UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Um Modelo Referência de Objetos para Sistemas de Banco de Dados

por *Regina Coeli Ruschel*
orientador Prof. Dr. Léo Pini Magalhães

Tese submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos requisitos parciais para obtenção do Título de Doutor em Engenharia Elétrica (área Automação).

5 de julho de 1996

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

R893m

Ruschel, Regina Coeli
Um modelo referência de objetos para sistemas de banco de dados / Regina Coeli Ruschel.--Campinas, SP: [s.n.], 1996.

Orientador: Léo Pini Magalhães.
Tese (doutorado) - Universidade Estadual de Campinas Faculdade de Engenharia Elétrica e de Computação.

1. Banco de dados orientado a objetos. 2. Formalismo. 3. Teoria dos conjuntos. I. Magalhães, Léo Pini. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Um Modelo Referência de Objetos para Sistemas de Banco de Dados

Autor: *Regina Coeli Ruschel*

Orientador: Prof. Dr. Léo Pini Magalhães

Data: 05.07.1996

Banca :

- Presidente:
  Prof. Dr. Léo Pini Magalhães (DCA/FEEC/UNICAMP)
- Membros Titulares:
  Prof. Dr. Ivan L.M. Ricarte (DCA/FEEC/UNICAMP)
  Prof. Dr. Mario Jino (DCA/FEEC/UNICAMP)
  Profa. Dra. Claudia M. Bauzer Medeiros (IC/UNICAMP)
  Prof. Dr. Paulo C. Masiero (ICM/USP-São Carlos)
- Membros Suplentes:
  Profa. Dra. Beatriz M. Daltrini (DCA/FEEC/UNICAMP)
  Prof. Dr. Philipe R.B. Devloo (DCC/FEC/UNICAMP)

Conteúdo

Lista de Figuras

Lista de Tabelas

Resumo

A análise dos vários modelos de objetos de SBDOOs (Sistemas de Banco de Dados Orientados a Objetos) existentes mostra não existir um padrão de consenso. Tal falha prejudica a portabilidade e inter-operabilidade dos SBDOOs, desmotivando a aceitação e adoção destes no mercado. Com o objetivo de solucionar esta problemática este trabalho propõe um modelo referência de objetos e um formalismo correspondente. O modelo referência proposto é compatível com o padrão ODMG-93. Portanto, trata as divergências entre os atuais modelos de objetos com uma abordagem padronizada. O formalismo desenvolvido fundamenta o modelo referência, matematicamente, na teoria de conjuntos. Este formalismo é flexível e abrangente, voltado não apenas para o modelo referência proposto, mas também para o paradigma de orientação a objetos. Conseqüentemente, o modelo referência, via seu formalismo, pode ser utilizado para caracterizar e/ou mapear modelos de objetos divergentes no contexto de SBDOO, contribuindo assim significativamente para uma melhor conceituação da área.

Abstract

An analysis over existing object models of OODBSs (Object Oriented Database System) shows the absence of a standard. This flaw limits the portability and interoperability of OODBSs, harming its endorsement, therefore demotivating its use in the market. Aiming for a solution to the problem this work proposes a reference object model and the corresponding formalism. The reference object model is compatible to the proposed standard ODMG-93. Therefore, a standardized solution is given to the divergence among current object models. The formalism developed founds the reference model, mathematically, through the set theory. This formalism is flexible and broad in such a way that it is not restricted to the reference model and, therefore, also represents the object oriented paradigm. The reference object model, through its formalism, can be used to characterize and/or map diverging object models in the context of OODBS, hence contributing significantly to improve the conceptualization of the area.

Dedico este trabalho a
Renato e Alaides, meus pais,
Aluizio, o maridão, e
Paola, que acaba de chegar.

Agradecimentos

A lista de agradecimentos é extensa.

