentos relevantes). Quando a consulta é executada, são recuperados vários documentos que podem se divididos em dois grupos: os que realmente atendem a consulta do usuário (documentos relevantes, com *x* elementos), e os outros documentos que foram recuperados pois o método de busca os selecionou por engano (*y*). Assim, podemos definir dois termos que são métricas fundamentais de desempenho em sistemas de recuperação de informações (SALTON e McGILL, 1983, DAMIANI *et al.*, 1999). O primeiro termo definido é revocação (*recall*), e corresponde ao número de documentos recuperados que satisfazem a consulta (*x*) sobre o número total de documentos que satisfazem a consulta (*n*). Portanto, revocação é a percentagem de componentes relevantes recuperados.

O segundo termo é a precisão (*precision*) e corresponde ao número de documentos recuperados que satisfazem a consulta (*x*) sobre o número total de documentos recuperados (*x* + *y*). O *precision* corresponde ao percentual dos documentos recuperados que são relevantes.

II.4.1. Técnicas Tradicionais para a Recuperação de Código

Essas técnicas executam basicamente uma busca por elementos, ou termos, léxicos ou sintáticos nos componentes reutilizáveis, como palavras ou frases. As buscas léxicas ignoram quaisquer relações, sintáticas ou semânticas, entre os elementos. As buscas sintáticas se baseiam em comparações das estruturas em que estão organizados os elementos que descrevem os componentes.

Então, os métodos de busca de componentes estão diretamente relacionados com a forma em que os componentes são representados, podendo esta representação limitar os tipos de busca que podem ser efetuados. Se um componente é descrito através de uma documentação textual, será impossível para um sistema de apoio à reutilização executar buscas que considerem o comportamento deste componente.

Entre as principais técnicas tradicionais para a recuperação de código destacam-se a Busca por Palavras Chaves (busca léxica a partir de palavras-chaves indexadas nos componentes), a Classificação Facetada (cada componente é descrito por um conjunto de características, denominadas facetas, sendo realizadas busca pelas facetas) e Recuperação Automática de Informações (busca léxica a partir da

documentação – comentários ou código fonte – dos componentes, permitindo a classificação automática e consultas em linguagens naturais).

II.4.2. Padrões

Nos últimos anos vêm crescendo a utilização de padrões para a reutilização de software orientado a objetos (JOHNSON, 1997). Mas o conceito de padrão é antigo e começou a ser aplicado em projetos arquitetônicos e suas reutilizações. Segundo ALEXANDER (1979), um padrão descreve um problema que ocorre repetidamente em nosso meio e inclui uma solução genérica para o mesmo, de tal maneira que se pode usá-la mais de um milhão de vezes, sem nunca fazê-lo de forma idêntica.

Esta definição foi aplicada em várias áreas de conhecimento como, por exemplo, análise de sistemas, programação e projeto de sistemas. A aplicação desta técnica ganhou força com a adoção dos paradigmas da orientação a objetos no desenvolvimento e implementação de software. Neste contexto, um padrão pode ser visto como um conjunto de classes e objetos que associados são utilizados para resolver um tipo de problema. Os padrões suportam a construção de instâncias parametrizadas, assim como a criação de instâncias de alto nível, como no caso de classes genéricas (LEA, 1994).

A recuperação de componentes ocorre através da identificação de padrões de projetos (*design patterns*) que podem ser aplicados para resolver um determinado problema. Em geral, um padrão de projeto inclui várias classes que devem ser (re) utilizadas. Para maiores informações sobre a reutilização de componentes através de padrões de projeto consulte (JOHNSON, 1997) e GAMMA *et al.* (1995).

II.4.3. Recuperação por Métodos Formais

Neste tipo de busca, todos os componentes da biblioteca devem ter uma descrição através de métodos formais. Um método é formal se possui uma base matemática robusta, tipicamente dada por uma linguagem formal de especificação. Vários métodos formais, e suas correspondentes linguagens, podem ser encontrados na literatura. Para uma visão global, porém mais detalhada, sugerimos WING (1990) e COHEN *et al.* (1986). Um exemplo de método e linguagem formal aplicado à reutilização de software e componentes pode ser encontrado em (XEXEO, 1994).

Segundo ZAREMSKI e WING (1997), a descrição de um componente em uma linguagem formal é composta por uma assinatura (interface do componente) e um conjunto de teoremas que transformam as informações de entrada nos resultados da execução do componente. Esses teoremas podem então ser provados com o auxílio de um provador de teoremas, como por exemplo o *Larch Prover* (GARLAND e GUTTAG, 1991).

Essa técnica possui diversas vantagens como: a eliminação da ambigüidade das descrições textuais, a garantia de qualidade (a correção de um componente pode ser demonstrada provando-se o conjunto de teoremas que o descreve), e principalmente, o fato da descrição formal manter informações sobre o comportamento do componente. A principal desvantagem está na dificuldade de se descrever um componente de médio ou grande porte em uma linguagem formal. Além disso, é pouco provável que todos os desenvolvedores de componentes utilizem os métodos formais como forma de documentação.

Entre os ambientes de recuperação de componentes que adotam este tipo de recuperação estão o PARIS (KATZ *et al.*, 1987) e o CAPS (STEIGERWALD, 1992).

II.4.4. Busca por Comportamento

Todas as técnicas vistas até aqui, com exceção da busca por métodos formais, não utilizam a propriedade fundamental que distingue procedimentos e funções de outros tipos de textos: a possibilidade de execução de um método e sua capacidade de transformar entradas em saídas processadas. A busca por comportamento explora essa propriedade. Neste tipo de busca, os componentes são recuperados pelo seu comportamento e não através dos termos que o descrevem.

A busca por comportamento (*Behavior Sampling*) foi proposta por PIERCE e PODGURSKY (1992) para recuperar funções em uma biblioteca de funções para reutilização. A característica principal deste método é recuperar componentes levando em consideração as suas execuções, ou seja, a transformação de entradas em saídas.

II.4.4.1. Funcionamento

Para implementar a busca por comportamento, é necessário oferecer um mecanismo para que o usuário que esteja procurando algum componente possa especificar os parâmetros de entrada e saída de cada componente procurado, definindo

o número e o tipo de cada parâmetro, ou seja, a interface do componente. Os nomes de cada parâmetro não precisam ser especificados. Com a interface do usuário definida (interface alvo), são identificados, dentro da biblioteca de componentes, os componentes que possuem uma interface compatível com a interface alvo (componentes candidatos). Durante a comparação, a ordem dos parâmetros é ignorada já que esta ordem não constitui um importante critério para a reutilização.

Assim, depois desta fase preliminar, o usuário deve informar um ou mais conjuntos de valores de entrada para os parâmetros e as saídas esperadas para cada conjunto. Cada conjunto de valores constitui uma amostra e deve especificar um valor para cada parâmetro, seja de entrada ou saída. Os componentes candidatos são então executados, recebendo como entrada os valores definidos pelo usuário. Após a execução, os parâmetros de saída são comparados com as saídas especificadas pelo usuário. Se forem iguais, o componente atende aos requisitos da busca e pode ser recuperado para a reutilização. Se o usuário especificou mais de uma amostra de entrada, os componentes devem ser executados para todas elas e apenas são recuperados os componentes cujas saídas obtidas forem iguais as saídas esperadas, para todas as amostras utilizadas.

Apesar do mecanismo simples, alguns problemas surgem durante a implementação deste tipo de busca (PODGURSKI e PIERCE, 1993). Componentes que falharem durante a execução com uma das amostras de entrada não devem ser executados para as outras amostras. A ferramenta de busca deve ser capaz de tratar exceções ocorridas durante a execução dos componentes candidatos. Também pode ser utilizado um mecanismo de interrupção por relógio (*time-out*) para manipular as rotinas que não terminem dentro de um intervalo de tempo esperado. Componentes que possuem este tipo de comportamento devem ser considerados como inaptos e não constarem da relação de recuperados para o usuário. Esse e outros problemas são abordados na Seção II.4.4.3.

II.4.4.2. Eficiência

A precisão de uma ferramenta para a recuperação de componentes determina quanto esforço será gasto pelo usuário para selecionar entre os componentes recuperados quais satisfazem os seus requisitos. A precisão da busca por comportamento aumenta naturalmente com o aumento do tamanho da amostra de

entrada. Quanto maior o número de amostras de entrada, mais restritiva é a busca. Mas o custo do método também aumenta com o tamanho da amostra e sua eficiência decresce. Logo, a utilidade do método depende do tamanho da amostra de entrada a ser executada sobre cada componente. Através de experimentos realizados sobre uma biblioteca de funções escritas na linguagem C, PODGURSKI e PIERCE (1993) mostraram que em oitenta e três porcento das pesquisas, o tamanho da amostra necessário para distinguir os componentes que atendiam aos requerimentos do usuário dos que não serviam foi de no máximo cinco entradas. O tamanho da amostra de entrada médio para distinguir os componentes foi de apenas 2,6 entradas.

Os resultados mostraram que a busca por comportamento é muito precisa, mesmo quando usamos uma amostra de entrada pequena. Outro fator importante para a eficiência do método, diz respeito ao número de componentes candidatos que devem ser executados. Se forem necessários executar muitos componentes, a recuperação pode demorar muito, ao passo que um número pequeno de componentes candidatos a serem executados corresponde a uma baixa revocação. Para evitar um grande número de componentes candidatos com o objetivo de restringir os componentes a serem executados, podem ser utilizados outros métodos de busca e recuperação, como os mostrados na Seção II.4.1.

II.4.4.3. Problemas e Soluções

Conforme destacado acima, o grande número de componentes candidatos a serem executados pode se tornar um problema para a utilização deste método. Além deste, a busca por comportamento possui alguns problemas que podem limitar a sua aplicação. Estes problemas e algumas soluções propostas na literatura são abordados nesta seção.

HALL (1993) propôs uma técnica denominada Recuperação baseada em Comportamento Generalizado (*Generalized Behavior-based Retrieval* - GBR), que pretende contornar alguns dos problemas da busca por comportamento. Um dos problemas abordados é o teste de componentes com comportamento complexo ou que envolvam efeitos colaterais indesejáveis. Um exemplo deste tipo de componente é uma função que apaga todos os arquivos de um diretório. Neste caso, os arquivos do diretório deveriam ser copiados e restaurados após a execução do método. Para contornar este problema, o GBR não executa o código do componente propriamente,

mas um modelo funcional definido para componentes complexos que seja livre de efeitos colaterais.

Outro problema contemplado pelo GBR é a baixa revocação, resolvida com a procura não apenas de componentes mais também de rotinas ou sub-rotinas que combinadas possam atender a consulta do usuário. A combinação de uma ou mais rotinas é possível devido a utilização de um modelo funcional para cada componente ou rotina.

HALL (1993) justifica que o esforço de escrever esses modelos é amortizado pelo tempo de funcionamento da biblioteca. Mas se um esforço extra deve ser gasto para incorporar um componente a uma biblioteca de recuperação, uma alternativa melhor é aplicar este esforço para definir uma especificação formal do componente. Conforme visto na Seção II.4.3 a especificação formal permite recuperar o componente com base no seu comportamento e também testar a sua correção.

Outro problema existente na Busca por Comportamento é a necessidade de o usuário fornecer todos os argumentos para os componentes candidatos. Um grande número de componentes candidatos pode sobrecarregar o usuário, portanto o ambiente de recuperação deve prover meios para facilitar a especificação dos parâmetros para os componentes.

PODGURSKI e PIERCE (1993) destacam também que a aplicação da busca por comportamento em ambientes que envolvam tipos abstratos de dados (classes, por exemplo) pode ser problemática, pois em geral, esses tipos envolvem estruturas de dados complexas e inter-relacionadas.

Alguns componentes podem retornar resultados "parecidos" ou próximo dos resultados esperados pelo usuário (uma função numérica pode retornar sempre o valor inverso ao esperado, ou uma função de cadeia de caracter retornar uma cadeia em maiúscula e não em minúsculas). A extensão mais flexível para este caso é permitir ao usuário especificar também uma função que é executada para verificar se uma saída é adequada ou não (PODGURSKI e PIERCE, 1993).

Nesta Seção, foram abordados os principais problemas do método de busca por comportamento. Através da análise desses problemas, conclui-se que a utilização da busca por comportamento depende das extensões adotadas e do tipo de aplicação desejado. PODGURSKI e PIERCE (1993) destacam que a busca por comportamento pode não ser aplicável para a recuperação de qualquer tipo de componente. O Capítulo

IV apresenta uma técnica de comparação que utiliza este método de busca, com algumas adaptações, para identificar classes similares em um banco de dados orientados a objetos.

II.5. OUTRAS PROPOSTAS

Durante os últimos anos, surgiram várias propostas para a realização de integração de esquemas, envolvendo diversos modelos de dados, estratégias e técnicas para a comparação de classes e detecção de conflitos.

Em PITOURA *et al.* (1995), os conflitos são solucionados com a ajuda dos métodos. Eles são utilizados para mapear um atributo em outro, implementando funções de conversão. Essa técnica foi denominada mapeamento operacional e utiliza também as características típicas do modelo orientado a objetos, como por exemplo, herança e polimorfismo, permitindo realizar o mapeamento de uma hierarquia de classes.

LI e CLIFTON (1994) propuseram a criação de uma ferramenta semi-automática que utilizasse tanto o metadado, quanto os dados armazenados como fonte de informações para uma rede neural. O número de nós de entrada da rede neural é igual ao número de características consideradas durante a comparação. O número de nós de saída é igual ao número de categorias existentes nos esquemas. Essas categorias são determinadas com a ajuda de um algoritmo de agrupamento classificador de padrões. Cada padrão corresponde a um novo conceito e, portanto, a uma nova categoria. Também são apresentados algoritmos para o treinamento da rede neural.

A lógica difusa foi utilizada em FANKHAUSER *et al.* (1991) para determinar a semelhança entre classes. A relação entre as classes é classificada em quatro tipos: generalização, especialização, associação negativa (relações complementares, incompatíveis ou antagônicas) e associação positiva (sinônimos). Uma rede de associação entre os conceitos é criada, com cada caminho possuindo um tipo e um peso (número entre 0 e 1 que indica a "força do relacionamento"). Assim, a equivalência entre as classes é calculada percorrendo-se os caminhos que as interligam, utilizando funções de agregação para obter o grau de semelhança.

III. Lógica Difusa

III.1. Introdução

Os princípios da lógica difusa foram propostos por ZADEH (1965) na primeira publicação sobre o assunto mas apenas na última década a lógica difusa tem sido amplamente divulgada, surgindo recentemente vários trabalhos sobre lógica difusa e suas aplicações.

Os conceitos da Teoria de Conjuntos e Lógica Difusa e suas aplicações formam um método de trabalho conceitual novo, que possui muitos pontos em comum com os modelos desenvolvidos para a Teoria e Lógica de Conjuntos Ordinários, mas possui um escopo de aplicação mais amplo por ser mais genérico, principalmente nas áreas de classificação de padrões, processamento de informações, controles especialistas e de tomada de decisões. Essencialmente, este novo modelo de trabalho proporciona uma maneira natural de lidar com problemas nos quais a fonte de incerteza é a ausência de um critério bem definido para determinar a presença de um elemento em um conjunto. ZADEH (1965) introduziu a Teoria de Conjuntos Difusos como um novo paradigma para representar o pensamento humano que é capaz de tomar decisões e realizar uma classificação baseada apenas em informações incompletas e ambíguas.

A lógica difusa proporciona um meio de representar incertezas, possibilidades e aproximações. Onde as teorias tradicionais utilizam os valores zero e um, que expressam a completa ausência e a certeza absoluta, respectivamente, a teoria difusa trabalha com o intervalo [0,1], flexibilizando as restrições e limites dos conjuntos avaliados.

Este Capítulo apresenta os conceitos básicos da lógica difusa utilizados no desenvolvimento da Tese. Existe uma extensa bibliografia disponível na área e uma revisão mais completa pode ser obtida em COX (1995). A próxima Seção introduz os conceitos da teoria e conjuntos difusos. A Seção seguinte descreve as relações difusas. A Seção III.4 apresenta os conceitos das funções de pertinência e suas fórmulas de cálculo. A última Seção mostra algumas técnicas que podem ser utilizadas em um processo de decisão difuso.

III.2. Teoria de Conjuntos Difusos

Na Teoria de Conjuntos tradicional, um elemento obrigatoriamente pertence a um conjunto ou não, não havendo meio termo (conceito conhecido como a Lei de Exclusão Mútua). A Teoria de Conjuntos Difusos é uma extensão da teoria tradicional, na qual o elemento possui um grau de pertinência em relação a um conjunto. Em um conjunto tradicional, um elemento somente pertence ao conjunto se sua pertinência é total (igual a um), enquanto que elementos de um conjunto difuso podem ter pertinências parciais (menores que um).

A pertinência ao conjunto pode ser descrita como sendo uma função que determina quanto um elemento possui de características comuns ao conjunto. Dado um universo de elementos U, uma função de característica f_A descreve o conjunto (não difuso) A como:

$$f_A : U \rightarrow \{0,1\}$$
$$\begin{cases} f_A(x) = 1, \text{ se } x \in A \\ f_A(x) = 0, \text{ se } x \notin A \end{cases} \qquad 3.1$$

Conjuntos difusos usam uma função de pertinência, μ_A, que amplia a imagem de f para [0,1]:

$$\mu_A : U \rightarrow [0,1], \qquad 3.2$$

onde a função μ_A é construída caso a caso para representar um grau de pertinência. Também é comum usar $A(x)$ para representar $\mu_A(x)$. Para um universo de discurso finito $\{x_1, x_2, \ldots, x_n\}$, a função μ_A pode ser representada também deste modo:

$$\mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \mu_A(x_3)/x_3 + \ldots + \mu_A(x_n)/x_n, \qquad 3.3$$

Nesta notação, o denominador não é um quociente e sim um delimitador que indica o elemento avaliado pela função de pertinência. Um exemplo de um conjunto difuso representado através da notação acima é o conjunto de números próximos de quatro para o universo $\{1, 2, 3, 4, 5, 6, 7\}$:

$$0.2/1 + 0.5/2 + 0.8/3 + 1/4 + 0.8/5 + 0.5/6 + 0.2/7.$$

III.2.1. Operações sobre Conjuntos Difusos

Todas as operações definidas para os conjuntos tradicionais podem ser aplicadas aos conjuntos difusos com exceção da Lei de Exclusão Mútua e da Lei da Contradição. As Figuras III.1 e III.2 ilustram essas duas leis para os conjuntos tradicionais e para os conjuntos difusos, respectivamente. Em *(a)*, é apresentado o conjunto A e seu complemento (A') e as Figuras *(b)* e *(c)* mostram $A \cup A' = X$ (Lei da Exclusão Mútua) e $A \cap A' = \varnothing$ (Lei da Contradição).

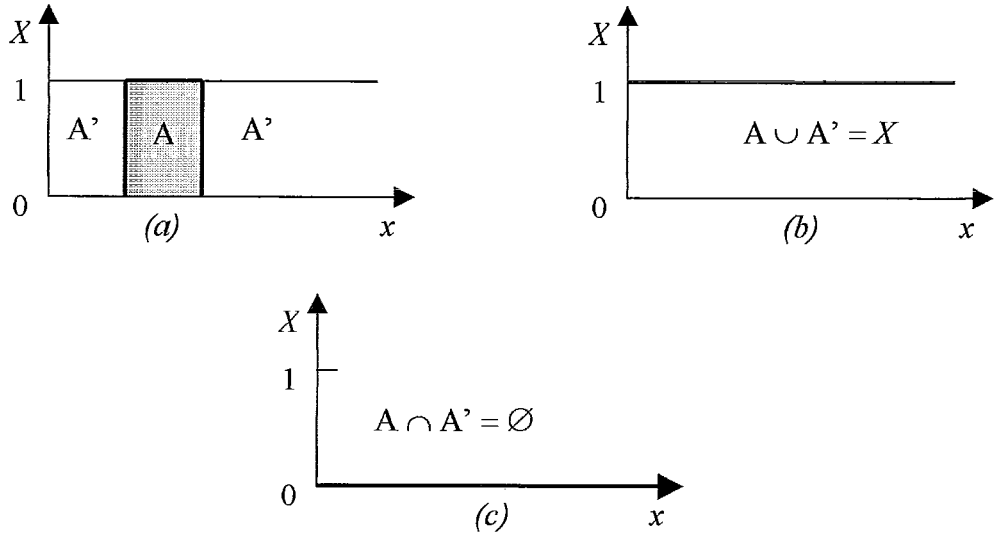


Figura III.1 – Lei da Exclusão Mútua para conjuntos tradicionais

Os elementos de conjuntos difusos podem possuir uma pertinência parcial ao conjunto ($\mu_A(x) < 1$) e portanto podem pertencer tanto ao conjunto A, quanto ao seu complemento A' (calculado como $1 - \mu_A(x)$), invalidando as duas leis acima.