Primeiro gostaria de agradecer a dedicação e extrema paciência do Prof. Dr. Léo Pini Magalhães, que soube ser orientador e amigo cada qual na medida e momentos certos: sem o qual este trabalho não teria sido realizado. Em seguida gostaria de agradecer aos colegas matemáticos da FEC, Renato Soliani e Mario Conrado Cavicchia, que por inúmeras vezes discutiram comigo a parte de formalização deste trabalho.

Agradeço aos professores Prof. Dr. Ivan L.M. Ricarte, Prof. Dr. Mario Jino, Profa. Dra. Claudia M. Bauzer Medeiros, Prof. Dr. Paulo C. Masiero, Profa. Dra. Beatriz M. Daltrini e Prof. Dr. Philipe R.B. Devloo, por terem aceito participar da avaliação deste trabalho; em especial aos professores Jino e Ricarte, pelas profícuas discussões do mesmo.

Agradeço também o apoio da Faculdade de Engenharia Civil da UNICAMP e de colegas, especialmente das professoras Doris C.C.K. Kowaltowski e Stelamaris Rolla Bertoli, grandes incentivadoras.

Aos colegas de pós-graduação, hoje espalhados por todo país — Curitiba, Salvador, Bauru, Jaú, Joinville, Campinas ... , também seguem agradecimentos. Ao colega e amigo "Pingo" uma menção especial pela ajuda na utilização do formatador de texto LaTeX e também pelo auxílio em inúmeras instalações do mesmo.

Last but not least, vão os meus agradecimentos a Aluizio Saiter Mota — amigo, marido, ex-colega de pós-graduação, grande fornecedor de bibliografia, muito paciente, que soube esperar, cobrar e incentivar. Aliás, uma história de amor iniciada nos corredores da FEEC.

Capítulo 1

Motivação

O desenvolvimento de Sistemas de Banco de Dados Orientados a Objetos (SBDOOs) foi motivado pela falha na aplicabilidade de sistemas de banco de dados tradicionais em domínios de aplicações como CAD/CAM, Inteligência Artificial, Sistemas de Informação e CASE, entre outros. Surgiram então três abordagens diferentes, que associam orientação a objetos a sistemas de banco de dados (ou a persistência), para solucionar esta questão. Uma abordagem partiu de grupos de linguagens de programação, que procuraram formas de acrescentar persistência aos dados. Outra abordagem originou-se de grupos de banco de dados, que procuraram acrescentar a sistemas de banco de dados a capacidade de modelagem de objetos complexos e de linguagem estendível de definição de dados. A última abordagem partiu de grupos que tentaram algo novo, nem linguagem de programação persistente e nem sistemas de banco de dados não convencionais, mas um sistema que oferecia as duas facilidades [Pet87].

Desta terceira abordagem surgiram os verdadeiros SBDOOs, para os quais apresentamos definições sugeridas por diferentes autores:

F. Bancilhon e W. Kim em [BK90] e W. Kim em [Kim90] :

> *Um SBDOO é um sistema de banco de dados que suporta diretamente um modelo de dados orientado a objeto. O modelo de objetos deve oferecer o conceito de objetos encapsulados, identidade de objetos, atributos e métodos para descrever respectivamente estado e comportamento de objetos, classes e hierarquia de classes. Como qualquer sistema de banco de dados, deve ainda prover persistência de dados e da descrição destes (i.e., esquema) e prover interfaces para definição e manipulação de esquemas e dados. Além destas funcionalidades básicas deve poder ser estendido*

com facilidades de linguagem de consulta, controle de integridade (gerenciamento de transações e gatilhos), melhoramento de desempenho (indexação secundária e clustering), controle de concorrência e autorização e gerenciamento de versões tanto dos dados (objetos) como da descrição dos dados (esquemas).

R. Unland e G. Schlageter em [US90] :

Um SBDOO deve satisfazer dois critérios: deve ser um sistema de banco de dados e deve ser um sistema orientado a objetos. O primeiro critério traduz-se em seis facilidades: persistência, gerenciamento de armazenamento secundário, compartilhamento de dados (concorrência), confiabilidade de dados (gerenciamento de transações e recuperação), linguagem de consulta ad-hoc e manipulação de esquema. O segundo critério traduz-se em oito facilidades: objetos complexos, encapsulamento/abstração de dados, tipos ou classes, herança, polimorfismo/ligação tardia (late binding), ser estendível e ter completitude computacional.