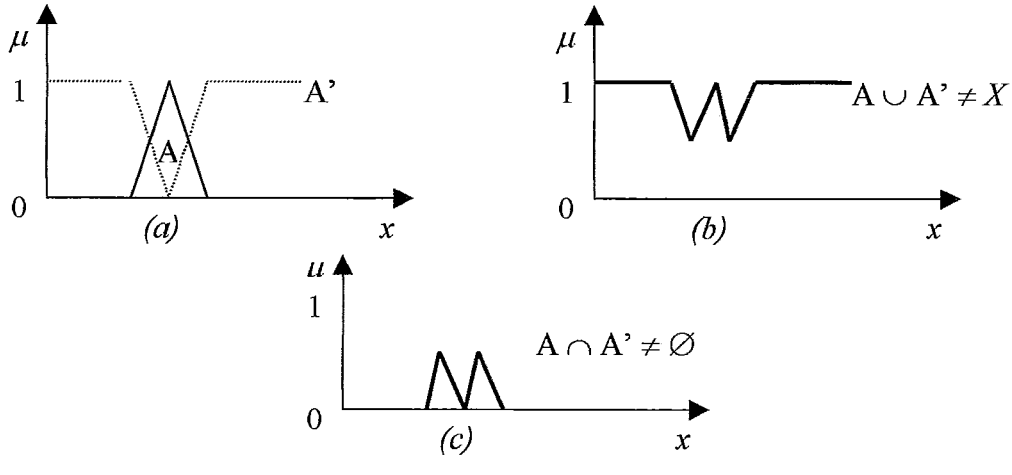


Figura III.2 – Lei da Exclusão Mútua para conjuntos difusos

Sendo A, B e C, três conjuntos difusos no universo X, as operações básicas de união, interseção e complemento são definidas para conjuntos difusos como sendo:

Interseção:	$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x)$	3.4
União:	$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x)$	3.5
Complemento:	$\mu_{A'}(x) = 1 - \mu_A(x)$	3.6

Os operadores $\wedge$ e $\vee$ podem ser quaisquer funções binárias que satisfaçam algumas condições, formando uma classe de funções denominada *t-norm* (ou norma triangular) para interseção e *s-norm* (ou *t-conorm*) para união:

1. Identidade:
   Sendo 1 universo e 0 vazio: $1 \wedge x = x$, $1 \vee x = x$ e
   $0 \vee x = x$, $0 \wedge x = 0$
2. Comutatividade: $x \wedge y = y \wedge x$
   $x \vee y = y \vee x$
3. Monotocidade: se $w \leq x$ então $w \wedge y \leq x \wedge y$
   se $w \leq x$ então $w \vee y \leq x \vee y$
4. Associatividade: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
   $x \vee (y \vee z) = (x \vee y) \vee z$

A função padrão *t-norm* para a interseção é a *min*, que é também a única idempotente, ou seja, $x \wedge x = x$. Para união, a função padrão *s-norm* é a *max*, que também é idempotente, $x \vee x = x$. Outras possíveis funções *t-norm* e suas equivalentes *s-norm* são apresentadas na Tabela III.1.

Em geral, o complemento é calculado utilizando a fórmula da Equação 3.6. Mas outras funções podem ser utilizadas, como o complemento de Sugeno (COX, 1994).

Além disso, como os conjuntos difusos trabalham com o intervalo [0,1], ao invés de apenas {0, 1}, outras operações podem ser realizadas, destacado-se a operação de agregação utilizada, por exemplo, para agregar um conjunto de opiniões de especialistas. O operador de agregação combina vários conjuntos difusos para formar apenas um único conjunto. Existem diversas formas de agregação, efetuadas com diferentes funções, como por exemplo min, max, média, média ponderada, etc.. O operador de agregação possui um significado especial no contexto do sistema ou

aplicação difusa em que for utilizado e portanto a escolha de qual função utilizar para realizar uma agregação é orientada pelos objetivos que a aplicação espera alcançar.

Tabela III.1 – Funções para interseção e união

Tipo	t-norm	t-conorm
Padrão	$min(\mu_A(x), \mu_B(x))$	$max(\mu_A(x), \mu_B(x))$
Algébrica	$\mu_A(x) \cdot \mu_B(x)$	$\mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$
Limitada	$Max(0, \mu_A(x) + \mu_B(x) - 1)$	$min(1, \mu_A(x) + \mu_B(x))$
Robusta	$\begin{cases} \mu_A(x), \text{se } \mu_B(x) = 1 \\ \mu_B(x), \text{se } \mu_A(x) = 1 \\ 0, \text{ caso contrário} \end{cases}$	$\begin{cases} \mu_A(x), \text{se } \mu_B(x) = 0 \\ \mu_B(x), \text{se } \mu_A(x) = 0 \\ 1, \text{ caso contrário} \end{cases}$

III.3. RELAÇÃO DIFUSA

Esta Seção introduz a noção de Relação que serve de base para várias operações sobre conjuntos como, por exemplo, produtos cartesianos, composição de relações e propriedades de equivalência.

III.3.1. Produto Cartesiano

Produto cartesiano pode ser entendido como um meio de produzir um relacionamento entre conjuntos. Sendo A e B dois conjuntos difusos, o produto cartesiano é representado como A × B e consiste na combinação, não ordenada, dos elementos de A com os elementos de B. Um exemplo de um produto cartesiano, R = A × B, é:

$$\mu_R(x, y) = \mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y)) \qquad 3.7$$

III.3.2. Relações Difusas

Uma relação é um subconjunto do produto cartesiano de dois ou mais conjuntos, seja na teoria tradicional ou na lógica difusa. Para cada elemento da relação é associado um valor, denominado "força de relacionamento" entre os elementos de cada conjunto que compõem a relação. Esse valor é obtido através de uma função *X*.

Para relações tradicionais, esta função retorna 0 (nenhum relacionamento) ou 1 (completamente relacionado):

$$X_{A\times B}(a, b) = \begin{cases} 1, (a,b) \in A\times B \\ 0, (a,b) \notin A\times B \end{cases} \qquad 3.8$$

Para relações difusas, este valor é obtido através de um função de pertinência μ (Equação 3.9). As relações difusas podem ser consideradas conjuntos difusos e todas as operações definidas para os conjuntos são aplicáveis também às relações.

$$\mu_R(x, y) = \mu_{A\times B}(x, y) = \min(\mu_A(x), \mu_B(y)) \qquad 3.9$$

onde $R = A\times B$, $x \in A$, $y \in B$ e μ_R é a função de pertinência da relação.

Uma relação pode ser expressa também na forma de uma matriz cujas linhas e colunas são determinadas pelos elementos dos universos que formam a relação.

III.3.3. Composição de Relações

Composição é uma operação aplicável sobre relações. Seja R uma relação entre os universos X e Y e S uma relação entre os universos Y e Z, a composição é utilizada para descobrir uma relação T, que relacione elementos de X e Z, a partir das relações R e S.

Existem duas formas básicas de composição: a primeira é denominada composição max-min e a outra composição max-produto. A composição max-min é definida como:

$$T = R \circ S$$

$$\mu_T(x, z) = \bigvee_{y \in Y} (\mu_R(x, y) \wedge \mu_S(y, z)) \qquad 3.10$$

A composição max-produto é definida pela Equação 3.11.

$$T = R \circ S$$

$$\mu_T(x, z) = \bigvee_{y \in Y} (\mu_R(x, y) \bullet \mu_S(y, z)) \qquad 3.11$$

Nessas Equações, $\wedge$ representa a função *min*, $\vee$ a função *max* e $\bullet$ o operador de multiplicação (algébrica).

III.3.4. Propriedades

As relações (difusas ou não) podem ter várias propriedades importantes, com destaque para reflexividade, simetria e transitividade. Essas propriedades existem também para grafos e são traduzidas para relações difusas como:

1. Reflexividade: $\mu_R(x, x) = 1$
2. Simetria: $\mu_R(x, y) = \mu_R(y, x)$
3. Transitividade: $\mu_R(x, y) = \lambda_1$, $\mu_R(y, k) = \lambda_2$, $\mu_R(x, k) = \lambda$ onde $\lambda \geq \min[\lambda_1, \lambda_2]$

Uma relação difusa é considerada uma **relação de equivalência** se ela possuir as três propriedades definidas acima. Uma relação é dita de **tolerância** quando possui apenas as propriedades de reflexividade e simetria. A partir de uma relação de tolerância entre n elementos, é possível determinar uma relação de equivalência através de no máximo n – 1 composições. Para maiores detalhes sobre essas demonstrações, pode-se consultar ROSS (1995).

III.4. Função de Pertinência

Esta Seção aborda os meios existentes para determinar os valores de pertinência para cada elemento de um conjunto difuso ou para os valores de uma relação difusa. Esses valores são determinados por funções, denominadas funções de pertinência, que refletem uma modelagem do problema analisado.

Para determinar uma função de pertinência de um conjunto difuso é necessário analisar, coletar e obter resultados reais do universo estudado. Um dos métodos mais adotados para determinar a função de pertinência é a utilização de regras lingüísticas,

que são regras definidas por especialistas e transformadas em regras da forma se – então ou modeladas como funções. Em geral, é mais prático escolher a forma da função capaz de modelar o problema antes de determiná-la. Duas formas básicas, que se adaptam a maioria dos problemas e são facilmente calculáveis, são as funções triangulares e trapezóides. Exemplos de funções de pertinência destes tipos aparecem na Figura III.3. Outras funções, representadas por curvas mais ou menos complexas, podem ser utilizadas, como por exemplo, funções lineares, na forma de curvas S, etc..

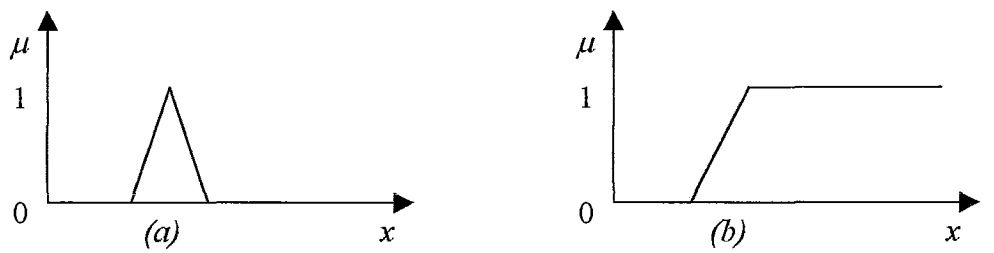


Figura III.3 – Funções de pertinência triangulares e trapezóides

Para as relações difusas, o valor associado a um elemento da relação também é determinado por uma função de pertinência já que uma relação difusa é também um conjunto difuso. Porém, a função de pertinência pode ser calculada através de outros métodos como, por exemplo, o produto cartesiano, agregação ou métodos de similaridade por manipulação de dados. No último caso, os valores das relações são calculados a partir das informações disponíveis sobre cada elemento da relação. Existem alguns procedimentos que permitem realizar este cálculo, como amplitude de coseno, coeficiente de similaridade exponencial (DUBOIS e PRADE, 1980) e outros. Todos esses métodos procuram determinar algum tipo de padrão nos dados através de métricas. Na prática, um método de similaridade é qualquer método capaz de calcular o valor da função de pertinência baseado na manipulação de informações disponíveis para cada elemento de uma relação (ROSS, 1995).

O próximo Capítulo, que apresenta a estratégia de comparação proposta, descreve um método de similaridade por manipulação de dados para calcular a similaridade entre duas classes.

III.5. Processo de Decisão

Uma das aplicações da Lógica Difusa é no auxílio ao processo de tomada de decisões. O processo de decisão envolve a coleta de informações sobre um problema

proposto (ou um conjunto de elementos) e a escolha de uma solução (ou um elemento do conjunto). Para alguns problemas, as informações disponíveis contêm um grau de incerteza que pode influir na escolha de uma boa decisão. Para auxiliar este processo de decisão, foram desenvolvidas diversas estratégias baseadas na Teoria da Lógica Difusa capazes de manipular as incertezas e imprecisões das informações em busca da melhor alternativa.

A literatura disponível nesta área é extensa e inclui diversas estratégias como a avaliação sintática difusa, ordenação difusa, método de decisão bayesiano com lógica difusa, tomada de decisão com múltiplos objetivos, preferência e consenso, e outros (ROSS, 1995, SAKAWA, 1993, ADAMS, 1994).

Cada estratégia considera a incerteza de uma maneira diferente, portanto, não existe um método melhor ou pior que outro, mas apenas métodos que se adaptam melhor ao tipo de problema proposto do que outros. A seguir, é apresentada a estratégia de tomada de decisão utilizada na técnica de comparação de classes proposta no próximo Capítulo, a Ordenação Difusa (*Fuzzy Ordering*). Esta estratégia se enquadrou muito bem dentro do modelo do problema proposto, principalmente por permitir a ordenação de elementos através de relações difusas não simétricas. Este tipo de relação é obtido durante a comparação de classes.

III.5.1. Ordenação Difusa

A Ordenação Difusa é baseada em uma noção da relatividade entre os elementos de uma relação. Primeiro, considere que x e y são elementos de um universo X e que existem duas funções de comparação de pares de elementos, expressas em $f_y(x)$ como sendo a pertinência de x com relação a y, e em $f_x(y)$ como sendo a pertinência de y em relação a x. Esta noção de pertinência pode representar um grau de um relacionamento qualquer entre x e y.

Então, é possível definir uma função de relatividade entre x e y:

$$f(x, \mid y) = \frac{f_y(x)}{\max\left(f_x(y), f_y(x)\right)} \qquad 3.12$$

Esta função expressa o valor de pertinência em escolher x ao invés de y. A Equação 3.12 pode ser generalizada para muitas variáveis, x_1, x_2, ..., x_n, definidas no

universo X e agrupadas no conjunto $A = \{ x_1, x_2, ..., x_i, ..., x_n\}$. Sendo A' um conjunto idêntico a A, com exceção da ausência do elemento x_i, a função de relatividade para um conjunto transforma-se em,

$$f(x_i \mid A') = f(x_i \mid x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n) = \qquad 3.13$$
$$\min \{ f(x_i \mid x_1), f(x_i \mid x_2), ..., f(x_i \mid x_{i-1}), f(x_i \mid x_{i+1}), ..., f(x_i \mid x_n) \}$$

Esta equação representa uma medida nebulosa da escolha de x_i sobre todos os elementos de A'. A função mínimo foi utilizada já que a equação acima promove a interseção de várias variáveis. Assim, como a função de relatividade de uma variável com ela mesma é a unidade, isto é, $f(x_i \mid x_i) = 1$, então:

$$f(x_i \mid A) = f(x_i \mid A') \qquad 3.14$$

Com os resultados obtidos da função de relatividade, é construída uma matriz quadrada de ordem n, denominada matriz de comparação C, onde cada elemento é o valor da função $f(x_i \mid x_j)$ tal que i, j = 1, 2, ..., n e x_i e x_j pertencem ao universo X. Para determinar a ordenação final, é calculado o menor valor de cada linha da matriz C, isto é:

$$C_i = \min f(x_i \mid X),\ i = 1, 2, ..., n \qquad 3.15$$

onde C_i é o valor de pertinência do ranking para a variável i. A função de mínimo é utilizada pois ela tem um peso menor na definição da ordenação e além disso, a função de máximo, se utilizada, freqüentemente retornaria 1, resultando em uma ambigüidade e indefinição na ordenação em decorrência de empates.

A utilização deste tipo de ordenação engloba a subjetividade da comparação de um elemento com outro e é mais efetivo do que uma simples ordenação por valores de pertinência atribuídos às variáveis comparadas.

IV. Comparação de Classes baseada no Comportamento

IV.1. Introdução

A comparação de classes proposta neste trabalho se destaca por utilizar duas técnicas, lógica difusa e comparação do comportamento, com o objetivo de captar melhor a semântica de cada classe. Neste sentido, ela se distingue dos métodos vistos no Capítulo II, que estão baseados na comparação sintática (estrutural) e na comparação semântica (semântica envolvendo nomes ou estruturas de relacionamentos), mas que não envolvem o comportamento dos métodos.

Este Capítulo apresenta o funcionamento da técnica de comparação implementada. Primeiramente é apresentada a estrutura da técnica de comparação e a presença da lógica difusa durante o processo. Em seguida, são detalhadas as etapas da comparação, com destaque para a última etapa, que corresponde a comparação do comportamento. Finalmente é apresentada a técnica de lógica difusa utilizada para classificar os resultados obtidos na comparação.

IV.2. Técnica de Comparação

O propósito deste trabalho é comparar duas ou mais classes (componentes ou não de esquemas diferentes), considerando não apenas a estrutura e o nome das classes comparadas mas também o comportamento dos seus métodos, avaliados a partir dos resultados obtidos após as suas execuções. A comparação de classes implementada utiliza um método de similaridade próprio, baseado na lógica difusa. Nesse contexto, as classes de cada esquema (diferentes ou não) a serem comparadas constituem o universo a partir do qual serão formados pares ordenados entre as classes. Cada par de classes está associado a uma "força de relacionamento" entre seus elementos. O conjunto dos pares ordenados forma uma relação difusa que será utilizada para classificar e identificar similaridades entre as classes.

Para determinar os valores da relação dos pares de classes foi utilizado um método de similaridade por manipulação de dados (Seção III.4). Neste trabalho, os dados manipulados pelo método de similaridade correspondem às informações disponíveis de cada classe, no caso: nome, atributos, assinatura de métodos e comportamento. Estas informações são combinadas para valorar ou quantificar a

"força de relacionamento" entre as classes. Como foi visto no Capítulo II, diversos trabalhos identificam a semelhança entre classes considerando um ou mais aspectos na comparação. A abordagem adotada é bem similar a utilizada em SOUZA (1986a), onde cada aspecto considerado no processo de comparação é avaliado através de uma função de similaridade e a similaridade entre as classes é composta pela agregação dos valores retornados pelas funções. Existem diversas maneiras de agregar os valores das funções de similaridades, entre elas a solução adotada por SOUZA (1986a), e repetida aqui, na qual os valores são combinados através de pesos, conforme a Equação 2.1. A escolha dos valores dos pesos de cada aspecto permite selecionar os aspectos de maior importância (maior peso) e de menor importância. Os pesos adotados nesta tese são explicados nas próximas seções, mas é importante ressaltar que os valores associados aos pesos podem ser configurados antes da aplicação da técnica e portanto correspondem a um parâmetro de implementação. O Anexo I apresenta os valores de todos os pesos utilizados na técnica proposta.

Ainda como em SOUZA (1986a), as funções de similaridade adotadas para cada aspecto considerado na comparação de classes podem ser formadas por outras funções, constituindo uma hierarquia de funções. Este tipo de abordagem garante a flexibilidade do processo de comparação, permitindo que o mesmo seja estendido com novas formas e / ou estratégias de comparação. Apesar das diversas semelhanças, a técnica de comparação proposta se difere do trabalho de SOUZA (1986a) principalmente por considerar o comportamento dos componentes de cada esquema. O aspecto comportamental incorpora muitas informações semânticas sobre a classe (PITOURA *et al.*,1995), aperfeiçoando o processo de comparação. Além disso, a técnica de comparação desta tese está baseada no modelo orientado a objetos, enquanto em SOUZA (1986a), a comparação ocorria entre os elementos do modelo conceitual ACS.

A próxima seção explica a função de similaridade utilizada na comparação de duas classes. As seções seguintes apresentam o processo de comparação para cada um dos aspectos considerados.

IV.2.1. Comparação de Classes

A comparação de classes envolve quatro aspectos: nome das classes, seus atributos, a assinatura e o comportamento dos métodos. Esses aspectos estão

relacionados entre si, como, por exemplo, a assinatura dos métodos e o seu comportamento, isto é, a execução deles. Os atributos também estão relacionados aos métodos e ao comportamento já que estes podem ter resultados diferentes de acordo com os valores dos atributos. Apesar desses inter-relacionamentos, esses aspectos são comparados separadamente e depois combinados.