Deve-se notar que as definições são coerentes, ambas enfatizando a necessidade do sistema ser um sistema de banco de dados tradicional e também ser um sistema orientado a objetos. Esta segunda característica é satisfeita pela adoção de um modelo de dados orientado a objetos, *i.e.*, um modelo de objetos. Entretanto, na ausência de um modelo padrão de objetos e de seu formalismo, este é sempre descrito por um conjunto mínimo de funcionalidades. Esta inexistência de um modelo padrão de objetos tem sua origem na característica ainda experimental da área [ABD+92, Bee89, Kim90, LKM90].

Vários trabalhos surgiram com o intuito de suprir esta falta de padronização e fundamentação teórica no modelo de objetos para sistemas de banco de dados. Alguns sugerem o seu modelo de objeto como o modelo padrão, como é o caso de Galileo [AGOO85], EXTRA/EXCESS [CDV88] e o O2 [LRV88]. Grupos e comitês propõem padrões (SQL3 [Kul94] e ODMG-93 [Cat94a]). Outros são trabalhos de formalismo propriamente dito, *i.e.*, uma descrição algébrica de um modelo genérico de objetos complexos para sistemas de banco de dados [BS92] ou uma descrição com técnicas de especificação formal de tipos abstratos de dados [Bau90].

Entretanto, persistem e destacam-se trabalhos que se dedicam à descrição informal do modelo de objetos através do esforço de cristalização dos conceitos fundamentais de orientação a objetos em sistemas de banco de dados [ABD+92, Bee89]. Em [ABD+92], a definição de SBDOO de [US90] é adotada, sendo que os autores acrescentam, às oito facilidades de

orientação a objetos (OO) obrigatórias, facilidades opcionais (herança múltipla, verificação e inferência de tipos, distribuição, projeto de transações, versões) e abertas (paradigma de programação, sistema de representação, sistema de tipos e uniformidade).

O presente trabalho é uma contribuição para a convergência de um modelo de objetos comum entre os SBDOOs através da formalização de um modelo referência de objetos compatível com uma proposta de padronização. O formalismo é construído sobre a teoria de conjuntos, tendo como diretrizes ser flexível e ter uniformidade de tratamento, de tal forma a fundamentar matematicamente um modelo referência de objetos que seja abrangente e completo.

Os capítulos que se seguem estão organizados da seguinte forma. Inicialmente, o Capítulo 2 apresenta os modelos de objeto adotados nos protótipos de pesquisas de SBDOOs que mais se destacaram na área, o ORION e o O2, e que foram transformados em produtos. Também será apresentado o modelo de objetos do padrão ODMG-93, que pretende tornar-se o padrão *de facto* adotado pela indústria. A ênfase deste capítulo é mostrar como os modelos de objeto de SBDOOs não seguem um padrão, detectar divergências e verificar como estas divergências são solucionadas numa proposta de padronização.

Em seguida, o Capítulo 3 apresenta informalmente o modelo referência de objetos proposto. Este modelo referência inclui todas as facilidades de orientação a objetos (OO) consideradas obrigatórias [ABD+92, Bee89, US90] e adota o padrão ODMG-93 para orientar-se por um tratamento único e padronizado das divergências encontradas nos modelos de objetos analisados. O Capítulo 4 descreve formalmente o modelo referência de objetos proposto usando a teoria de conjuntos. Finalmente, o Capítulo 5 discute a aplicabilidade do modelo referência e formalismo, assim como sua compatibilidade com o padrão ODMG-93.

O Apêndice A pretende mostrar a aplicação do modelo referência proposto descrevendo três estudos de caso, na notação gráfica introduzida no Capítulo 3. O Apêndice B apresenta a simbologia e nomenclatura utilizadas no formalismo do Capítulo 4. O Apêndice C descreve, com recursos do formalismo do modelo referência proposto, variações existentes no mecanismo de herança. Este apêndice procura mostrar a flexibilidade e potencialidade da formalização adotada. O Apêndice D descreve detalhadamente o primeiro estudo de caso, apresentado no Apêndice A, utilizando a sintaxe adotada no formalismo.

Capítulo 2

Três Modelos de Objetos

A década de 80 foi rica no desenvolvimento de SBDOOs no formato de protótipos de pesquisa sendo que produtos começaram a surgir no final da mesma. As Tabelas 2.1 e 2.2 listam os SBDOOs citados nas revisões bibliográficas [ZM90b], [Vos91] e [SG92] e em artigos com análises comparativas de SBDOOs como [AWSL92] e [Loo94]. Analisando os modelos de dados em que o modelo de objetos destes sistemas são baseados, nota-se a inexistência de um modelo padrão de objetos entre os SBDOOs citados (ver Tabela 2.3), confirmando afirmações neste sentido de vários autores [Kim90, LKM90, Bee89, ABD+92]. Cada SBDOO tem seu modelo de objetos próprio herdado de uma linguagem de programação orientada a objetos (Smalltalk, C++, Common Lisp), de um modelo de dados semântico (ER, TAXIS, SDM, FDM, RM/T, Event Model, SHM+, IFO) [PM88, Hul89] ou de um modelo de objetos complexos (modelos de dados não obedecendo à primeira forma normal, modelos de conjuntos e tuplas e modelos baseados em lógica com termos complexos) [Hul89, KA90].

A década de 90 já pode ser caracterizada, na área de SBDOOs, por esforços de padronização. A falta de um modelo padrão de objetos e, conseqüentemente de uma linguagem, é apontada por W. Kim em [Kim91] como um dos pontos fracos de SBDOOs, prejudicando uma melhor aceitação destes no mercado. Existem dois trabalhos sendo desenvolvidos separadamente que propõem um modelo padrão de objetos: o SQL3 dos comitês ANSI X3H2 e ISO/IEC JTC1/SC21 [Kul94] e o ODMG-93 do grupo ODMG (*Object Database Management Group*) [Cat96]; sendo que:

- o SQL3 é uma extensão multi-volume compatível com o SQL-92. O trabalho, que aborda um sistema de tipo estendível e orientado a objetos, baseado na noção de Tipos Abstratos de Dados, ainda está em elaboração e tem sua publicação esperada

Tabela 2.1: SBDOO — Produtos.

Nome	**Desenvolvido por**	**Referência**
GemStone	Servio Logic	[MSOP86]
IDB Object Database	Persistent Data Systems	[Loo94]
ITASCA	Itasca Systems, Inc.	[ABD+91]
Kala	Penobscot Development Corp.	[Loo94]
M.A.T.I.S.S.E.	ADB/Intellitic	[Loo94]
Montage	Montage Software	[Loo94]
O2	O2 Technology	[SG92]
Objectivity/DB	Objectivity, Inc.	[SG92, Loo94]
ObjectStore	Object Design International	[LLOW91]
ODMBS	VC Software Construction GMbH	[Loo94]
ONTOS-Vbase	ONTOS, Inc.	[AH87]
OpenODB	Hewlett-Packard Company	[Loo94]
POET	POET Software	[Loo94]
UniSQL/X DBMS	UniSQL, Inc.	[Kim94]
VERSANT ODBMS	Versant	[Gor87]
VISION	Innovative Systems	[CS90]

Tabela 2.2: SBDOO — Protótipos de Pesquisa.