Assim, o resultado da comparação de duas classes é obtido por uma função de similaridade que considera estes quatro aspectos. Ao invés de utilizar uma função que expresse a equivalência entre as duas classes comparadas, a função de similaridade escolhida retorna o grau de pertinência de uma classe ser subtipo de outra. Uma relação de subtipo, também utilizada por THIEMES e SIEBES (1993) para determinar a similaridade entre classes, foi escolhida porque permite identificar classes que podem ser generalizações ou especializações de outras. Além disso, em muitos casos é satisfatório saber que uma classe é capaz de fazer tudo que uma outra faz, principalmente se o objetivo final da comparação é a reutilização de classe, não sendo necessário portanto que as duas classes sejam equivalentes.

A função de subtipo utilizada é parecida em alguns aspectos com a utilizada por THIEMES e SIEBES (1993) e apresentada na Seção II.3.1, sendo a principal diferença o fato da função retornar um número difuso de pertinência de uma classe ser subtipo de outra, ao invés de apenas determinar se uma classe é subtipo da outra ou não. Outra diferença importante ocorre na comparação dos métodos, já que o comportamento analisado em THIEMES e SIEBES (1993) está restrito aos comandos de atribuição, enquanto na técnica proposta o resultado da execução é considerado. Além disso, os nomes de classes, atributos e métodos foram ignorados por THIEMES e SIEBES (1993) mas estão presentes no processo de comparação aqui apresentado.

Portanto, cada um dos aspectos de uma classe é analisado sobre o critério de um conjunto ser subconjunto de outro. Para isso, cada aspecto possui uma função de similaridade própria e os resultados das quatro funções são agregados pela função de similaridade de classes com a utilização de pesos, segundo a Equação 4.1:

$$F_C = w_N \circ F_N + w_A \circ F_A + w_M \circ F_M + w_E \circ F_E \qquad 4.1$$

Nesta equação, w_N, w_A, w_M e w_E são os pesos associados, respectivamente, a comparação de nomes, atributos, as assinaturas e a execução dos métodos e F_N, F_A, F_M e F_E são as funções de similaridade para a comparação de nomes, atributos, assinaturas e execução de métodos, respectivamente.

Os valores atribuídos a cada um dos pesos podem variar, enfatizando alguns aspectos em detrimento de outros. Como o objetivo desta tese é realizar a comparação considerando também o comportamento das classes, os pesos associados aos métodos correspondem pela maioria do resultado final. Por outro lado, o peso da comparação dos nomes foi minimizado. De qualquer modo, a implementação foi elaborada de modo que os pesos podem ser configurados pelo usuário. A Tabela IV.1 mostra os valores atribuídos aos pesos da Equação 4.1 para cada um dos aspectos considerados. As próximas equações aparecem com os pesos adotados neste trabalho, mas estes valores podem ser alterados se desejado. O Anexo I apresenta todos os pesos configuráveis e os valores adotados para cada um deles.

Tabela IV.1 - Pesos de cada aspecto considerado na comparação de classes

Peso	Valor
w_N (nomes)	0,1
w_A (atributos)	0,3
w_M (assinatura dos métodos)	0,3
w_E (execução dos métodos)	0,3

Os valores obtidos através da função de similaridade da Equação 4.1 formam uma relação difusa que indica quanto uma classe representa um subconjunto de outra. Portanto, se a comparação de duas classes, c_1 e c_2, efetuada pela função F_C (c_1, c_2), tem como resultado 1, c_1 está inteiramente contido em c_2 ($c_1 \subseteq c_2$). Isto significa, por exemplo, que todos os atributos e métodos de c_1 possuem equivalentes em c_2 mas o inverso não é necessariamente verdade, ou seja F_C (c_1, c_2) pode ser diferente de F_C (c_2, c_1). Assim, se $c_1 \subseteq c_2$ então F_C (c_1, c_2) = 1 e F_C (c_2, c_1) $\leq F_C$ (c_1, c_2). Se o resultado obtido for zero, significa que c_1 não possui nada em comum (nomes, atributos, métodos e comportamento) com c_2. Note que neste caso, F_C (c_2, c_1) também será zero já que as duas classes não possuem nenhuma característica em comum.

A relação difusa assim definida é reflexiva mas não é simétrica. Portanto, esta relação não constitui uma relação de equivalência ou de tolerância (Seção III.3). Porém, esta relação pode ser utilizada para classificar e ordenar as classes comparadas através da técnica de Ordenação Difusa (Seção III.5.1). A seguir, são apresentadas as funções de similaridade utilizadas em cada um dos quatro aspectos que compõe a comparação de classes proposta.

IV.2.2. Comparação de Nomes

Para efetuar a comparação entre os nomes de classes foi utilizado um algoritmo muito simples, que está expresso na Equação 4.2. Conforme este algoritmo, a função de similaridade entre nomes retorna 1 se o primeiro nome, n_1, está contido ou é igual ao segundo nome, n_2. Caso o segundo nome esteja contido no primeiro, a função retorna 0,5. Caso contrário, a função retorna zero. Esta função é reflexiva mas não é simétrica.

$$F_N(n_1, n_2) = \begin{cases} 1{,}0 \text{ se } n_1 \subseteq n_2. \\ 0{,}5 \text{ se } n_1 \supseteq n_2. \\ 0 \text{ caso contrário} \end{cases} \qquad 4.2$$

Este algoritmo simples não prejudica o método de comparação proposto já que a participação da comparação de nomes no resultado final foi reduzida pela atribuição de valor menor no peso associado (veja a Tabela IV.1 - Pesos de cada aspecto considerado na comparação de classes).

Este algoritmo também é utilizado na comparação dos nomes dos atributos, métodos e parâmetros. Como será apresentado no próximo Capítulo, a comparação foi implementada de maneira que a adoção de algum outro algoritmo para comparar nomes pode ser efetuada sem a necessidade de alterar os outros aspectos da comparação. Assim, se desejado, é possível substituir a comparação dos nomes descrita acima, por um método que utilize um dicionário de sinônimos, por exemplo.

IV.2.3. Comparação de Atributos

A comparação de atributos de duas classes segue a noção de subtipo, assim como o método de comparação em geral. A função $F_A(c_1, c_2)$ é utilizada para obter a pertinência do conjunto de atributos, A_1, da primeira classe, c_1, ser um subconjunto

dos atributos (conjunto A_2) da segunda classe, c_2, ou seja, c_2 ser um subtipo de c_1. Para isso, cada atributo de A_1 é comparado com cada um dos atributos de A_2. A comparação de dois atributos é explicada na Seção IV.2.3.1. Ao término dessas comparações, são selecionados os pares de atributos comparados (um atributo da classe c_1 e outro de c_2) que possuam o maior grau de similaridade entre si. Esses pares são selecionados de modo que cada atributo de A_2 seja mapeado por um e somente um atributo de A_1 (o que tiver a maior similaridade com ele).

Os graus dos pares selecionados são somados e o total dividido pelo número de atributos da classe c_1. Este valor é retornado pela função F_A (c_1, c_2) da Equação 4.1. Assim, se F_A (c_1, c_2) = 1, todos os atributos de c_1 possuem pelo menos um atributo em c_2 que foi mapeado com grau 1 em relação a eles. Neste caso, F_A (c_2, c_1) $\leq$ F_A (c_1, c_2). Se $c_1 = c_2$, F_A (c_1, c_2) = F_A (c_2, c_1) = 1, ou seja, a função define uma relação reflexiva mas, novamente, a função assim definida não produz uma relação simétrica já que F_A (c_1, c_2) pode ser diferente de F_A (c_2, c_1).

IV.2.3.1. Comparação de Atributo com Atributo

A comparação de dois atributos também é realizada por uma função de similaridade, denominada F_a. O objetivo desta função é determinar o grau de similaridade entre dois atributos, a_1 e a_2, levando em consideração dois aspectos: o nome e o tipo dos atributos. Estes dois aspectos são comparados e os resultados combinados através de pesos, conforme a Equação 4.3:

$$F_a(a_1, a_2) = 0{,}3 \circ F_N(a_1.\text{nome}, a_2.\text{nome}) + 0{,}7 \circ F_T(a_1.\text{tipo}, a_2.\text{tipo}) \quad 4.3$$

Nesta equação, a função F_N (a_1.nome, a_2.nome) é a mesma utilizada na comparação dos nomes de duas classes (Equação 4.2) e F_T (a_1.tipo, a_2.tipo) é a comparação entre os tipos dos atributos. Esta última função é detalhada na Seção IV.3 e assim como o restante do método, determina quanto um tipo é igual ou é um supertipo de um outro tipo.

IV.2.4. Comparação de Assinaturas de Métodos

A comparação de assinaturas de métodos segue a mesma estratégia apresentada na comparação de atributos, ou seja, as assinaturas dos métodos são

comparadas com o objetivo de medir quanto o conjunto de métodos de uma classe pode ser considerado um subconjunto dos métodos de outra classe. Para isso, cada método da primeira classe, c_1, tem sua assinatura comparada com a de todos os métodos da segunda classe, c_2. A comparação das assinatura de dois métodos é explicada na próxima subseção. Como na comparação dos atributos, no final de todas as comparações são selecionados para cada método de c_1 o método de c_2 que possui o maior grau de similaridade com ele. Novamente, um método de c_2 somente pode ser mapeado para um único método de c_1. Os graus dos pares de métodos selecionados são somados e o total dividido pelo número de métodos de c_1. Este é o valor retornado pela função $F_M(c_1, c_2)$ da Equação 4.1.

Assim, se $F_M(c_1, c_2) = 1$, todos os métodos de c_1 possuem pelo menos um método em c_2 cuja a assinatura foi mapeada com grau 1 em relação a eles. Neste caso, $F_M(c_2, c_1) \leq F_M(c_1, c_2)$. Se $c_1 = c_2$, $F_M(c_1, c_2) = F_M(c_2, c_1) = 1$, ou seja, a função define uma relação reflexiva mas, novamente, a função assim definida não produz uma relação simétrica já que $F_M(c_1, c_2)$ pode ser diferente de $F_M(c_2, c_1)$.

IV.2.4.1. Comparação de Assinatura com Assinatura

A comparação da assinatura de dois métodos também é realizada por uma função de similaridade, denominada F_s. O objetivo desta função é determinar a pertinência de uma assinatura ser um subconjunto de outra. Para isso, são considerados três aspectos: o nome, o tipo, e os parâmetros dos métodos. Estes três aspectos são comparados e os resultados combinados através de pesos, conforme a Equação 4.4:

$$\begin{aligned} F_s(m_1, m_2) = \; & 0{,}1 \circ F_N(m_1.\text{nome}, m_2.\text{nome}) + \\ & 0{,}45 \circ F_T(m_1.\text{tipo}, m_2.\text{tipo}) + \\ & 0{,}45 \circ F_P(m_1, m_2) \end{aligned} \qquad 4.4$$

onde $F_N(m_1.\text{nome}, m_2.\text{nome})$ é a mesma função utilizada na comparação dos nomes de duas classes (Equação 4.2) e $F_T(m_1.\text{tipo}, m_2.\text{tipo})$ é a comparação entre os tipos retornados pelos métodos. $F_P(m_1, m_2)$ é uma função de similaridade que compara os parâmetros dos dois métodos e é explicada na próxima seção.

IV.2.4.2. Comparação de Parâmetros de Métodos

A comparação de parâmetros de dois métodos segue a mesma estratégia da comparação de atributos das classes. Um conjunto de parâmetros é manipulado do mesmo modo que uma tupla de campos. Portanto, tanto a comparação de parâmetros quanto a de atributos cujo o tipo é uma tupla (tipo *tuple* no O2 ou *struct* em C++), são comparadas utilizando esta função.

A função F_P (P_1, P_2) retorna a pertinência do conjunto de parâmetros (ou campos de uma tupla), P_1, ser um subconjunto dos parâmetros (ou campos) de P_2. Para isso, cada parâmetro de P_1 é comparado com todos os parâmetros de P_2, através da função F_r (Equação 4.5). Esta função é semelhante a função F_a, utilizada para comparar dois atributos. Porém, os pesos utilizados nas duas funções podem ser diferentes, de acordo com a configuração de implementação utilizada.

$$F_r (p_1, p_2) = 0{,}3 \circ F_N (p_1.\text{nome}, p_2.\text{nome}) + 0{,}7 \circ F_T (p_1.\text{tipo}, p_2.\text{tipo}) \quad 4.5$$

Como na comparação de atributos e métodos, para cada parâmetro p_i de P_1, são selecionados os pares (p_i, p_j), $p_j \in P_2$, que apresentam a maior similaridade (calculada pela função F_r) de modo que um mesmo parâmetro de P_2 não seja mapeado por mais de um parâmetro de P_1. Os graus de similaridade dos pares selecionados são somados e divididos pelo número de parâmetros de P_1 e este é o valor retornado pela função F_P.

Assim, se F_P (P_1, P_2) = 1, todos os parâmetros (ou campos) de P_1 possuem pelo menos um parâmetro em P_2 que foi mapeado com grau 1 em relação a eles. Neste caso, F_P (P_2, P_1) $\leq F_P$ (P_1, P_2). Se $P_1 = P_2$, F_P (P_1, P_2) = F_P (P_2, P_1) = 1, ou seja, a função define uma relação reflexiva mas, novamente, a função assim definida não produz uma relação simétrica já que F_P (P_1, P_2) pode ser diferente de F_P (P_2, P_1).

Uma alternativa a esta estratégia de comparação de parâmetros (ou campos) é ser mais rigoroso, exigindo que os conjuntos de parâmetros comparados possuam o mesmo número de parâmetros para que sejam considerados iguais. Outra opção ainda mais restritiva é exigir que a ordem dos parâmetros seja respeitada. Neste caso, cada parâmetro de P_1 seria comparado apenas com o parâmetro de mesma posição de P_2.

Porém, foi adotada a mesma estratégia utilizada na comparação de outros aspectos como atributos e métodos.

IV.3. COMPARAÇÃO DE TIPOS

Assim como os aspectos considerados na comparação de classes, o valor retornado pela comparação de dois tipos, t_1 e t_2, é obtido através de uma função de similaridade, no caso F_T (t_1, t_2). Esta função é utilizada na comparação de atributos, métodos e parâmetros. Um tipo pode ser dividido em quatro categorias: tipos básicos (*integer*, *real*, *string*, etc.), coleções (*list* e *set*), tuplas (tuplas, *structs* e *records*) e classes (cada classe pode ser tratada como sendo um tipo). Assim, a função de similaridade utilizada para comparar dois tipos associa um grau de similaridade entre os dois tipos a partir da Tabela IV.2. Novamente, é importante ressaltar que os valores apresentados na tabela podem ser configurados antes do início do processo de comparação. A comparação de tipos básicos é direta, enquanto a comparação das outras categorias de tipos é realizada com auxílio de outras funções ou da própria função de tipos chamada recursivamente.

Tabela IV.2 – Similaridade entre tipos

Tipos	Integer	string	Real	boolean	char	set	unique set	list	Class	tuple
Integer	1,0		0,5							
String		1,0								
Real	0,4		1,0							
Boolean				1,0						
Char					1,0					
set						$1,0 * F_T$	$0,9 * F_T$	$0,8 * F_T$		
Un. Set						$0,9 * F_T$	$1,0 * F_T$	$0,8 * F_T$		
list						$0,8 * F_T$	$0,8 * F_T$	$1,0 * F_T$		
Class									$1 * F_C$	
tuple										$1 * F_P$

Pela tabela foram atribuídos valores de similaridade para refletir a semelhança entre os tipos. Por exemplo, F_T (*integer*, *real*) retorna 0,5, enquanto F_T (*real*, *integer*) retorna 0,4, ou seja, o tipo inteiro possui uma similaridade maior em relação a um tipo real do que o inverso.

IV.4. COMPARAÇÃO DE COMPORTAMENTO

As seções anteriores apresentaram as comparações, e as respectivas funções de similaridade, para a comparação sintática de classes. Sintática pois os aspectos envolvidos no processo de comparação até então foram avaliados a partir de suas sintaxes, sendo considerado apenas as informações estruturais das classes. Esta seção apresenta a comparação do comportamento das classes, que é efetuada através da função de similaridade F_E (Equação 4.1).

O comportamento de uma classe é avaliado através da execução dos métodos que a compõem. Muita informação semântica de uma classe está embutida em seus métodos (PITOURA *et al.*, 1995), e é este tipo de informação que será captado, sendo importante para identificar melhor a semelhança entre as classes.

A princípio, a comparação do comportamento adota a mesma estratégia seguida ao comparar atributos e métodos: os métodos de uma classe, c_1, são executados, assim como os métodos da segunda classe, c_2. Os resultados obtidos em cada execução são comparados, e os pares de métodos que possuem o maior grau de similaridade entre si são selecionados. Os pares selecionados têm os seus graus somados, obedecendo sempre a restrição de que um mesmo método de c_2 seja mapeado apenas por um método de c_1. O valor total desta soma é dividido pelo número de métodos de c_1 e o resultado retornado pela função F_E para a Equação 4.1.

Porém, como foi visto na Seção II.4.4, a comparação baseada na execução de métodos possui particularidades que impedem a aplicação simples e direta desta estratégia. Em primeiro lugar, para executar um método de uma classe é necessário instanciar um objeto da mesma classe, inicializar os seus atributos com valores - já que o comportamento pode ser influenciado pelo estado do objeto - e finalmente inicializar os parâmetros do método. A inicialização de atributos e parâmetros realizada para a primeira classe deve ser repetida para a segunda classe, de modo que os métodos sejam executados em objetos com o mesmo estado (ainda que não sejam da mesma classe). Outra característica ressaltada na Seção II.4.4 foi o custo de execução de diversos métodos (ou funções), tratando-se, por vezes, de um processo lento e demorado.

Em virtude dessas características, a comparação do comportamento é efetuada após a comparação dos outros aspectos (nomes, atributos e assinatura de métodos). Isto permite utilizar o mapeamento de atributos e métodos das duas classes realizado

pela outras comparações como ponto de partida da comparação do comportamento. O mapeamento entre atributos (e parâmetros também) das duas classes permite inicializar os atributos similares com valores iguais, sem a necessidade de executar um mesmo método várias vezes, de acordo com o possível número de combinações dos atributos (parâmetros). Por outro lado, o mapeamento dos métodos indica a melhor ordem de execução na busca por comportamentos similares. A implementação da execução dos métodos é detalhada no próximo Capítulo.

Levando em consideração as características particulares da execução de métodos citadas acima, a comparação de comportamento é realizada conforme descrito no início da seção, apenas destacando que a comparação de um método com todos os outros é evitada e que os mapeamentos criados pelas outras comparações são utilizados na inicialização e ordem de comparação dos métodos.

Assim, se F_E (c_1, c_2) = 1, todos os métodos de c_1 possuem pelo menos um método em c_2 cujo o valor retornado de sua execução foi idêntico ao valor retornado por um método de c_2. Neste caso, F_E (c_2, c_1) $\leq$ F_E (c_1, c_2), já que c_2 pode ter um número maior de métodos que não teriam similares em c_1. Se $c_1 = c_2$, F_E (c_1, c_2) = F_E (c_2, c_1) = 1, ou seja, a função define uma relação reflexiva mas, novamente, a função assim definida não produz uma relação simétrica já que F_E (c_1, c_2) pode ser diferente de F_E (c_2, c_1). A seguir é explicada a comparação de dois valores retornados pela execução de dois métodos.

IV.4.1.1. Comparação da Execução de dois Métodos

A comparação do comportamento de classes está baseada na comparação da execução de dois métodos. Esta comparação é efetuada simplesmente confrontando os valores retornados pelos métodos. Os valores retornados, assim como os tipos, podem ser divididos em quatro categorias: valores básicos, tuplas, coleções e classes. Deste modo, para efetuar a comparação são adotadas as mesmas estratégias utilizadas na comparação de dois tipos.

A comparação de valores básicos retorna um se os valores forem idênticos, sendo efetuada se necessário, antes da comparação, uma conversão dos tipos comparados. Se os valores forem diferentes, a comparação retorna zero. Dois valores do tipo *real*, são considerados iguais se a diferença entre eles for menor que uma

margem de erro pré-estabelecida. Uma alternativa é também utilizar funções de pertinência para medir a similaridade entre dois valores.