Nome	**Desenvolvido por**	**Referência**
Alltalk	Eastman Kodak Company	[RMS88]
CACTIS	University of Colorado	[HK86]
DAMOKLES	Forschungszentrum Informatik	[D+87]
ENCORE (Observer)	Brown University	[ZW86]
EXODUS(EXTRA-EXCESS)	University of Wisconsin	[C+90]
HiPAC	CCA[1]	[DBB+88]
Iris	Hewlett-Packard Laboratories	[FAB+89]
O2	Altaïr	[D+90]
ODE	AT&T Bells Laboratories	[AG89]
ORION	MCC[2]	[BCG+87]
OZ+	University of Toronto	[WL89]
POSTGRES	University of California - Berkeley	[SLH90]
PROBE	CCA	[DMB+90]
R2D2	IBM[3] e University of Karlsruhe	[Kem87]
ROSE	Rensselaer Polytechnic Institute	[HS88]
VODAK	GMD-IPSI[4]	[DKT88]
ZEITGEIST	Texas Instruments Incorporation	[FJL+88]

[1] Computer Corporation of America

[2] Microeletronics and Computer Technology Corporation

[3] IBM Scientific Center Heidelberg

[4] Institute for Integrated Publication and Information Systems, Darmstadt, FRG

Tabela 2.3: SBDOO — Modelos de dados adotados.

SBDOO	**Modelo baseado em**
GemStone	Smalltalk
Alltalk	Smalltalk
CACTIS	Modelo semântico
DAMOKLES	Modelo de objeto complexo
ENCORE - Observer	Modelos semânticos e Smalltalk
EXODUS	modelos semânticos, modelo de objeto complexo e LPOOs
HiPAC	PROBE
Iris	Modelos semânticos e linguagem funcional
O2	Modelo de objeto complexo
ObjectStore	C++
Objectivity/DB	C++
ODE	C++
Ontos/Vbase	C++
Orion - Itasca	Common Lisp
OZ+	Smalltalk, Actors, TADs
POSTGRES	Relacional estendido
PROBE	Modelo semântico
R2D2	Relacional estendido
ROSE	Relacional estendido
UniSQL/X DBMS	Relacional estendido
VERSANT ODBMS	C++
VISION	Especialização de Objetos [Sci89]
VODAK	Modelo de objeto complexo
ZEITGEIST	Common LISP

para 1997 [MM95].

- O padrão ODMG-93 inclui os componentes: arquitetura, modelo de objetos, linguagem de definição (ODL), linguagem de consulta (OQL) e *bindings* para as linguagens de programação C++ e Smalltalk. A primeira versão do padrão ODMG-93 foi publicada em 1993 [Cat94a] e revisada em 1995 [Cat96].

Existe pouca sobreposição entre estes trabalhos. Entretanto, estes grupos têm desenvolvido esforços para diminuir diferenças na sintaxe e semântica da linguagem de consulta entre seus padrões [Cat96].

Este capítulo tem como intuito identificar os pontos de divergência entre os modelos de objetos adotados em SBDOOs, e mostrar como estas divergências são solucionadas numa proposta de padronização de modelo de objetos. Para isto são apresentados o modelo de objeto de dois SBDOOs protótipos de pesquisa (que se tornaram produtos) e o padrão ODMG-93 (que já tem uma versão publicada e revisada).

Como preparação para este trabalho foram estudados os modelos de dados e linguagens dos SBDOOs: GemStone [MSOP86], Encore [ZW86], Iris [FAB+89], ORION [BCG+87], O2 [D+90], EXODUS [C+90] e ObjectStore [LLOW91]. Entre estes, escolheu-se incluir neste texto resumos dos protótipos de pesquisa Orion e O2, que provêem uma amostra deste estudo. Estes protótipos de pesquisa foram expressivos em contribuições para o desenvolvimento na área. O ORION, desenvolvido na *MCC (Microeletronics and Computer Technology Corporation)*, foi transformado no produto *ITASCA* [ABD+91] e o O2, desenvolvido no Altaïr, é comercializado pela *O2 Technology*. As facilidades obrigatórias que classificam um sistema como orientado a objetos, segundo a definição de SBDOO de [ABD+92, US90], serão identificadas no modelo de objeto destes sistemas.