Para comparar tuplas de valores, é utilizada a mesma estratégia da comparação de tuplas (parâmetros), em que o valor de cada campo de uma tupla é testado com os valores da outra. Para a comparação de coleções, verifica-se se os valores da primeira coleção retornada estão presentes na outra coleção. Em ambos os casos, o resultado da comparação é 1 se todos os valores retornados pelo primeiro método forem retornados também pelo segundo método. Caso alguns dos valores não tenham sido retornados, o resultado obtido é um valor entre zero e um, calculado a partir da soma das similaridades entre os valores retornados pelos dois métodos e dividindo-se o total pelo número de valores retornados pelo primeiro método (seja o número de campos na tupla ou número de elementos na coleção).

Alternativamente, pode-se também adotar um critério um pouco mais rigoroso, sendo necessário para que a comparação retorne 1, que todos os valores, seja dos campos da tupla ou dos elementos da coleção, estejam presentes no resultado do segundo método. Caso contrário, o valor retornado é zero. Nesta alternativa, não são retornados valores diferentes de zero ou um, pois ou os valores retornados por um método são iguais aos valores esperados, ou não. Porém, foi adotada a primeira alternativa já que esta segue a estratégia de medir a relação de subtipo entre os elementos comparados utilizada durante toda a técnica de comparação.

A comparação de valores que sejam classes pode ser efetuada a partir da comparação dos seus atributos. Assim, os objetos retornados (de classes iguais ou não) têm os valores dos seus atributos comparados, utilizando os mesmos critérios estabelecidos acima para a comparação de valores retornados pelos métodos. A similaridade entre os dois objetos é obtida, então, do mesmo modo que a similaridade entre tuplas. Porém, esta comparação não foi implementada pois seu cálculo é potencialmente recursivo e difícil de ser tratado, já que a comparação de dois objetos envolve a comparação de seus atributos, que por sua vez podem ser tipos básicos, tuplas, coleções ou mesmo outras classes. Uma explicação mais detalhada sobre este processo de comparação é apresentada no próximo capítulo.

A comparação de métodos que não retornam nenhum valor (métodos que são procedimentos e não funções) pode ser realizada comparando-se as alterações no estado da base de dados, ou seja, alterações nos objetos armazenados na base (objetos

novos criados, alteração nos atributos de alguns objetos, etc.). Uma forma de controlar essas alterações é através do registro de modificações do banco de dados. Os métodos que provocassem as mesmas alterações seriam considerados similares. Novamente, a complexidade deste tipo de comparação impossibilitou a sua implementação e neste caso, os métodos que não retornam nenhum valor foram ignorados durante a comparação do comportamento.

IV.5. ORDENAÇÃO DIFUSA

Conforme descrito no início do Capítulo, a função de similaridade utilizada na comparação de classes F_C definiu uma relação difusa, onde cada elemento da relação é o resultado da comparação de um par de classes. Foi visto que esta relação é reflexiva, $F_C(c_1, c_2) = F_C(c_2, c_1)$ se $c_1 = c_2$, mas não é simétrica, ou seja, se $c_1 \neq c_2$, $F_C(c_1, c_2)$ não necessariamente é igual a $F_C(c_2, c_1)$. Assim, não é possível obter uma relação de equivalência entre as classes. Porém, pode-se aplicar uma técnica de classificação da teoria da lógica difusa para ordenar os elementos da relação. Esta técnica, denominada Ordenação Difusa (*Fuzzy Ordering*), e explicada na Seção III.5.1, pode ser aplicada em relações não simétricas. A partir da ordenação difusa, é possível escolher o par de classes mais similar entre vários pares de classes.

IV.5.1. Aplicação da Ordenação Difusa

A Ordenação Difusa foi aplicada na comparação de classes com o objetivo de selecionar a classe mais similar a uma outra classe. Seja uma classe *e* e um conjunto de classes $C = \{ c_1, c_2, ..., c_n\}$, o objetivo é ordenar as classes de C para escolher a mais similar em relação a *e*. Note que *e* pode inclusive fazer parte do conjunto C.

Para isso, as funções apresentadas na Seção III.5.1 são definidas como sendo a pertinência de escolher a comparação de *e* com c_i ao invés de *e* com c_j. Esta definição está expressa através da Equação 4.6.

$$f_{c_j}(c_i) = \frac{F_C(e, c_i)}{F_C(e, c_i) + F_C(e, c_j) - \cap F_C(e, c_i) F_C(e, c_j)} \quad 4.6$$

onde $F_C(e, c_i)$ é a comparação da classe *e* com a classe c_i. O termo $\cap F_C(e, c_i) F_C(e, c_j)$ exprime quanto a comparação de *e* com c_i possui em comum em relação a

comparação de *e* com c_j. Graficamente, esta interseção pode ser representada como na Figura IV.1.

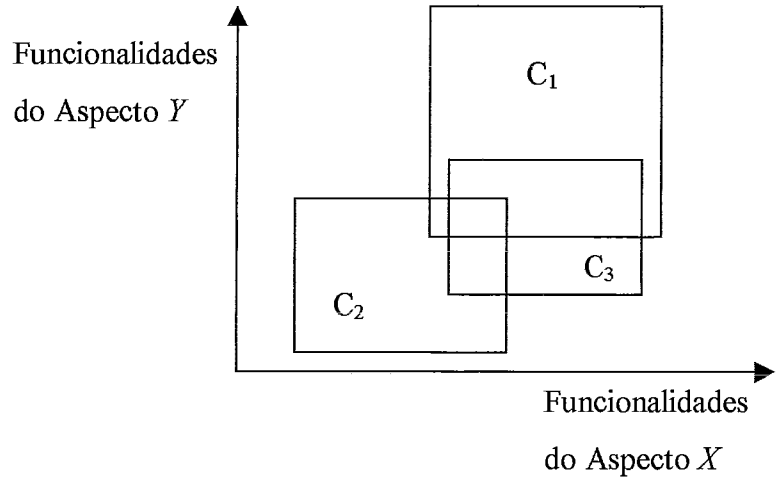


Figura IV.1 – Interseção entre classes por aspectos

A interseção será calculada separadamente para cada um dos aspectos considerados na comparação das classes (nomes, atributos, assinatura dos métodos e comportamento), e depois agregada utilizando os pesos da Tabela IV.1, como mostra a Equação 4.7:

$$\begin{aligned} \cap F_C(e, c_i)\, F_C(e, c_j) = {} & w_N \circ (F_N(e, c_i) \cap F_N(e, c_j)) + \\ & w_A \circ (F_A(e, c_i) \cap F_A(e, c_j)) + \\ & w_M \circ (F_M(e, c_i) \cap F_M(e, c_j)) + \\ & w_E \circ (F_E(e, c_i) \cap F_E(e, c_j)) \end{aligned} \qquad 4.7$$

Para o aspecto relativo aos nomes das classes, a interseção é obtida através da função mínimo (Equação 4.8):

$$F_N(e, c_i) \cap F_N(e, c_j) = \min \{ F_N(e, c_i), F_N(e, c_j) \} \qquad 4.8$$

Como foi visto, a comparação de atributos é realizada pela função $F_A(c_1, c_2)$ (Equação 4.1). A parte em comum entre $F_A(e, c_i)$ e $F_A(e, c_j)$ é calculada como a própria função F_A, ou seja, com base nos atributos de *e*. Assim, sendo A_e o conjunto de atributos de *e* e $a_k \in A_e$, temos:

$$F_A(e, c_i) \cap F_A(e, c_j) = \frac{\sum_{k=1}^{n} \min \{F_a(a_k, s_k), F_a(a_k, t_k)\}}{n} \qquad 4.9$$

onde F_a é a função da Equação 4.3, a_k é o k-ésimo atributo da classe *e*, s_k representa o atributo da classe c_i que possui o maior grau em relação a a_k (veja a seção de comparação de atributos), t_k representa o atributo da classe c_j que possui o maior grau em relação a a_k e *n* é o número de atributos da classe *e*.

Note que o cálculo da parte em comum entre a comparação de dois atributos ficou restrito ao mínimo das pertinências obtidas nas comparações dos atributos, não sendo aplicado o mínimo para cada um dos aspectos considerados na comparação de atributos (no caso, nome e tipo), para depois serem agregados. A alternativa adotada simplifica os cálculos e a possível diferença nos valores obtidos pelas duas formas de calcular a interseção é diluída no resultado final pela existência dos outros aspectos considerados na comparação de classes. De qualquer modo, como todas as classes estão sujeitas as mesmas regras de cálculo, a ordenação final não será destorcida em favor de uma ou outra classe, sendo a imprecisão dos cálculos diluída pela aplicação das técnicas de lógica difusa.

Analogamente, o mesmo raciocínio foi adotado para calcular a parte em comum entre as comparações dos outros aspectos: assinatura e comportamento dos métodos. Assim, temos as Equações 4.10 e 4.11:

$$F_M(e, c_i) \cap F_S(e, c_j) = \frac{\sum_{k=1}^{n} \min \{F_s(m_k, s_k), F_s(m_k, t_k)\}}{n} \qquad 4.10$$

onde, m_k é o k-ésimo método de *e*, s_k é o método de c_i que possui o maior grau (calculado pela função F_s) em relação a m_k, t_k é o método de c_j que possui o maior grau em relação a m_k (calculado pela função F_s) e *n* é o número de métodos de *e*.

$$F_E(e, c_i) \cap F_E(e, c_j) = \frac{\sum_{k=1}^{n} \min \{F_e(m_k, s_k), F_e(m_k, t_k)\}}{n} \qquad 4.11$$

onde, m_k é o k-ésimo método de e, s_k é o método de c_i cuja a execução possui o maior grau (representado pela função F_e) em relação a m_k, t_k é o método de c_j cuja a execução possui o maior grau em relação a m_k e n é o número de métodos de e.

Com o termo $\cap F_C(e, c_i) F_C(e, c_j)$ da Equação 4.6 definido através das Equações 4.8, 4.9, 4.10 e 4.11, a função de relatividade pode ser escrita a partir das funções $fc_j(c_i)$ e $fc_i(c_j)$:

$$f(c_i \mid c_j) = \frac{f c_j(c_i)}{\max(f c_j(c_i), f c_i(c_j))} \qquad 4.12$$

Generalizando a equação para o conjunto de classes C, como mostrado na Equação 3.14, temos:

$$f(c_i \mid C) = \min \{f(c_i \mid c_1), f(c_i \mid c_2), \ldots, f(c_i \mid c_n)\} \qquad 4.13$$

Com esses valores é construída a matriz de comparação C e a partir dela pode-se obter a classe do conjunto C mais similar a classe e. Um exemplo da aplicação desta técnica será visto no Capítulo VI.

V. Arquitetura de Implementação

Este capítulo descreve a arquitetura criada para implementar o processo de comparação de classes mostrado no Capítulo IV, assim como as principais telas do protótipo desenvolvido. O objetivo da arquitetura é organizar a implementação de modo que as estratégias de comparação escolhidas para cada aspecto possam ser facilmente substituídas por outras técnicas, garantindo a flexibilidade e permitindo a expansão do método.

Para alcançar este objetivo, foi elaborado um modelo de classes de modo que a substituição de uma estratégia de comparação envolva apenas a especialização da classe apropriada. Assim, se for desejável substituir a comparação de nomes explicada na Seção IV.2.2, por exemplo, basta especializar a classe que implementa esta comparação, criando uma classe que compare nomes utilizando algum outro algoritmo, como, por exemplo, um dicionário de sinônimos. A arquitetura proposta é genérica e pode ser desenvolvida em qualquer ambiente. Nesta tese, a implementação foi realizada utilizando o banco de dados O2 (LÉCLUSE e RICHARD, 1989) e C++.

O O2 foi escolhido principalmente por ter características técnicas que permitiram a implementação da comparação por comportamento. A primeira dessas características é a existência de um conjunto de classes para representar o meta-esquema das classes existentes em um banco de dados. A consulta aos metadados é essencial em qualquer processo de comparação de classes. Assim, as classes que serão comparadas devem estar definidas e implementadas no O2, em um ou mais esquemas, o que possibilita a consulta aos seus metadados e a execução de seus métodos quando necessário.

A segunda e decisiva característica é a possibilidade de criar uma classe ou um método em tempo de execução. Este recurso foi utilizado para comparar a execução dos métodos. Outra característica importante é a capacidade de executar os métodos conforme a necessidade, a qualquer momento e sem ordem definida. Como a ordem e as condições de execução são determinados somente no momento da execução, durante o processo de comparação, esta característica torna-se importante pois permite a execução do teste de comportamento sem a necessidade de recompilar o código fonte original. A presença desta funcionalidade foi uma fonte de motivação para o

início deste trabalho e somente com a utilização dessas características foi possível desenvolver a comparação da execução dos métodos.

Outra vantagem na utilização do O2 foi a facilidade de armazenar as informações necessárias durante o processo de comparação, já que o O2 é um banco de dados orientado a objetos com suporte a consultas. A persistência dessas informações (principalmente do mapeamento de atributos e métodos entre as diversas classes comparadas) facilitou a implementação e permitiu que o processo de comparação fosse executado em partes, de acordo com a disponibilidade e desejo do usuário.

O C++ foi utilizado na implementação de algumas classes que necessitavam de uma maior flexibilidade e que não poderiam estar vinculadas a um esquema e a uma base do O2 pois as classes e os métodos implementados no O2 não podem acessar duas bases diferentes em esquemas diferentes.

As próximas Seções apresentam o modelo de classes adotado na arquitetura e os aspectos relevantes da implementação. Também são apresentadas as principais telas do protótipo que implementa a estratégia de comparação proposta, com o objetivo de ilustrar melhor o funcionamento do processo de comparação.

V.1. Modelo de Classes

O modelo de classes adotado tem como propósito principal flexibilizar a implementação da técnica de comparação, permitindo sua expansão com outras estratégias de comparação.

Assim, a implementação foi construída sobre um modelo de classes de modo que as comparações apresentadas no Capítulo anterior foram isoladas em classes distintas, apenas com uma interface bem definida para as outras classes. Através desta implementação foi alcançada a flexibilidade (possibilidade de substituição dos algoritmos utilizados) e a expansibilidade (acréscimo de novos aspectos ao processo de comparação).

Para permitir uma melhor visualização, o modelo será apresentado em visões específicas para cada aspecto presente na comparação, utilizando a notação UML. A Figura V.1 mostra o modelo de classes principal que é explicado na Seção V.2. A próxima subseção explica as classes criadas para a manipulação das informações utilizadas durante o processo de comparação e as classes abstratas que oferecem

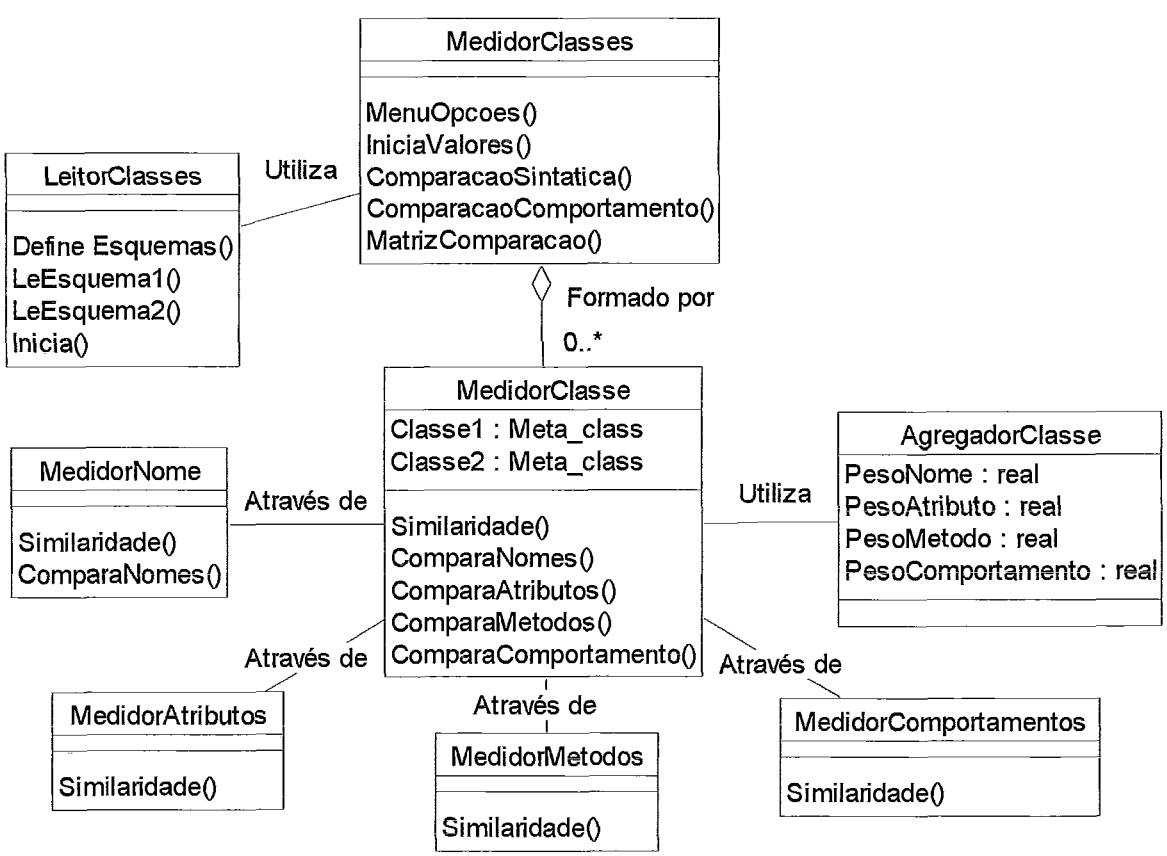
suporte às principais funções deste processo. Especializações dessas classes são utilizadas em conjunto na implementação das comparações de cada aspecto. Também são apresentadas as classes do O2 que armazenam os metadados utilizados durante a comparação. Nas outras Seções os modelos para a comparação de cada aspecto são apresentados.

Figura V.1 – Modelo principal da arquitetura de comparação de classes

V.1.1. Classe Medidor

As especializações desta classe são responsáveis pelas diversas comparações realizadas durante o processo de comparação. Essas comparações podem ser entre dois elementos (dois nomes, dois atributos, etc.) ou entre dois conjuntos de elementos (atributos de duas classes, assinaturas de métodos de duas classes, etc.). As especializações da classe Medidor medem a similaridade entre dois elementos ou entre um conjunto de elementos comparados, implementando as funções definidas no Capítulo IV.

Um objeto de uma subclasse de Medidor calcula a similaridade entre os elementos comparados e retorna este valor através de um método específico (método



Similaridade), único para todas especializações criadas. Os elementos que serão comparados são informados através de métodos próprios de cada especialização (métodos para inicialização), já que as informações necessárias à comparação dependem do tipo de elemento envolvido. Para efetuar este cálculo, o objeto pode utilizar outros objetos de medição, além de um objeto da classe Agregador (explicada adiante) associado a ele que encapsula a fórmula de cálculo e os pesos utilizados.

Algumas especializações da classe Medidor são utilizadas também no mapeamento entre os elementos. Isto ocorre paras as classes MedidorAtributos, MedidorMetodos e MedidorParametros que associam a cada par de atributos, métodos e parâmetros, respectivamente, um grau de similaridade. Este mapeamento é utilizado na comparação do comportamento e na construção da matriz de comparação ao final do processo.

V.1.2. Agregador

As subclasses de Agregador implementam a fórmula de cálculo para agregar de dois ou mais valores difusos. Conforme o Capítulo IV, as agregações difusas presentes na técnica proposta são de dois tipos: agregação através de pesos e uma agregação baseada na média de um conjunto de valores difusos.

As subclasses de agregação através de pesos têm como objetivo encapsular os pesos utilizados, facilitando sua alteração. Assim, para alterar os pesos utilizados no cálculo de uma similaridade por um objeto da classe Medidor, basta substituir o objeto de agregação difusa por outro objeto de outra subclasse de Agregador ou então alterar os valores dos pesos armazenados no objeto. Assim, basta executar a comparação novamente para obter os novos valores de similaridade calculados de acordo com os novos pesos, não sendo necessário alterar o restante do processo. Os pesos e configurações utilizados na implementação da comparação de classes aparecem no Anexo I.

Do mesmo modo, as subclasses de Agregador para conjunto de elementos permitem alterar a fórmula para agregar os diversos valores sem a necessidade de efetuar alterações no resto do processo. Isto garante a flexibilidade da técnica de comparação e de alteração dos critérios e estratégias adotadas.

V.1.3. Classes de Metadados

O processo de comparação inicia-se através da recepção das informações dos elementos (classes, atributos, etc.) a serem comparados e, a partir dessas informações, é calculada a similaridade resultante da comparação, assim como um conjunto de informações sobre o mapeamento entre os elementos. Essas duas últimas tarefas são executadas através das classes Medidor e Agregador. A recepção das informações depende da definição de um conjunto de classes que modelem os metadados dos elementos a serem comparados e permita a sua manipulação.

Como a técnica proposta foi desenvolvida tendo o O2 como repositório das classes, foi utilizado o conjunto de classes do meta-esquema que o O2 oferece para obter as informações necessárias à comparação, não havendo necessidade de criar um conjunto de classes auxiliares para tal. As classes Medidor implementadas manipulam essas informações. Portanto, toda informação necessária ao processo de comparação é obtida a partir do meta-esquema do O2, utilizando as suas próprias classes. Estas informações estão disponíveis no meta-esquema do O2 através de um conjunto de classes que representam os metadados de um esquema. A Figura V.2 mostra a hierarquia das classes do O2 que implementam os metadados.

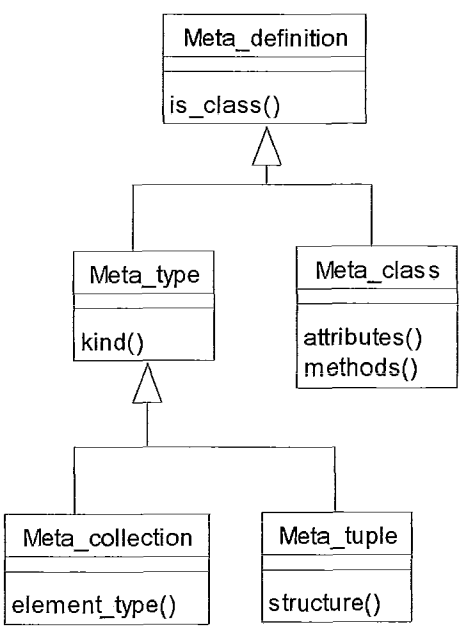


Figura V.2 – Estrutura de classes do meta-esquema do O2

Os métodos *attributes* e *methods* da classe Meta_class retornam uma lista de tuplas contendo o nome, visibilidade (se é público ou privado) e o tipo de cada

atributo e método da classe, respectivamente. Para os métodos, a tupla retornada ainda inclui informações sobre os parâmetros (nome e tipo). O mesmo ocorre para o método *structure* da classe Meta_tuple.

Se a implementação ocorresse em uma ferramenta que não o O2, seria necessário criar um conjunto de classes que armazenasse as informações sobre os esquemas a serem comparados. De qualquer modo, essas informações podem ser modeladas de forma semelhante às classes do meta-esquema do O2, sem prejuízo a estratégia proposta.

V.2. COMPARAÇÃO DE CLASSES

A identificação de classes similares é o objetivo final do processo de comparação e portanto o modelo principal da arquitetura de implementação. Todas as classes estão direta ou indiretamente relacionadas a este modelo, que aparece na Figura V.1.

A classe MedidorClasses é a responsável pela comparação de dois conjuntos de classes. Entre as funções desempenhadas por ela estão a leitura das informações sobre as classes a serem armazenadas (através de consultas ao meta-esquema do O2), controle do fluxo processo de comparação e a construção da matriz de comparação para as classes. Esta classe foi implementada parte no C++ e parte no O2 (utilizado principalmente para construir a interface) e ao contrário das outras subclasses de Medidor, ela não calcula nenhum grau de similaridade entre os elementos comparados (no caso, dois esquemas). Mas isto seria possível, sendo necessário definir uma fórmula de calcular esta similaridade.

Todo o processo de comparação é controlado por um objeto desta classe, denominado Controlador. No protótipo desenvolvido, esta classe possui um método que apresenta um menu com as opções disponíveis durante o processo (Figura V.3), como, por exemplo, a leitura das classes a serem comparadas, a comparação sintática e a comparação do comportamento. O objeto Controlador é armazenado de forma persistente no O2, salvando as informações coletadas durante o processo de comparação.

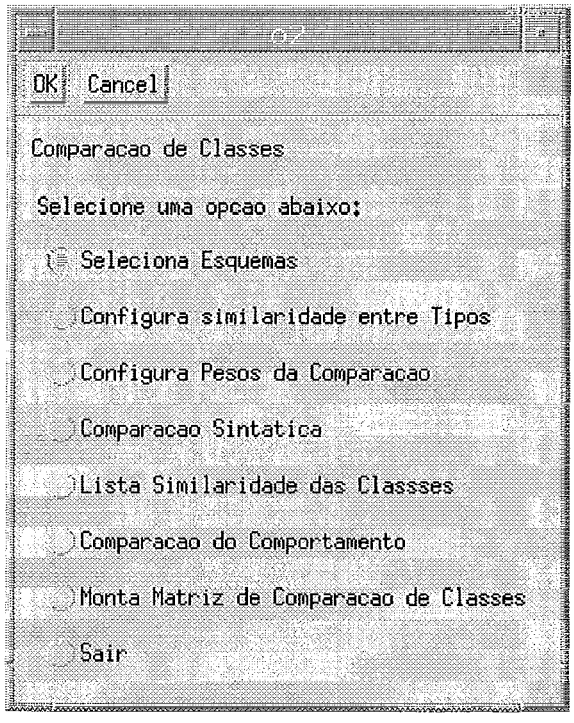


Figura V.3 – Menu principal do protótipo

A leitura das classes a serem comparadas foi implementada em uma classe separada, escrita em C++, flexibilizando o processo de coleta das informações sem alterar a estratégia de comparação como um todo. A classe TLeitorClasses é acionada pelo objeto Controlador e realiza uma consulta ao meta-esquema do O2 para ler as definições das classes de dois esquemas (diferentes ou não). A Figura V.4 mostra a interface do protótipo criada para selecionar os esquemas que serão comparados.

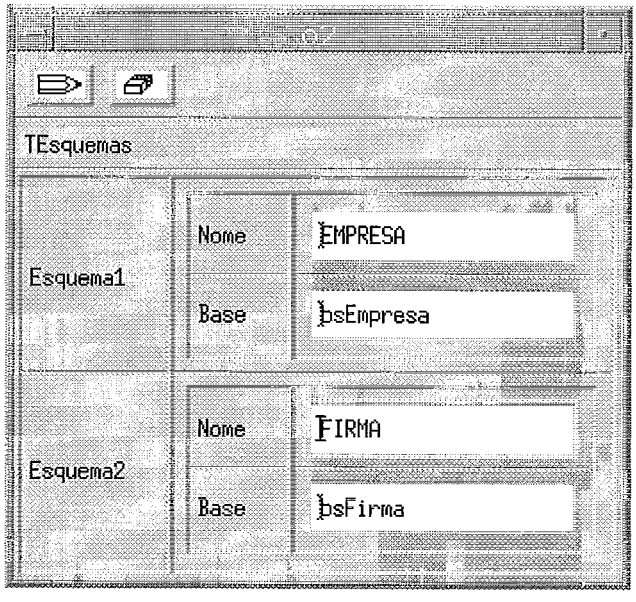


Figura V.4 – Tela de seleção de esquemas a serem comparados

A partir do menu exibido pelo objeto Controlador é possível alterar os pesos utilizados nas funções de similaridade de atributos, métodos e parâmetros. Para isso, são executados os métodos das subclasses de Agregador que implementam a agregação dessas funções. A Figura V.5 mostra o diálogo do protótipo desenvolvido que permite ao usuário definir os valores dos pesos utilizados na comparação das classes. O Anexo I apresenta todos os pesos utilizados nesta implementação.

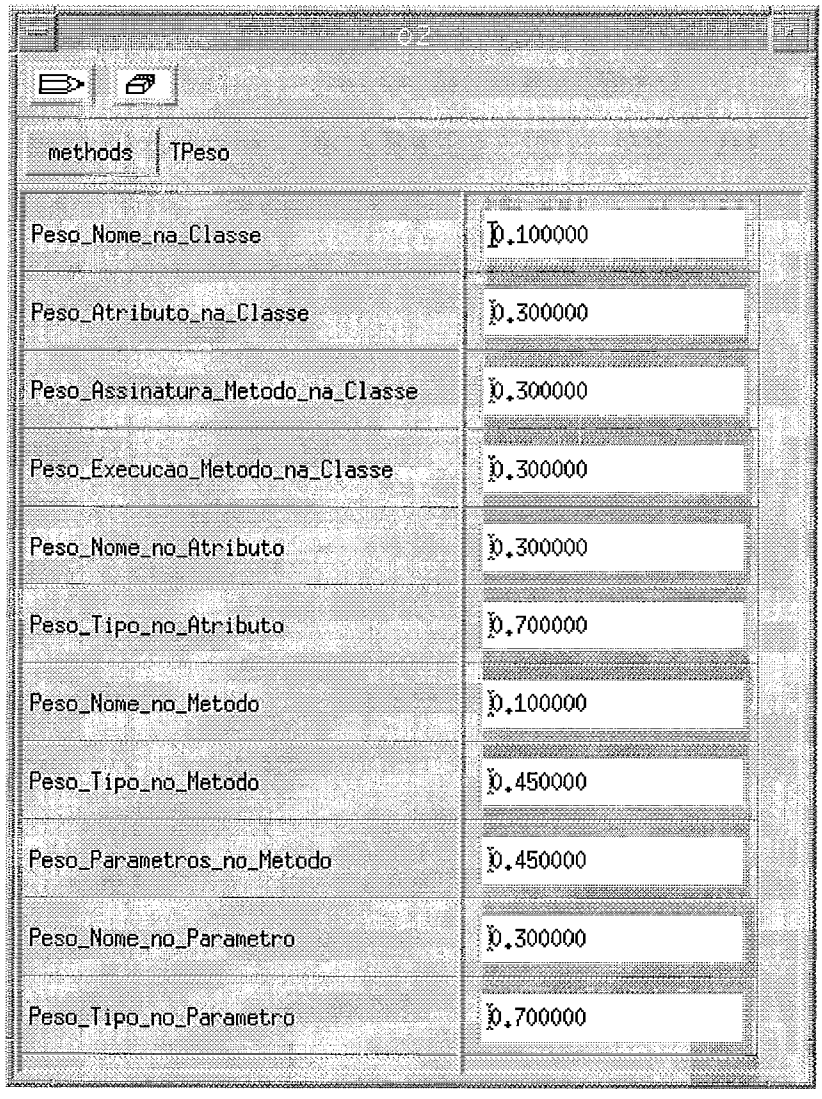


Figura V.5 – Diálogo para a configuração dos pesos utilizados na comparação

Após a determinação e leitura das classes dos esquemas alvos da comparação, o objeto Controlador inicializa os objetos da classe MedidorClasse para cada par de classes lida do meta-esquema do O2. A inicialização ocorre através de um método existente na classe que MedidorClasse que recebe como parâmetro dois objetos da classe Meta_class do meta-esquema do O2 (uma classe de cada esquema).

Após esta inicialização, o objeto Controlador aciona a comparação sintática das classes. Como foi visto no Capítulo IV, a comparação sintática é formada pelas comparações dos nomes, atributos e assinaturas dos métodos. Durante a comparação sintática, são criados os mapeamentos preliminares entre as classes, atributos e métodos. Esses mapeamentos são utilizados em seguida durante a comparação do comportamento.

A classe MedidorClasse executa a comparação das classes utilizando a estratégia definida no Capítulo IV e expressa na Equação 4.1. Assim como a equação, MedidorClasse efetua a comparação com o auxílio de quatro outras classes, cada uma delas implementando uma das funções da Equação 4.1. O objeto da classe MedidorNome realiza a comparação de dois nomes (recebidos na inicialização) e retorna um valor de similaridade entre eles conforme a Equação 4.2. As outras três classes são explicadas nas próximas Seções.

A classe MedidorClasse implementa também um controle das classes que já foram comparadas ou que estão sendo comparadas. Assim, quando um objeto MedidorClasse é acionado para retornar a similaridade de duas classes, ele verifica se já efetuou a comparação sintática. Se a comparação já foi concluída, ele retorna o valor imediatamente, senão verifica se ele já foi acionado antes mas ainda não terminou a comparação. Esta situação ocorre se duas ou mais classes comparadas possuam relacionamentos que formem um ciclo, ou uma das classes possua um auto-relacionamento. Nestes casos, o objeto retorna um valor baseado na comparação que a classe Meta_class do meta-esquema do O2 implementa, através do método *compare*. Esta comparação retorna três valores diferentes para a comparação de duas classes (c_1 e c_2): COMPATIBLE, se c_2 é igual a c_1 ou se herda de c_1; SUBCLASS se c_1 é subclasse de c_2; e NOT_COMPARABLE para os outros casos. Estes valores são transformados em graus de similaridade (Tabela V.1) e retornados pelo método *Similaridade*, quebrando o ciclo ou a recursividade e evitando um ciclo infinito.

Tabela V.1 – Graus de Similaridade para comparações do O2

Valor retornado por c_1.compare(c_2)	Grau de similaridade retornado
COMPATIBLE	0,5
SUBCLASS	1,0
NOT_COMPARABLE	0,0

Os objetos de MedidorClasse definem então um mapeamento e são armazenados de forma persistente em um *name* do O2, de onde são recuperados por outras classes do modelo, como por exemplo MedidorClasses (para montar a matriz de comparação) ou MedidorComportamento (para ler as informações necessárias a execução do teste de comportamento).

V.3. COMPARAÇÃO DE ATRIBUTOS

A classe MedidorAtributos, que aparece na Figura V.1, implementa a função F_A da Equação 4.1, realizando a comparação entre os atributos de duas classes. Para isso, esta classe utiliza outros objetos de medição e um objeto de uma subclasse de Agregador para agregar os diversos valores recebidos. O modelo que mostra as classes envolvidas na comparação de atributos é apresentado na Figura V.6.

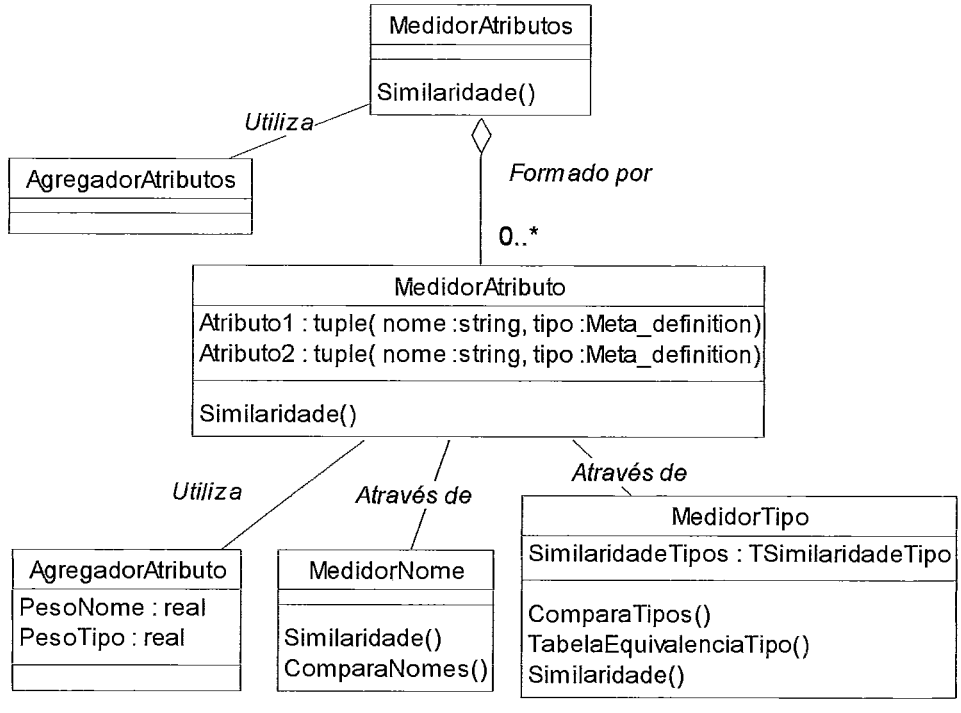


Figura V.6 – Modelo de classes para a comparação de atributos

O medidor de conjunto de atributos é formado por vários medidores de atributo e, com o auxílio de um objeto agregador da classe AgregadorAtributos, determina a similaridade entre os conjunto comparados. A classe AgregadorAtributos recebe vários números difusos e calcula a similaridade final realizando a média

aritmética comum. Os valores passados ao agregador foram selecionados pelo medidor de conjunto de atributos, seguindo a estratégia definida na Seção IV.2.3.

Estes valores são obtidos dos objetos da classe MedidorAtributo, responsável pela comparação de um par de atributos e pela definição de um mapeamento entre eles. Estes objetos são mantidos persistentes por navegação, a partir dos objetos MedidorClasse armazenados, permitindo a consulta aos mapeamentos depois. Para efetuar a comparação, esta classe segue a estratégia definida no Capítulo IV e implementa a função F_a (Equação 4.3), utilizando outros objetos: um agregador que encapsula os pesos definidos para os aspectos dos atributos (nomes e tipos), um medidor de nomes (o mesmo utilizado na comparação entre duas classes) e um medidor de tipo, responsável pela implementação da função F_T, que retorna a similaridade entre dois tipos.

A classe MedidorTipo armazena a Tabela IV.2 através do atributo *SimilaridadeTipos* e pode acionar outros objetos medidores se necessário. Por exemplo, para comparar dois tipos que sejam classes, são acionados os objetos MedidorClasses apropriados e para comparar dois tipos que sejam coleções é acionado outro objeto MedidorTipo para comparar o tipo dos elementos das coleções.

V.4. Comparação da Assinatura de Métodos

A comparação da assinatura de métodos é realizada pela classe MedidorMetodos de acordo com a estratégia explicada na Seção IV.2.4. Para tal, esta classe utiliza as classes MedidorMetodo e AgregadorMetodos como mostra o modelo da Figura V.7. A classe AgregadorMetodos calcula a similaridade final através dos valores retornados pelos objetos de MedidorMetodo e selecionados por MedidorMetodos, retornando a média aritmética.

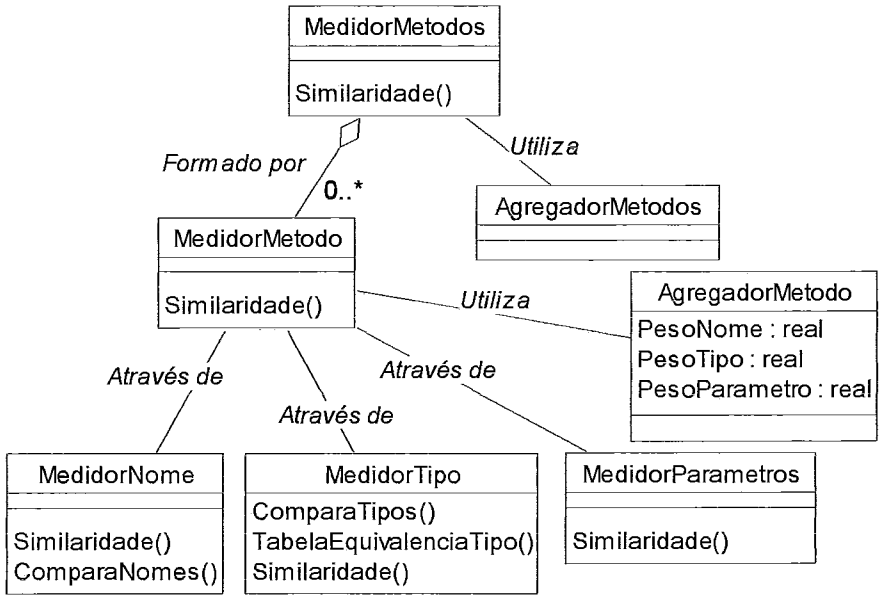


Figura V.7 – Modelo de classes para a comparação de assinaturas de métodos

Os objetos da classe MedidorMetodo calculam a similaridade entre duas assinaturas segundo a estratégia da seção IV.2.4.1. Para isso, utilizam as classes MedidorNome, MedidorTipo, MedidorParametros e AgregadorMetodo. As duas primeiras também foram utilizadas na comparação de atributos. O objeto da classe AgregadorMetodo recebe os valores dos outros objetos e os agrega utilizando os pesos, conforme a função F_s (Equação 4.4).

A classe MedidorParametros calcula a similaridade para dois conjuntos de parâmetros conforme descrito na Seção IV.2.4.2. As classes que implementam a comparação de parâmetros também são utilizadas na comparação de tuplas. Esta comparação é detalhada na próxima Seção.

Novamente, os objetos da classe MedidorMetodo são mantidos persistentes por navegação, a partir dos objetos da classe MedidorClasse e serão utilizados na comparação do comportamento.