2.1 ORION

O ORION é um protótipo de SBDOO implementado no MCC (*Microeletronics and Computer Technology Corporation, Austin - TX*) com a participação do Departamento de Ciência da Computação da Universidade do Texas. A ênfase do trabalho é integrar um sistema de programação orientado a objetos a um sistema de banco de dados; sendo assim o ORION estende a linguagem de programação orientada a objetos Common LISP com as capacidades de: persistência, armazenamento compartilhado, controle de integridade,

acesso associativo com transparência e mapeamento eficiente entre os objetos na memória virtual e secundária [KCGW88].

Foram implementadas três versões do sistema: ORION-1 (um sistema mono-usuário e multi-tarefa), ORION-1SX (um sistema cliente/servidor para redes locais) e ORION-2 (um sistema de banco de dados totalmente distribuído) [KGBW90]. As duas primeiras implementações, ORION-1 e ORION-1SX, foram distribuídas para as empresas financiadoras do projeto. O ORION-1 foi integrado ao sistema especialista PROTEUS e ao sistema POGO de desenvolvimento de interfaces gráficas. O ORION-1SX foi integrado à plataforma de desenvolvimento de *software* orientado a objetos denominada DELI também desenvolvida no MCC. O ORION-1SX está sendo utilizado como sistema de armazenamento para um ambiente experimental de programação orientada a objetos com persistência denominado DOVE desenvolvido no NCR [KGBW90].

O Modelo de Dados do ORION e a taxonomia de evolução de esquema são apresentados em [BCG+87]. O Modelo de Consultas é primeiramente apresentado em [BKK88] e depois melhorado em [Kim89] e a arquitetura do sistema é descrita em [KGBW90]. O processo de integração da linguagem de programação Common LISP e as capacidades de sistemas de banco de dados são descritos em [KCGW88]. Os conceitos de SBDOO segundo os projetistas do ORION são discutidos de uma forma generalizada em [Kim90].

2.1.1 O modelo de dados

Objeto

No ORION qualquer entidade é uniformemente modelada como um objeto. O identificador do objeto[1] é utilizado para apontar um objeto a ser extraído. Todo objeto encapsula um estado e comportamento[2]. O estado de um objeto é o valor dos atributos do objeto, e o comportamento de um objeto é um conjunto de métodos que operam sobre o estado do objeto.

Os artigos [KBC+87, KBG89] apresentam formalmente a semântica de objeto composto do modelo de objetos do ORION. Objetos compostos são utilizados como unidade de controle de integridade, de armazenamento e extração, e de controle de concorrência.

Objetos são formados de duas maneiras. Um objeto pode ser formado por um conjunto

[1] Deve-se notar o conceito de identificador de objeto presente no ORION.
[2] Deve-se notar o conceito de encapsulamento presente no ORION.

de atributos com valores, onde estes valores são por sua vez também objetos, ou então um objeto não tem atributos, apenas valor. Todo objeto é identificado com unicidade, independentemente de seu valor ou endereço. Objetos sem atributos pertencem a classes primitivas (*i.e.*, `integer`, `string`, `boolean`). Objetos que possuem atributos pertencem a classes não-primitivas. Um objeto que possui um atributo cujo valor é de uma classe não-primitiva é dito ser um objeto composto[3].

Quando o valor de um atributo é o identificador de um objeto pertencente a uma classe não-primitiva, então diz-se que este objeto referencia o objeto indicado pelo identificador e que o objeto é composto por outro objeto. Temos então um relacionamento de composição entre objeto composto e componente. O ORION estende a semântica do relacionamento de composição com o relacionamento inverso, *i.e.*, o relacionamento de *parte-de* do objeto componente em relação ao objeto composto. Desta forma se o objeto `obj4` *é-composto* pelo objeto `obj3`, então o objeto `obj3` *é-parte* do objeto `obj4`. Nem todo modelo de objetos permite representar este relacionamento inverso[4], *i.e.*, o fato de um objeto estar ciente de quais objetos ele faz parte.

Semântica de objeto complexo

Adicionalmente, o ORION acrescenta aos relacionamentos *é-composto/é-parte* o controle de dependência de existência e de compartilhamento de objetos. Para tal existem cinco tipos de referências entre objetos para a formação de objetos compostos:

referência fraca: implica apenas num relacionamento de composição do objeto composto para o objeto componente, sem nenhuma semântica extra associada a este relacionamento.

referência composta-exclusiva dependente: implica nos relacionamentos de *composição* e *participação* entre objetos composto e componente respectivamente, com dependência de existência do objeto componente para com o objeto composto e proibe que o objeto componente seja referenciado por outros objetos.

referência composta-exclusiva independente: implica nos relacionamentos de *composição* e *participação* entre objetos composto e componente respectivamente, sem de-

[3] Deve-se notar o conceito de objeto complexo presente no ORION denominado de objeto composto.

[4] O SBDOO ObjectStore oferece uma facilidade semelhante que é a possibilidade de modelar relacionamentos 1:m, m:n entre objetos.

pendência de existência do objeto componente para com o objeto composto e proibe que o objeto componente seja referenciado por outros objetos.

referência composta-compartilhada dependente: implica nos relacionamentos de *composição* e *participação* entre objetos composto e componente respectivamente, com dependência de existência do objeto componente para com o objeto composto e permitindo que o objeto componente seja referenciado por outros objetos.

referência composta-compartilhada independente: implica nos relacionamentos de *composição* e *participação* entre objetos composto e componente respectivamente, sem dependência de existência do objeto componente para com o objeto composto e permitindo que o objeto componente seja referenciado por outros objetos.

Cada tipo de referência é representado por um tipo de atributo: atributo fraco, atributo composto-exclusivo[5] (dependente ou independente) e atributo composto-compartilhado (dependente ou independente). Um objeto composto formado apenas por atributos do tipo composto-exclusivo gera uma hierarquia de objetos semelhante a uma hierarquia de partes físicas. Se o objeto composto é formado por atributos do tipo composto-compartilhado, então uma hierarquia de objetos semelhante a uma hierarquia de partes lógicas é gerada. O fato de um atributo ser dependente ou independente influencia o mecanismo de eliminação de um objeto. Atributos dependentes aninhados levam à eliminação de objetos em cascata.

Classe

Todos os objetos que compartilham o mesmo conjunto de atributos e métodos são agrupados numa classe[6]. Um objeto só pode pertencer a uma classe. O relacionamento entre um objeto e sua classe é chamado de *instância-de*. Uma classe, além de agrupar objetos, é também visualizada como um tipo abstrato de dado[7]. Desta forma, uma classe define quais atributos compõem o estado de seus objetos e implementa os métodos que definem o comportamento de seus objetos. Uma classe pode ser primitiva (`integer`, `boolean` e `string`), sendo que classes primitivas não possuem atributos. O ORION permite definir

[5] Atributos do tipo composto-exclusivo geram objetos não compartilhados, o que equivale ao conceito de valor-complexo do modelo de dados do O2 (ver Seção 2.2.2).

[6] Classe possui o significado de repositório de objetos.

[7] Classe também é vista como tipo abstrato de dados, portanto utilizada para definir tipos.

na classe valores *default* para os atributos e *valores compartilhados* por todas instâncias da classe.

No ORION, ambos os conceitos de classes e instâncias de uma classe são implementados como objetos. A interação entre objetos é feita através de passagem de mensagens; portanto, para se criar uma instância de uma classe (um objeto) deve-se enviar uma mensagem à classe.