V.5. COMPARAÇÃO DE PARÂMETROS

Para implementar a função F_P (Seção IV.2.4.2), responsável pela comparação de parâmetros, foi criada a classe MedidorParametros que compara dois conjuntos de parâmetros ou duas tuplas (manipuladas como dois conjuntos de campos). Esta classe utiliza as classes AgregadorParametros e MedidorParametro para efetuar o cálculo de F_P. A classe AgregadorParametros calcula a similaridade final, a partir dos valores

retornados pelos objetos MedidorParametro e selecionados pelo objeto MedidorParametros, através da média aritmética.

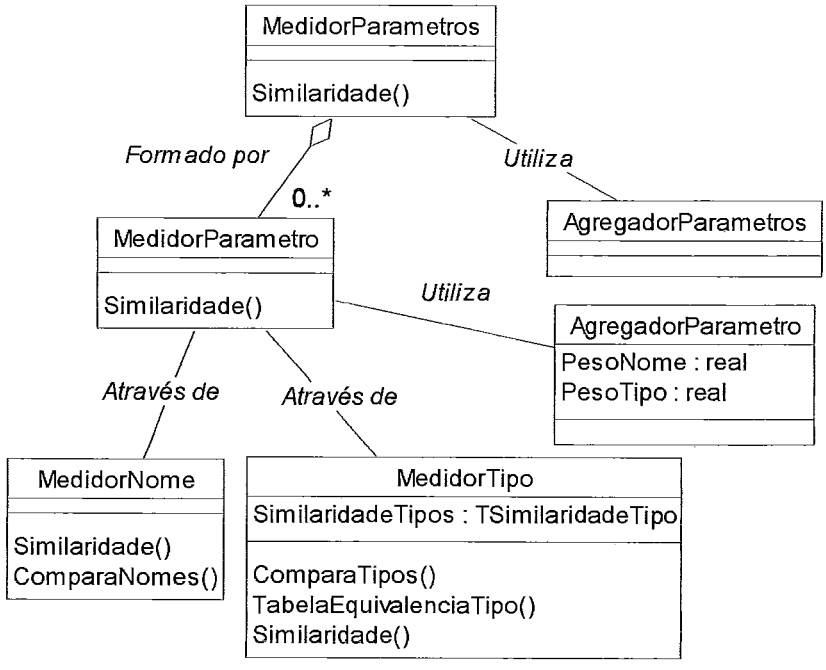


Figura V.8 – Modelo de classes para a comparação de parâmetros

Os objetos da classe MedidorParametro calculam a similaridade entre dois parâmetros através de objetos das classes MedidorNome e MedidorTipo, seguindo a fórmula da função F_r na Equação 4.5. Os valores desses objetos são agregados pela classe AgregadorParametro que utiliza pesos para somar os dois valores.

V.6. COMPARAÇÃO DE COMPORTAMENTO

A comparação de comportamento de duas classes foi definida na Seção IV.4, sendo determinada pela função F_E. O modelo de classes criado para executar esta tarefa aparece na Figura V.9. Nele, observa-se a classe MedidorComportamentos que calcula o valor da função F_E para duas classes, c_1 e c_2, selecionando os valores retornados pelos objetos da classe MedidorComportamento. A agregação é efetuada por um objeto da classe AgregadorComportamentos que calcula a média aritmética dos valores selecionados. Além de realizar o cálculo da similaridade de comportamento entre dois conjuntos de métodos de duas classes, a classe MedidorComportamentos controla o fluxo de execução. Como foi ressaltado no Capítulo IV, a comparação de

comportamento possui algumas particularidades que a tornam um pouco mais complexa. Alguns desses aspectos acabaram por determinar adaptações, restrições e limitações no processo de comparação.

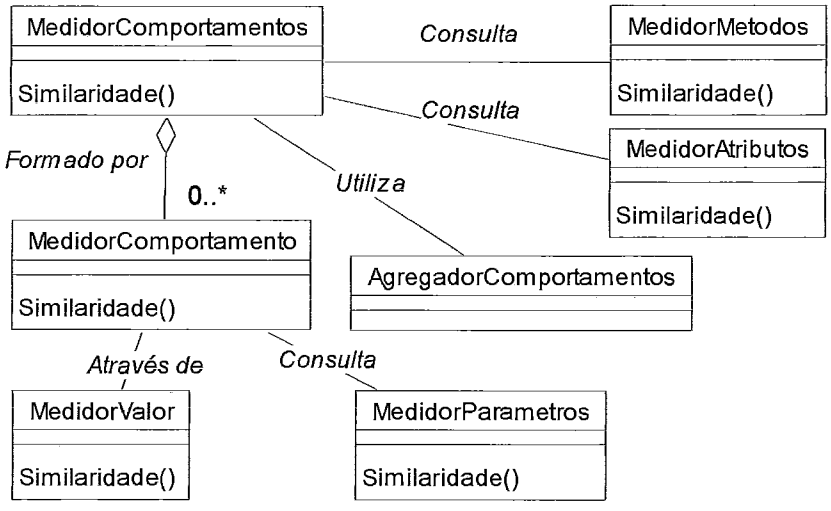


Figura V.9 – Modelo de classes para a comparação do comportamento

A comparação de comportamento através da execução dos métodos de objetos foi baseada no trabalho de PIERCE e PODGURSKY (1992) que propõe uma busca de funções para reutilização, tendo como principal critério de seleção os valores retornados pelas execuções das funções (Seção II.4.4). Os procedimentos de comparação utilizados por PIERCE e PODGURSKY (1992) foram seguidos aqui também.

A primeira etapa do processo de busca descrito é selecionar as funções cujas assinaturas são compatíveis com a assinatura procurada. Esta etapa foi implementada nesta tese através da Comparação de Assinaturas de Métodos (Seção V.4). Portanto, as informações coletadas na comparação das assinaturas dos métodos são utilizadas aqui para limitar os métodos que serão executados na comparação de duas classes. Essa interação aparece no modelo através do relacionamento "Consulta", entre as classes MedidorComportamentos e MedidorMetodos.

A segunda etapa consiste da inicialização dos parâmetros da função que será executada. Na comparação de classes, a inicialização dos parâmetros foi estendida para incluir também a inicialização dos atributos, já que os valores dos atributos de um objeto podem determinar o seu comportamento (resultado dos métodos). A

inicialização dos atributos (implementada pela classe MedidorComportamentos) ocorre antes mesmo da inicialização dos parâmetros (implementada pela classe MedidorComportamento). Para isso, são utilizados os mapeamentos entre os atributos construídos pela Comparação de Atributos (Seção IV.2.3).

A inicialização dos objetos que terão seus métodos executados é complexa já que uma classe pode conter atributos que são outros objetos, inclusive com a ocorrência de referência circular ou recursividade. Outro aspecto importante consiste na escolha dos valores que devem ser utilizados na inicialização dos atributos. Solicitar estes valores ao usuário que está realizando a comparação é possível, porém praticamente inviável se o número de atributos e classes comparadas for grande. A solução proposta em PIERCE e PODGURSKY (1992), de inicializar os parâmetros com valores aleatórios, pode ser imprópria pois não reflete os valores reais esperados para os atributos dos objetos.

Para contornar este problema, adotou-se a seguinte solução: se desejado, é possível inicializar os atributos do objeto comparado manualmente em tempo de execução, mas é oferecida a possibilidade de selecionar um objeto de uma base de dados do O2 para servir como amostra de entrada para a comparação. Aproveitando o recurso de o O2 ser um banco de dados e poder armazenar objetos em coleções (*names*), pode-se criar uma base de dados para cada esquema envolvido na comparação, com diversos objetos das diferentes classes instanciados e armazenados em coleções. No momento de comparar a execução dos métodos de duas classes, o objeto da classe MedidorComportamentos carrega um objeto da classe c_1 de uma coleção apropriada e o utiliza na execução. Depois, os valores dos seus atributos são utilizados para inicializar os atributos apropriados do objeto da classe c_2.

A Figura V.10 mostra a tela do protótipo que permite a escolha de um repositório de objetos para a seleção do objeto que será utilizado no teste. Nesta tela o usuário deve indicar o nome do repositório do O2 (*name* do O2) e o tipo do repositório (se é uma coleção do tipo conjunto ou do tipo lista ordenada. Em seguida, o usuário pode selecionar o objeto a partir de uma lista de objetos obtida no repositório (Figura V.11). O objeto assim selecionado é utilizado na comparação com todas as classes do outro esquema.

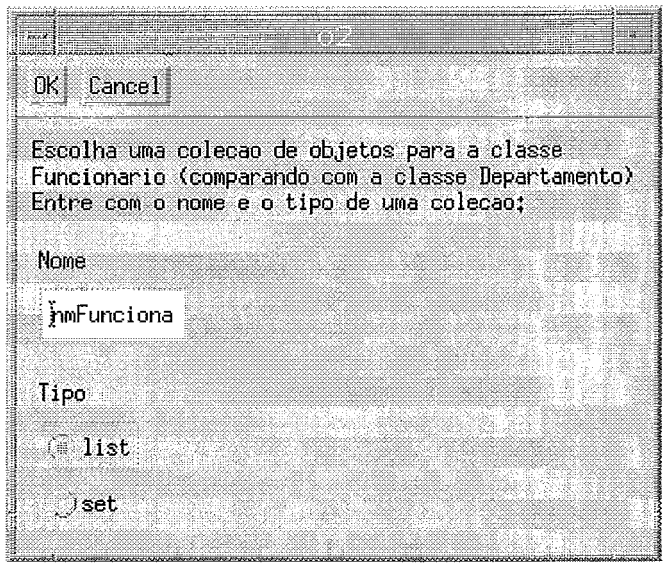


Figura V.10 – Tela para escolha de um repositório de objetos

Esta solução permite executar o método do objeto de c_1 com atributos preenchidos com valores apropriados, incluindo os que representam relacionamentos com outros objetos. A base de dados criada para a realização da comparação constitui um contexto, ou ambiente, adequado para a execução dos métodos. Os resultados apresentados no próximo Capítulo foram obtidos utilizando-se contextos adequados, ou seja, bases de dados preenchidas para a realização do teste.

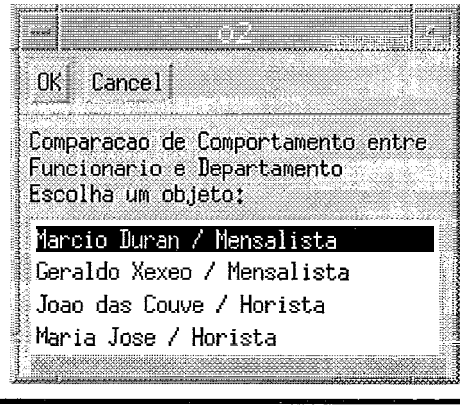


Figura V.11 – Seleção de um objeto do repositório

A inicialização dos parâmetros dos métodos é o próximo passo do processo de comparação, porém ela não ocorre de forma automática, sendo solicitado ao usuário que está acompanhando a comparação a entrada dos valores para os parâmetros do método da primeira classe. Esses valores são atribuídos aos parâmetros dos dois métodos executados (tanto da classe c_1 quanto da c_2). Cada parâmetro restante da classe c_2, se for o caso, é inicializado com o valor padrão definido pelo O2

(geralmente, zero ou uma cadeia de caracteres vazio, por exemplo). A Figura V.12 apresenta o diálogo para preenchimento do parâmetro.

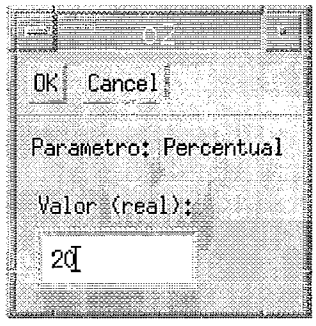


Figura V.12 – Diálogo para preenchimento de parâmetro

Devido a complexidade ao inicializar os parâmetros e atributos, a comparação de comportamento implementada está executando apenas os métodos cujos tipos dos parâmetros são básicos (*integer*, *real*, *boolean*, *char*, *string*). Do mesmo modo, os atributos do objeto da classe c_2 somente são inicializados se seus tipos forem do tipo básico. Apesar desta limitação prejudicar a eficiência da técnica proposta e limitar o seu uso, seu impacto não é impeditivo pois tipos complexos como tipo *class* e os tipos coleções (*list*, *set*), estão, em geral, associados a existência de relacionamentos entre as classes. A semântica destes relacionamentos é capturada através da comparação destes atributos.

A terceira e última etapa do procedimento de comparação de PIERCE e PODGURSKY (1992) consiste na execução das funções que foram selecionadas na primeira etapa e inicializadas na segunda. Esta etapa é implementada pela classe MedidorComportamento, cuja a função é comparar a execução de dois métodos. Porém, antes de executar os dois métodos, esta classe é responsável pela inicialização dos parâmetros conforme descrito acima.

Estando os parâmetros inicializados, o método do objeto selecionado da classe c_1 é executado e o valor de retorno armazenado. Em seguida, o método do objeto da classe c_2 tem seus parâmetros inicializados com os mesmos valores utilizados para a classe c_1. Valores iguais são atribuídos aos parâmetros identificados como mais similares pela comparação de parâmetros (Seção IV.2.4.2). Este mapeamento de parâmetros é consultado através da classe MedidorParametros. Após a inicialização, o método do objeto da classe c_2 é executado e o valor de retorno é comparado com o

valor retornado por c_1. Para realizar a comparação desses valores foi criada uma outra classe, denominada MedidorValor, capaz de comparar dois valores, utilizando a estratégia descrita na Seção IV.4.1.1. O valor retornado por esta classe é a similaridade entre a execução dos dois métodos.

Em PIERCE e PODGURSKY (1992), as funções que eram alvos da pesquisa são executadas diversas vezes, para várias amostras de entrada. Devido a maior complexidade envolvida na comparação de métodos de classes (inicialização de atributos além dos parâmetros), os métodos foram executados apenas para uma amostra de entrada, cujos valores foram retirados de um objeto armazenado na base de dados de um esquema. Além disso, os atributos e parâmetros do objeto da classe c_2, foram inicializados também uma única vez, ou seja, não foi realizada uma combinação da atribuição dos possíveis valores para os diversos atributos e parâmetros de c_2. Tal combinação resultaria em um grande número de amostras de entrada e, consequentemente, em várias execuções do método da classe c_2. Se fossem utilizadas várias amostras de entrada, a comparação do comportamento seria demasiadamente demorada. Apesar disso, a eficiência da técnica de comparação não será prejudicada pois a comparação do comportamento corresponde a um dos aspectos que contribuem para o resultado final, não sendo o único aspecto como em PIERCE e PODGURSKY (1992). Também cabe destacar que a comparação de comportamento é muito precisa (Seção II.4.4.2), mesmo para amostras de entrada pequenas. Além disso, os valores utilizados na inicialização dos atributos e parâmetros fazem parte do domínio de valores esperados para os objetos das classes, já que foram retirados de objetos existentes. Em PIERCE e PODGURSKY (1992), os valores utilizados foram gerados aleatoriamente, possibilitando a existência de amostras de entrada impróprias para execução (valores que, ao serem utilizados na inicialização, fazem com que as funções retornem valores de erro). E mesmo com a utilização de valores aleatórios por PIERCE e PODGURSKY (1992), o tamanho médio para a amostra de entrada capaz de distinguir componentes ficou apenas em 2,6, ou seja, foram necessárias apenas 2,6 execuções para diferenciar as funções que satisfazem a busca.

O objetivo deste trabalho é captar a semântica do comportamento na identificação de classes similares. O impacto da participação da comparação do comportamento na técnica proposta é analisado no Capítulo VI, onde são apresentados exemplos da aplicação da técnica na comparação de dois esquemas.

VI. Análise de Resultados

VI.1. Introdução

Este capítulo apresenta alguns exemplos da aplicação da técnica de comparação proposta, mostrando os resultados obtidos. A técnica foi utilizada em duas situações distintas. Na primeira situação, ela foi aplicada sobre dois esquemas iguais. Neste caso, pode-se verificar o resultado da comparação de uma classe com ela mesma. Na segunda situação, um dos esquemas comparados no exemplo anterior é alterado, com modificações nos nomes das classes, atributos e métodos, estrutura (tipos) de atributos e métodos e de comportamento. Neste caso, verifica-se o reflexo das alterações e o funcionamento da técnica de comparação proposta.

A primeira Seção apresenta o modelo de classes utilizado como esquema de teste na comparação. A Seção seguinte mostra o resultado da comparação entre esquemas iguais e as outras Seções analisam o efeito das alterações no esquema de teste e seus reflexos na comparação.

VI.2. Esquema para Exemplo

A Figura VI.1 mostra o esquema utilizado como exemplo para a aplicação da técnica de comparação proposta. Além das classes apresentadas no modelo, o esquema possui também quatro repositórios (*names* do O2) para as classes Funcionário, Departamento, Projeto e Alocação.

Os relacionamentos entre as classes deste esquema foram implementados através de chaves, como no modelo relacional. Isto ocorre devido as limitações e restrições, comentadas no Capítulo V, para a utilização de atributos que sejam referências a outros objetos durante a comparação do comportamento.

Assim, alguns métodos realizam consultas nos repositórios para calcular os seus resultados e utilizam os atributos que são chaves estrangeiras como parâmetros nas consultas. Neste ponto, os valores presentes na base de dados utilizada podem interferir nos resultados de alguns métodos, ressaltando a importância de um contexto durante o processo de comparação, conforme descrito no Capítulo V. Os métodos que apresentam este tipo de comportamento são os métodos Horas Alocadas e Saldo Disponível da classe Projetos, Salário da classe Funcionário Horista e Chefe da classe

Funcionário. A comparação do comportamento com bases de dados diferentes produzirá resultados diferentes para estes métodos.

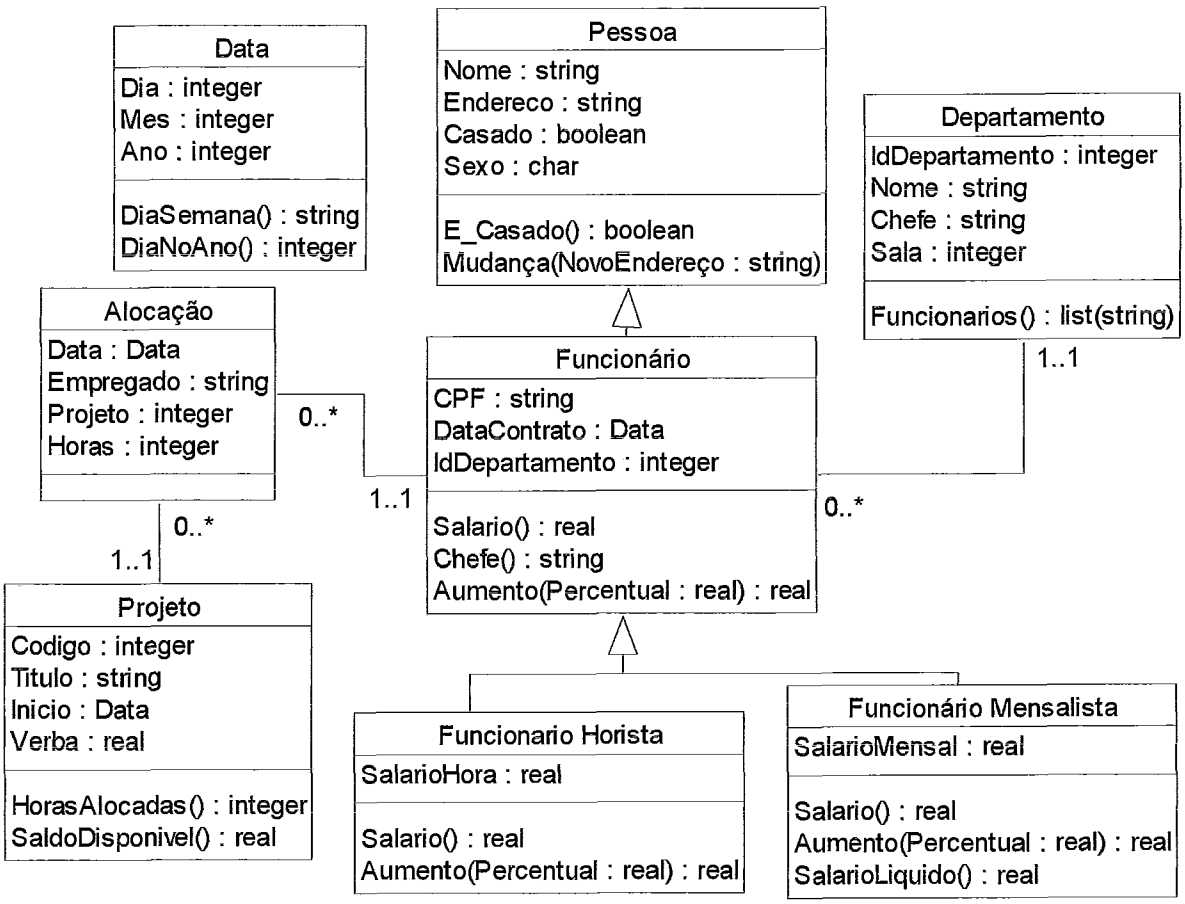


Figura VI.1 – Esquema exemplo utilizado na comparação de classes

Este esquema foi construído de forma que os métodos de classes diferentes não possuam o mesmo comportamento, já que o objetivo é avaliar a eficiência da comparação do comportamento na distinção de classes similares.

VI.3. Comparação de Esquemas Iguais

Esta Seção mostra os resultados da comparação do esquema da Figura VI.1 com ele mesmo, em duas situações distintas: com a mesma base de dados (mesmo contexto para a execução dos métodos) e com bases de dados diferentes.

VI.3.1. Comparação com Bases Iguais

Este teste foi executado utilizando a mesma base de dados (bases de cada esquema com os mesmos valores) para os dois esquemas comparados. Assim, a execução dos métodos ocorre sobre objetos destas bases de dados, ou seja, em um contexto igual.

Parte do resultado da comparação é mostrado na Tabela VI.1 (o resultado da comparação desta tabela está limitado as similaridades entre classes maiores que 0,35 para evitar uma tabela muito extensa). A Tabela VI.1 também apresenta os resultados por critério (comparação dos nomes das classes, atributos, assinaturas de métodos e comportamento ou execução).

Nesta tabela, as classes iguais tiveram um grau de similaridade igual a 1. A classe Pessoa teve um grau de similaridade próximo a 1 (0,9) em relação às suas subclasses. A diferença observada é decorrência das classes possuírem nomes completamente distintos. Ao analisar este grau de similaridade obtido, conclui-se que as classes Funcionário, Funcionário Horista e Funcionário Mensalista (subclasses de Pessoa) possuem as mesmas funcionalidades (sem alterações) da classe Pessoa. Portanto, o projetista de um banco de dados ou um integrador pode utilizar esta informação para mapear os objetos da classe Pessoa em objetos das outras classes já que as funcionalidades implementadas são as mesmas.

Porém, a classe Funcionário, em relação às suas subclasses, registrou similaridade menor (0,85) pois o comportamento de dois métodos desta classe são diferentes para as subclasses (métodos Salário e Aumento), diminuindo a similaridade no critério de execução. Ou seja, as subclasses redefiniram esses dois métodos, alterando o comportamento existente na classe Funcionário. Consequentemente, o mapeamento de objetos da classe Funcionário para objetos de Funcionário Horista e Funcionário Mensalista não pode ser realizado diretamente, exigindo a aplicação de uma função de mapeamento que atue sobre os resultados dos métodos discrepantes.

Tabela VI.1 – Similaridades para a comparação de esquemas iguais

Critério / Classes Comparadas	Nome (0,1)	Atributo (0,3)	Método (0,3)	Execução (0,3)	Final
Alocação = Alocação	1	1	1	1	1,00000
Data = Data	1	1	1	1	1,00000
Departamento = Departamento	1	1	1	1	1,00000

Critério / Classes Comparadas	Nome (0,1)	Atributo (0,3)	Método (0,3)	Execução (0,3)	Final
Funcionário = Funcionário	1	1	1	1	1,00000
Funcionário Horista = Funcionário Horista	1	1	1	1	1,00000
Funcionário Mensalista = Funcionário Mensalista	1	1	1	1	1,00000
Pessoa = Pessoa	1	1	1	1	1,00000
Projeto = Projeto	1	1	1	1	1,00000
Pessoa = Funcionário	0	1	1	1	0,90000
Pessoa = Funcionário Horista	0	1	1	1	0,90000
Pessoa = Funcionário Mensalista	0	1	1	1	0,90000
Funcionário = Funcionário Horista	1	1	1	0,5	0,85000
Funcionário = Funcionário Mensalista	1	1	1	0,5	0,85000
Funcionário Horista = Funcionário	0,5	0,875	1	0,5	0,76250
Funcionário Horista = Funcionário Mensalista	0	0,9625	1	0,5	0,73875
Funcionário Mensalista = Funcionário	0,5	0,875	0,83333	0,4	0,68250
Funcionário Mensalista = Funcionário Horista	0	0,9625	0,83333	0,4	0,65875
Projeto = Funcionário Mensalista	0	0,7	0,7875	0	0,44625
Projeto = Funcionário Horista	0	0,7	0,675	0	0,41250
Departamento = Funcionário Horista	0	0,8375	0,45	0	0,38625
Departamento = Funcionário Mensalista	0	0,8375	0,45	0	0,38625
Funcionário = Pessoa	0	0,57142	0,4	0,25	0,36643
Projeto = Funcionário	0	0,525	0,675	0	0,36000

A Tabela VI.2 mostra a matriz de comparação C (Seção IV.5.1) para a classe Funcionário Horista. Nesta matriz, as linhas e colunas numeradas correspondem às comparações entre a classe Funcionário Horista e as outras classes, como por exemplo Funcionário Horista X Funcionário (linha 6). Os elementos da matriz exibem os valores da função de relatividade (Equação 4.12) entre as classes indicadas pela linha e pela coluna com respeito a comparação com a classe Funcionário Horista.

Através dessa matriz pode-se ordenar as classes comparadas com a classe Funcionário Horista, selecionando-se as mais similares. A ordenação é possível após o estabelecimento dos valores de ranking para cada linha da matriz. Este valor corresponde ao valor mínimo dos elementos da linha (Equação 4.13). A classe mais similar possui o maior valor de ranking.

Tabela VI.2 – Matriz de Comparação para a Classe Funcionário Horista

Classes	1	2	3	4	5	6	7	8	Ranking
1	1,000	0,812	0,888	0,394	0,444	0,432	0,692	0,921	0,394
2	1,000	1,000	1,000	0,486	0,546	0,532	0,852	1,000	0,486
3	1,000	0,914	1,000	0,444	0,500	0,486	0,779	1,000	0,444
4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
5	1,000	1,000	1,000	0,888	1,000	0,973	1,000	1,000	0,888
6	1,000	1,000	1,000	0,912	1,000	1,000	1,000	1,000	0,912
7	1,000	1,000	1,000	0,570	0,641	0,624	1,000	1,000	0,570
8	1,000	0,881	0,964	0,428	0,482	0,469	0,751	1,000	0,428

Numeração das Classes:

1 – Alocação
2 – Projeto
3 – Departamento
4 – Funcionário Horista
5 – Funcionário Mensalista
6 – Funcionário
7 – Pessoa
8 – Data

Na matriz, está em destaque a classe Funcionário Horista pois esta corresponde a classe mais similar a classe comparada, no caso a própria classe Funcionário Horista. Quanto maior o valor de um elemento da matriz, maior a similaridade entre as classes comparadas. Um elemento da matriz igual a 1 indica que a classe da linha correspondente é funcionalmente mais adequada a classe Funcionário Horista do que a classe da coluna. Assim, o elemento da linha 6, coluna 5, indica que a classe Funcionário (linha 6) é mais similar em relação a classe Funcionário Horista do que a classe Funcionário Mensalista (coluna 5).

Neste momento é importante destacar o efeito da comparação do comportamento no resultado final da comparação, aumentando a diferença entre os graus de similaridades das classes realmente similares e as outras classes. A Tabela VI.3 mostra o resultado da comparação das classes do esquema com a classe Funcionário Horista sem considerar o comportamento dos métodos no processo de comparação (a função para a comparação do comportamento retornou 1 para todas as classes) e o resultado final obtido (incluindo a comparação do comportamento).

Tabela VI.3 – Resultado sem considerar o comportamento dos métodos

Tipo da Comparação / Classes Comparadas	Sem considerar comportamento	Considerando o comportamento
Alocação = Alocação	1,00000	1,00000
Data = Data	1,00000	1,00000

Tipo da Comparação Classes Comparadas	Sem considerar comportamento	Considerando o comportamento
Departamento = Departamento	1,00000	1,00000
Funcionário = Funcionário	1,00000	1,00000
Funcionário = Funcionário Horista	1,00000	1,00000
Funcionário = Funcionário Mensalista	1,00000	1,00000
Funcionário Horista = Funcionário Horista	1,00000	1,00000
Funcionário Mensalista = Funcionário Mensalista	1,00000	1,00000
Pessoa = Pessoa	1,00000	1,00000
Projeto = Projeto	1,00000	1,00000
Pessoa = Funcionário	0,90000	0,90000
Pessoa = Funcionário Horista	0,90000	0,90000
Pessoa = Funcionário Mensalista	0,90000	0,90000
Funcionário Horista = Funcionário	0,91250	0,76250
Funcionário Horista = Funcionário Mensalista	0,88875	0,73875
Funcionário Mensalista = Funcionário	0,86250	0,68250
Funcionário Mensalista = Funcionário Horista	0,83875	0,65875
Projeto = Funcionário Mensalista	0,74625	0,44625
Projeto = Funcionário Horista	0,71250	0,41250
Departamento = Funcionário Horista	0,68625	0,38625
Departamento = Funcionário Mensalista	0,68625	0,38625
Funcionário = Pessoa	0,59143	0,36643
Projeto = Funcionário	0,66000	0,36000

A aplicação da comparação do comportamento aumentou a capacidade de diferenciação da técnica de comparação, aumentando a diferença entre os graus de similaridades das classes comparadas. Este efeito sobre os resultados está de acordo com os resultados obtidos por PIERCE e PODGURSKY (1992) para a Busca por Comportamento (Seção II.4.4) que demonstraram a grande precisão deste tipo de busca.

VI.3.2. Comparação com Bases Diferentes

Devido às limitações existentes na implementação da técnica proposta quanto a comparação do comportamento, a utilização de bases de dados diferentes (contextos diferentes) para os dois esquemas alvos da comparação produz resultados distintos em relação aos resultados obtidos com o uso de bases iguais.

As classes afetadas por este tipo de comportamento foram Projeto, Funcionário Horista e Funcionário (e seus descendentes), cujos métodos produzem resultados diferentes com a utilização de bases diferentes. No pior caso, que ocorre para as classes em que todos os métodos são dependentes da base de dados, a utilização de bases diferentes tem como efeito a anulação da parcela relativa à comparação do comportamento no resultado final. A Tabela VI.4 mostra este efeito para a comparação da classe Projeto.

Tabela VI.4 – Comparação da Classe Projeto em Contextos Diferentes

Tipo da Comparação / Classes Comparadas	Mesmo contexto	Contexto diferente
Projeto = Projeto	1,00000	0,70000
Projeto = Funcionário Mensalista	0,44625	0,44625
Projeto = Funcionário Horista	0,41250	0,41250
Projeto = Funcionário	0,36000	0,36000
Projeto = Data	0,27600	0,27600
Projeto = Departamento	0,19350	0,19350
Projeto = Alocação	0,17850	0,17850
Projeto = Pessoa	0,12000	0,12000

A comparação das classes cujos métodos produzem resultados iguais, independente da base de dados em que são executados, não sofreram alterações. Deste modo, a utilização de contextos (bases de dados) diferentes no suporte a comparação não afeta diretamente a técnica proposta já que seu efeito só ocorre para algumas classes. Para a maioria das classes, a inicialização dos atributos e parâmetros é suficiente para determinar o comportamento dos seus métodos. Um exemplo deste caso ocorre para a classe Funcionário Mensalista (Tabela IV.5), onde apenas um, em um total de cinco métodos considerados no teste de comportamento, é afetado pela a utilização de um contexto diferente.

Tabela VI.5 – Comparação da Classe Funcionário Mensalista em Contextos Diferentes

Tipo da Comparação / Classes Comparadas	Contexto diferente	Mesmo contexto
Funcionário Mensalista = Funcionário Mensalista	0,94000	1,00000
Funcionário Mensalista = Funcionário	0,62250	0,68250
Funcionário Mensalista = Funcionário	0,59875	0,65875

Tipo da Comparação / Classes Comparadas	Contexto diferente	Mesmo contexto
Horista		
Funcionário Mensalista = Pessoa	0,31000	0,32500
Funcionário Mensalista = Projeto	0,18150	0,18150
Funcionário Mensalista = Departamento	0,13988	0,13988
Funcionário Mensalista = Data	0,11325	0,11325
Funcionário Mensalista = Alocação	0,09488	0,09488

VI.4. Comparação de Esquemas Diferentes

O objetivo desta Seção é avaliar o funcionamento da técnica de comparação proposta ao comparar classes similares mas com algumas diferenças em cada um dos critérios considerados (nomes, atributos, assinaturas de métodos), com exceção do comportamento dos métodos que são idênticos. Para isso, criou-se um segundo esquema de teste, com a presença das mesmas classes do Esquema 1, retirando-se apenas as classes Funcionário Horista e Funcionário Mensalista. Esta última foi substituída pela classe Trabalhador (Figura VI.2).

Com o objetivo de avaliar a capacidade da comparação do comportamento de uma classe em identificar uma classe similar, foi introduzida no Esquema 2 uma classe cuja estrutura sintática (nomes, tipos, atributos e assinaturas de métodos) é similar à classe Funcionário Mensalista do Esquema 1. Esta classe, denominada Empresa (Figura VI.2), implementa as funcionalidades associadas a uma pessoa jurídica e possui quase os mesmos atributos e métodos da classe Funcionário Mensalista (que implementa uma pessoa física), com pequenas variações de nomes. Porém, o comportamento dos métodos dessas duas classes é completamente distinto.

Os esquemas foram comparados utilizando bases de dados iguais, ou seja o mesmo contexto, evitando possíveis distorções devido aos estados diferentes existentes nos dois bancos de dados. A Tabela VI.6 mostra o resultado da comparação da classe Funcionário Mensalista do Esquema 1 com as classes Trabalhador, Empresa e Funcionário (as outras classes foram omitidas devido ao baixo grau de similaridade) do Esquema 2.

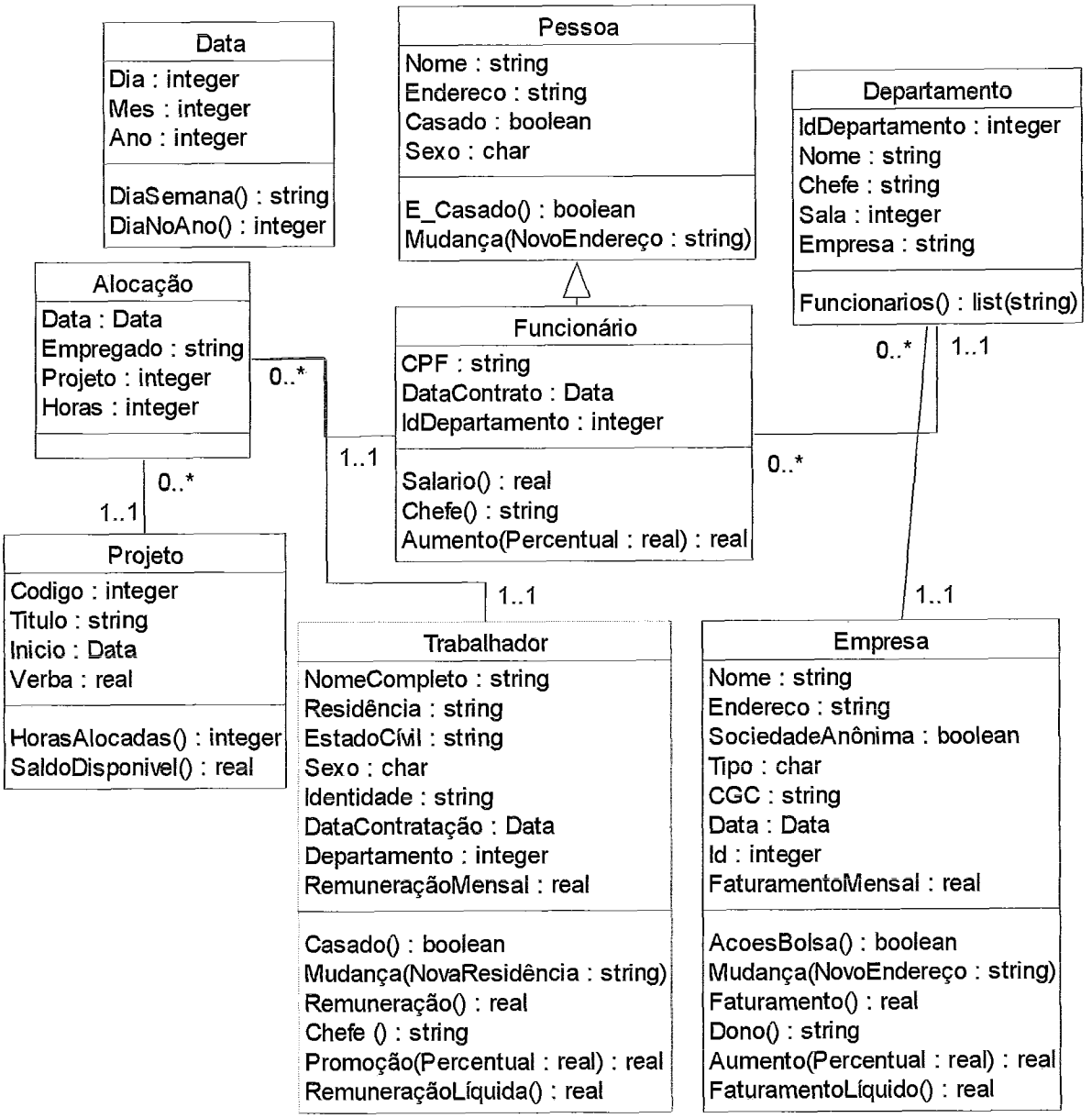


Figura VI.2 – Classes Novas do Esquema Diferente

Tabela VI.6 – Similaridades para a comparação de esquemas diferentes

Critério / Classes Comparadas	Nome (0,1)	Atributo (0,3)	Método (0,3)	Execução (0,3)	Final
Funcionário Mensalista = Empresa	0	0,8125	0,93333	0	0,52375
Funcionário Mensalista = Funcionário	0,5	0,875	0,833	0,4	0,68250
Funcionário Mensalista = Trabalhador	0	0,70625	0,91916	1	0,78762

Observando a tabela, destaca-se o efeito expressivo do critério relativo ao comportamento das classes no resultado final da comparação. A Tabela VI.7 mostra a similaridade entre as classes comparadas porém sem considerar a execução dos métodos no resultado final (foi atribuído 1 para o aspecto da execução a todas as classes).

Tabela VI.7 – Comparação de esquemas diferentes sem considerar o comportamento dos métodos

Tipo da Comparação / Classes Comparadas	Considerando o comportamento	Sem considerar comportamento
Funcionário Mensalista = Empresa	0,52375	0,82375
Funcionário Mensalista = Funcionário	0,68250	0,86250
Funcionário Mensalista = Trabalhador	0,78762	0,78762

Neste caso, a classe com o maior grau de similaridade em relação a classe Funcionário Mensalista é a classe Funcionário (aproximadamente 0,862), enquanto que a classe Trabalhador (0,78762), que possui um comportamento semelhante a Funcionário Mensalista, foi considerada menos similar do que a classe Empresa (0,82375). Isto ocorre pois a estrutura sintática (nomes, atributos e assinatura dos métodos) das classes Funcionário e Empresa são mais parecidas com Funcionário Mensalista do que a estrutura da classe Trabalhador.

Outros métodos de comparação de componentes (classes, entidades, etc.) podem produzir resultados indesejáveis ao serem aplicados neste exemplo, exigindo um acompanhamento e uma intervenção do projetista do banco de dados durante o processo para corrigir os falsos mapeamentos (DAMASCENO *et al.*, 1997, THIEMES e SIEBES, 1993, FANKHAUSER *et al.*, 1991, LARSON *et al.*, 1989, NAVATHE *et al.*, 1986). Isto ocorre pois estas técnicas de comparação consideram somente os nomes e estruturas dos seus atributos e métodos. Mesmos as técnicas que analisam o comportamento da classe de alguma forma (DAMASCENO *et al.*, 1997, THIEMES e SIEBES, 1993) estão sujeitas a produzir resultados inesperados ao não avaliar completamente o comportamento dos métodos da classe, já que estes podem ser utilizados para implementar várias regras de negócio.

A matriz de comparação construída para este exemplo permite destacar ainda mais a eficiência e a precisão decorrente da aplicação da comparação do comportamento no processo de comparação.

Tabela VI.7 – Matriz de Comparação em relação à classe Funcionário Mensalista para Esquemas Diferentes

Classes	Empresa	Funcionário	Trabalhador	Ranking
Empresa	1	0,76738	0,66498	0,66498
Funcionário	1	1	0,86653	0,86653
Trabalhador	1	1	1	1

A utilização da ordenação difusa permite identificar qual a classe mais similar em relação a uma classe de um esquema, assim como determinar o quanto mais similar ela é. Os graus de similaridades obtidos pela aplicação da função de pertinência da Equação 4.1 representam a comparação de um par de classes. A matriz de comparação complementa o processo ao representar a relação entre a comparação de dois pares de classes. Portanto, ao expressar uma análise relativa das comparações entre as classes, a matriz de comparação distingue mais expressivamente os resultados obtidos nas comparações.

VI.5. Considerações Finais

A incorporação da lógica difusa durante todo o processo de comparação permitiu determinar graus de similaridade entre as classes comparadas e não apenas a exata equivalência entre elas. Devido a incerteza presente durante o processo de comparação, a identificação de quanto uma classe é similar a outra é uma informação que pode ser utilizada no mapeamento entre classes de esquemas diferentes. Esta informação também é relevante para o processo de reutilizar classes de um esquema, já que a similaridade obtida na técnica proposta quantifica a funcionalidade em comum entre duas classes.

A atribuição de graus de similaridade entre as classes comparadas possibilitou também a adoção de métodos de ordenação, facilitando a seleção da classe mais similar.

O exemplo da Seção VI.4 ilustra uma situação em que a comparação do comportamento é decisiva na identificação de classes com mesma funcionalidade e equivalência semântica.

VII. Conclusão

VII.1. Contribuições

Esta tese abordou a comparação de classes, atividade essencial na integração de esquemas ou na reutilização de componentes. Muitos dos trabalhos existentes na área de integração de esquemas (integração de visões ou de banco de dados) pressupõe que a etapa de comparação dos esquemas está concluída ou apresenta uma proposta superficial para a sua realização, concentrando-se na resolução dos conflitos identificados durante a comparação através da elaboração de mapeamento entre os elementos dos esquemas (KASHYAP, SHETH, 1996, SPACCAPIETRA, PARENT, 1994, BERTINO *et al.*, 1994, LI e CLIFTON, 1994, BATINI *et al.*, 1986).

As técnicas tradicionais para a comparação de componentes, criadas para a recuperação de componentes (STOCKWELL e KRAUSE, 1993, XEXEO, 1994, GAMMA *et al.*, 1995) ou para a integração de esquemas (FANKHAUSER *et al.*, 1991, LARSON *et al.*, 1989, SOUZA, 1986a, NAVATHE *et al.*, 1986), podem ser aplicadas ao esquemas orientados a objetos. Algumas técnicas consideram também o comportamento das classes, representando o comportamento de regras de integridade e transição de estados (DAMASCENO *et al.*, 1997) ou considerando apenas os métodos de atualização das classes (THIEME E SIEBES, 1993). Porém, estas técnicas não consideram o comportamento dinâmico dos métodos que é um importante aspecto na implementação de um esquema orientado a objetos e engloba importantes regras de negócio.

A técnica de comparação proposta incorpora a comparação do comportamento como um dos critérios avaliados e utiliza a lógica difusa para manipular a incerteza decorrente do processo de comparação de conceitos distintos. Este trabalho mostrou que a comparação do comportamento dos métodos é possível, funciona e, assim, esta técnica pode ser utilizada para estender um método tradicional de comparação, como a comparação de nomes ou de estruturas, mostrando-se eficiente e precisa na distinção das classes que possuem as mesmas funcionalidades, como mostra o exemplo da Seção VI.4.

A técnica proposta pode ser adotada em um ambiente de reutilização de software (como uma biblioteca de componentes) ou de integração de esquemas (mediadores ou projetos de bancos de dados distribuídos) e incorpora características importantes de comparação operacional de componentes.

Outra contribuição deste trabalho é uma arquitetura para a implementação da ferramenta de comparação que assegure a flexibilidade e expansibilidade da técnica de comparação. A arquitetura é formada por classes responsáveis pela implementação de cada aspecto da técnica de comparação proposta e permite a incorporação de novos critérios de comparação (através da introdução de novas classes) ou a substituição (especialização) das classes existentes para alterar a estratégia de comparação de um aspecto.

Um protótipo de um sistema de comparação baseado na técnica de comparação proposta foi implementado em um banco de dados orientado a objetos. Devido ao ambiente de desenvolvimento e a complexidade dos problemas, foram adotadas algumas limitações à implementação, mas as soluções para estes problemas foram indicadas, como por exemplo a comparação de métodos que não retornem valores (procedimentos ao invés de funções). Estes métodos podem ser comparados a partir do estado da base, ou seja, a partir dos valores dos atributos que foram alterados pela sua execução. O registro de modificações da base de dados (*log* de alterações) pode ser consultado para isso.

Por último, um conjunto de testes gerados a partir do protótipo desenvolvido demonstra a aplicação da estratégia sugerida.

VII.2. Trabalhos Futuros

A comparação do comportamento adotada abrange a interface dos métodos (assinatura) e o comportamento observável (avaliado através da execução dinâmica dos métodos). Porém, existem outros critérios para avaliar o comportamento das classes e dos seus métodos, como por exemplo critérios internos da forma de implementação dos métodos refletidos nos algoritmos e estrutura de dados utilizados. Outro critério que pode ser avaliado é a comparação das exceções geradas pelos diferentes métodos.

Outra extensão ao trabalho é a utilização de outras linguagens de programação na implementação dos métodos a serem comparados. As linguagens que oferecem, de

alguma forma, informações sobre uma classe, e suas propriedades e métodos, podem ser utilizadas. Neste caso, a consulta ao meta-esquema do O2 pode ser substituída pela consulta a interface da própria classe em busca de informações sobre ela. A linguagem Java, por exemplo, oferece este tipo de recurso para consultar a interface de suas classes.

O mapeamento entre classes fornecido pela técnica proposta pode ser aplicado na elaboração de mediadores sendo utilizado como mais uma ferramenta para auxiliar a construção do esquema do mediador. Os resultados da comparação entre as classes por cada critério também podem ser utilizados para implementar uma ferramenta de reconhecimento de padrões baseada na lógica difusa.

VIII. Bibliografia

ADAMS, T., 1994, "Retaining Structure Selection with Unequal Fuzzy Project-Level Objectives", *Jounnal of Intelligent Fuzzy Systems,* v. 2, n. 3, pp. 251-266.

ALEXANDER, C., 1979, *The Timeless Way of Building.* New York, USA, Oxford University Press.

BARROS, M.O., 1995, *Recuperaçao de Componentes em Bibliotecas de Software: Uma Abordagem Conexionista*, Tese de M.Sc., COPPE / UFRJ, Rio de Janeiro, Brasil.

BATINI, C., LENZERINI, M., 1984, "A Methodology for Data Schema Integration in the Entity Relationship Model", *IEEE Transactions On Software Engineering,* v. 10, n. 6 (Nov.), pp. 650-664.

BATINI, C., LENZERINI, M., e NAVATHE, S.B., 1986, "Comparison of Methodologies for Database Schema Integration", *ACM Computing Surveys*, v. 18, n. 4 (Dec.), pp. 323-364.

BERTINO, E., NEGRI, M., PELAGATTI, G., e SBATTELLA, L., 1994, "Applications of Object-Oriented Technology to the Integration of Heterogeneous Database Systems", *Distributed and Parallel Databases*, v. 2, n. 4 (Oct.), pp. 343-370.

CANNATA, P.E., 1991, "The Irresistible Move towards Interoperable Database Systems", In:*Proceedings of the First International Workshop on Interoperability in Multidabase Systems*, v. 1, pp. 2-5, Kyoto, Japan, April.

CHEN, P., 1976, "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions On Database Systems*, v. 1, n. 1 (Mar), pp. 9-36.

COHEN, B., HARWOOD, W.T., JACKSON, M.I., 1986, *The Specification of Complex Systems*, 1st ed., Massachusetts, USA, Addison-Wesley.

COX, E., 1994, *The fuzzy systems handbook: a practitioner's guide to building and maintaining fuzzy systems*, Boston, USA, AP Professional.

COX, E., 1995, *Fuzzy Logic for Business and Industry*, Rockland, Massachusetts, USA, Charles River Media.

DAI, H., 1997, "An Object-Oriented Approach to Schema Integration and Data Mining in Multiple Databases". In: *Proceedings of the Technology of Object-Oriented Languages and Systems-Tools*, v. 24, Beijing, China, Setembro.

DAMASCENO, J.C., RIBEIRO, C.H.F.P., & OLIVEIRA, J.P.M., 1997, "Acesso Integrado a Banco de Dados Distribuídos Heterogêneos utilizando CORBA". In: *Anais do XII Simpósio Brasileiro de Banco de Dados,* pp. 333-348, Fortaleza, Ceará, Brasil, Outubro (13-15).

DAMIANI, E., FUGINI, M. G., e BELLETTINI, C., 1999, "A Hierarchy-Aware Approach to Faceted Classification of Object-Oriented Components", *ACM Transaction on Software Engineering and Methodology*, v. 8, n. 3 (July), pp. 215-262.

DUBOIS, D., PRADE, H., 1980, *Fuzzy Sets and Systems: Theory and Applications*, New York, USA, Academic Press.

ELMASRI, R., HEVNER, A., WELDREYER, J., 1985, "The Category Concept: An Extention to the Entity-Relationship Model", *Journal of Data and Knowledge Engineering*, v. 1, n. 1 (Jun), pp. 75-116.

FANKHAUSER, P., KRACKER, M., NEUHOLD, E., 1991, "Semantic vs. Structural Resemblance of Classes", *ACM SIGMOD Record*, v. 20, n. 4 (Dez), pp. 59-63.

GAMMA, E., HELM, R., JOHNSON, R., *et al.*, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA, Addison-Wesley.

GARLAND, S.J. , GUTTAG, J.V., 1991, *A Guide to LP, the Larch Prover*, Report 82, DEC Systems Research Center, Palo Alto, Calif., Dez.

HALL, R.J., 1993, "Generalized Behavior-based Retrieval", In: *Proceedings of the 15th International Conference on Software Engineering*, pp. 371-380, Baltimore, USA, Maio.

JOHNSON, R., 1997, "Frameworks = (Components + Patterns)", *Communications ACM*, v. 40, n. 10 (Oct.), pp. 39-42.

KASHYAP, V., & SHETH, A., 1996, "Semantic and Schematic Similarities Between Database Objects: a Context-Based Approach", *The VLDB Journal*, n. 5, pp. 276-304.

KATZ, S., RICHTER, C.A., e THE, K.-S., 1987, "PARIS: A System for Reusing Partially Interpreted Schemas", In: *Proceedings of the 9th International Conference on Software Engineering*, pp. 377-385, Monterey, CA, March.

LARSON, J.A., NAVATHE, S.B., & ELMASRI, R., 1989, "A Theory of Attribute Equivalence in Databases with Application to Schema Integration", *IEEE Transactions On Software Engineering*, v. 15, n. 4 (April), pp. 449-463.

LEA, D., 1994, "Christopher Alexander: An Introduction for Object-Oriented Designers", *ACM SIGSOFT Software Engineering Notes*, v. 19, n. 1 (Jan), pp. 39-46.

LÉCLUSE, C., RICHARD, P., 1989, "The O2 Database Programming Language", In: *Proceedings of the 15th International Conference on Very Large Databases*, pp. 411-422, Palo Alto, CA, USA.

LI, W.-S., CLIFTON, C., 1994, "Semantic Integration in Heterogeneous Databases Using Neural Networks", In: *Proceedings of the 20th VLDB Conference*, v. 20, pp. 1-12, Santiago, Chile, Set. (12-15).

NAVATHE, S.B., ELMASRI, R., LARSON, J.A., 1986, "Integrating User Views in Database Design", *IEEE Computer*, v. 19, n. 1 (Jan), pp. 50-62.

NAVATHE, S.B., SASHIDHAR, T., ELMASRI, R., 1984, "Relationship Matching in Schema Integration", In: *Proceedings of the 10th International Conference on Very Large Data Bases*, pp. 78-90, Singapure, Agosto (27-31).

ÖZSU, M.T., VALDURIEZ, P., DAYAL, U., 1994, "An Introduction to Distributed Database Management", In: *Distributed Object Management*, v. 1, *Introduction*, Morgan Kaufmann, pp. 1-24.

PITOURA, E., BUKHRES, O., & ELMAGARMID, A., 1995, "Object Orientation in Multidatabase Systems", *ACM Computing Surveys*, v. 27, n. 2 (June), pp. 141-195.

PODGURSKI, A., PIERCE, L., 1992, "Behavior Sampling: A Technique for Automated Retrieval of Reusable Components", In: *Proceedings of the 14th International Conference on Software Engineering*, pp. 349-360, Melbourne, Austrália, Maio (11-15).

PODGURSKI, A., & PIERCE, L., 1993, "Retrieving Reusable Software by Sampling Behavior", *ACM Transactions On Software Engineering and Methodology*, v. 2, n. 3 (July), pp. 286-303.

RIBEIRO, C.H.F.P., OLIVEIRA, J.P.M., 1994, "Heterogeneity in Object with Roles Conceptual", In: *Anais do IX Simpósio Brasileiro de Banco de Dados*, pp. 188-200, São Carlos, São Paulo, Brasil, Setembro (5-7).

RIBEIRO, C.H.F.P., & OLIVEIRA, J.P.M., 1995, "Tratamento de Heterogeneidade em Esquemas Conceituais Orientados a Objetos". In: *Anais do X Simpósio Brasileiro de Banco de Dados*, pp. 261-275, São Carlos, São Paulo, Brasil, Outubro (2-4).

ROSS, T., 1995, *Fuzzy Logic With Engineering Applications*, New York, USA, McGraw-Hill.

SAKAWA, M., 1993, *Fuzzy Sets and Interactive Multiobjective Optimization*, New York, USA, Plenum Press.

SALTON, G., McGILL, M.H., 1983, *Introduction to Modern Information Retrieval*, New York, USA, McGraw-Hill.

SHETH, A., LARSON, J.A., 1990, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, v. 22, n. 3 (Set), pp. 183-236.

SOUZA, J.M., 1986a, *Software Tools for Conceptual Schema Integration*, Tese de D.Sc., School of Information Systems, University of East Anglia, England.

SOUZA, J.M., 1986b, "SIS - A Schema Integration System", In: *Proceedings of the 5th British National Conference on Databases*, pp. 167-185, Canterbury, England, July.

SPACCAPIETRA, S., PARENT, C., 1994, "View Integration: A Step Forward in Solving Structural Conflicts", *IEEE Transactions on Knowledge and Data Engineering*, v. 6, n. 2 (Abr), pp. 258-274.

STEIGERWALD, R., 1992, "Reusable Component Retrieval with Formal Specifications", In: *5th Annual Workshop on Software Reuse*, Palo Alto, CA, USA.

STOCKER, P.M., CANTIE, R., 1983, "A Target Logical Schema: The ACS", In: *Proceedings of the 9th International Conference on Very Large Databases*, pp. 309-310, Florence, Italy.

THIEME, C., e SIEBES, A., 1993, "Schema Integration in Object-Oriented Databases". In: *Proceedings of International Conference on Advanced Information Systems Engineering*, pp. 54-70, Paris, France, June (8-11).

TUCHERMAN, L., FURTADO, A., CASANOVA, M.A., 1985, "A Tool for Modular Database Design", In:*Proceedings of the 11th International Conference on Very Large Databases*, Stockholme, Sweden, Agosto (21-23).

WIEDERHOLD, G., e GENESERETH, M., 1996, "The Basis for Mediation", *IEEE Expert*, n. 2 (May).

WING, J.M., 1990, "A Specifier's Introduction to Formal Methods", *IEEE Computer*, pp. 8-24.

XEXEO, G.B., 1994, *Um Sistema de Reutilizaçao de âmbito Global*, Tese de D. Sc., COPPE / UFRJ, Rio de Janeiro, RJ.

ZADEH, L., 1965, "Fuzzy Sets", *Information Control*, v. 8, pp. 338-353.

ZAREMSKI, A.M., WING, J.M., 1997, "Specification Matching of Software Components", *ACM Transactions On Software Engineering and Methodology*, v. 6, n. 4 (Out), pp. 333-369.

Anexo I

Este anexo apresenta os valores dos pesos utilizados nas equações do Capítulo IV propostos nesta implementação da técnica de comparação.

Os pesos da Tabela A1.1 foram utilizados na Equação 4.1 para a comparação de classes.

Tabela A1.1 – Pesos de cada aspecto considerado na comparação de classes

Peso	Valor
w_N (nomes)	0,1
w_A (atributos)	0,3
w_M (assinatura dos métodos)	0,3
w_E (execução dos métodos)	0,3

Os pesos da Tabela A1.2 foram utilizados na equação 4.3 para a comparação de atributos.

Tabela A1.2 – Pesos de cada aspecto considerado na comparação de atributos

Peso	Valor
wa_N (nome)	0,3
wa_T (tipo)	0,7

Os pesos da Tabela A1.3 foram utilizados na equação 4.4 para a comparação de métodos.

Tabela A1.3 – Pesos de cada aspecto considerado na comparação de métodos

Peso	Valor
wm_N (nome)	0,1
wm_T (tipo)	0,45
wm_P (parâmetros)	0,45

Os pesos da Tabela A1.4 foram utilizados na equação 4.5 para a comparação de parâmetros.

Tabela A1.4 – Pesos de cada aspecto considerado na comparação de parâmetros

Peso	Valor
wp_N (nome)	0,3
wp_T (tipo)	0,7

Tabela A1.5 – Similaridade entre tipos

Tipos	Integer	string	Real	boolean	char	set	unique set	list	Class	tuple
Integer	1,0		0,5							
String		1,0								
Real	0,4		1,0							
Boolean				1,0						
Char					1,0					
set						1,0 * F_T	0,9 * F_T	0,8 * F_T		
Un. Set						0,9 * F_T	1,0 * F_T	0,8 * F_T		
list						0,8 * F_T	0,8 * F_T	1,0 * F_T		
Class									1 * F_C	
tuple										1 * F_P

Na Tabela A1.5, a similaridade de alguns tipos são calculadas utilizando outras funções de similaridades. Essas funções são explicadas no Capítulo IV. As telas que permitem alterar a configuração dos valores da Tabela A1.5 aparecem nas Figuras A.1 e A.2.

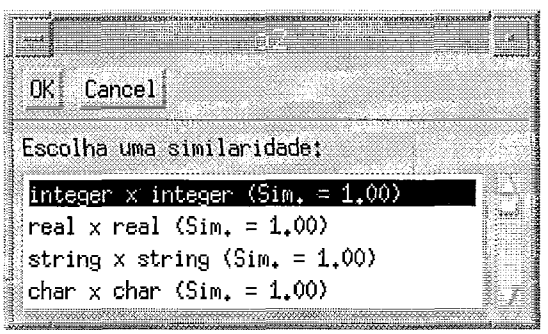


Figura A.1 – Diálogo para a seleção da similaridade entre tipos

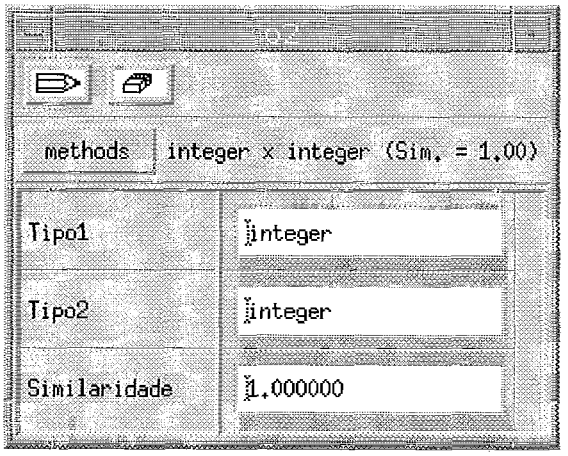


Figura A.2 – Tela para edição de uma similaridade entre tipos