Hierarquia de composição de classe

O valor de um atributo é um objeto; portanto, deve pertencer a uma classe. Esta classe é denominada de domínio do atributo do objeto. O domínio de um atributo pode ser qualquer classe[8], incluindo classes primitivas. O fato de que o domínio de um atributo pode ser qualquer classe resulta em estruturas aninhadas de definição de classes. Desta forma a definição de uma classe resulta num grafo dirigido de classes enraizadas na classe sendo definida; este grafo é denominado de *hierarquia de composição de classe.*

Hierarquia ou malha de classe (herança)

O conceito de hierarquia de classes ou malha (*lattice*) de classes é totalmente ortogonal ao conceito de hierarquia de composição de classe, apresentado na seção anterior. A hierarquia de classes ou malha de classes capta o relacionamento de generalização entre uma classe e um conjunto de classes especializadas a partir dela. Uma hierarquia de composição de classe não tem nada a ver com herança de atributos e métodos, ela capta o relacionamento *composição/parte-de* entre classes.

O ORION permite ao usuário derivar uma classe nova a partir de uma existente; a nova classe, chamada de subclasse, herda todos atributos e métodos da classe existente. A classe existente que originou a nova classe é chamada de super-classe. O usuário pode especificar novos atributos e métodos ou redefinir atributos e métodos herdados na nova subclasse[9]. Uma super-classe pode ter n subclasses e uma subclasse pode ter uma ou mais super-classes, permitindo assim herança múltipla[10]. Se o sistema permite herança simples, o grafo que representa o relacionamento de herança entre classes é denominado de *hierarquia de classes* e se o sistema permite herança múltipla o grafo é denominado de *malha de classes.*

[8] O ORION é estendível.
[9] Deve-se notar o conceito de sobreposição (*overriding*) presente no ORION (ver Seção 3.3.4).
[10] O ORION permite herança múltipla.

Quando uma classe tem várias super-classes, *i.e.*, possui múltipla herança, podemos ter dois tipos de conflitos nos nomes dos atributos e métodos herdados: conflito entre a classe e sua super-classe e conflito entre super-classes de uma classe. Quando o conflito entre nomes de atributos e métodos de uma classe e sua super-classe ocorre, é dada prioridade à definição dos atributos e métodos na classe. Desta forma, as definições na classe sobrepõem as herdadas com o mesmo nome. Esta é a forma de se redefinir atributos e métodos numa subclasse. Quando ocorre conflito entre nomes de atributos e métodos nas super-classes de uma classe, o sistema permite as seguintes soluções:

- considera-se a definição dos atributos e métodos da primeira super-classe na lista de super-classes da classe. ORION permite a permutação das super-classes na lista de super-classes de uma classe, podendo-se alterar a origem de herança das propriedades conflitantes,
- o usuário indica explicitamente de qual super-classe deseja herdar os atributos e métodos,
- o usuário pode dar novos nomes aos atributos e métodos conflitantes.

O ORION define uma classe chamada OBJECT que é a raiz da malha de classes. Além da classe OBJECT existem outras classes básicas:

- A classe PTYPE que define os domínios primitivos para os atributos (ou variáveis de instância).
- A classe COLLECTION que consiste de objetos que são coleções de objetos. Esta classe possui métodos de iteração sobre os objetos da coleção.
- A classe SET, subclasse da classe COLLECTION, cuja instância é um conjunto de objetos (sem duplicações). A classe possui métodos para procura, inserção de elemento no conjunto, etc.
- A classe CLASS, manipulada pelo sistema, que possui como instância todos os objetos do tipo classe.
- Para toda classe definida pelo usuário o sistema cria implicitamente uma classe SET-OF-<classe> subclasse da classe SET. Esta classe armazenará todas as instâncias

(objetos) da classe definida pelo usuário[11]. É esta classe que dá persistência aos objetos.

O ORION possui uma convenção de nomes para diferenciar o acesso a instâncias de uma classe ou as instâncias de uma classe e suas subclasses. Quando se deseja acessar as instâncias da classe e suas subclasses usa-se o nome da classe seguido por um asterisco.

2.1.2 Interface do ORION

A linguagem de programação de banco de dados do ORION é a linguagem de programação Common LISP[12] estendida em dois aspectos: nas primitivas e no modelo computacional. Foram acrescentados à linguagem comandos e formas de estruturação de dados para permitir uma modelagem semântica mais próxima do mundo real. O modelo computacional foi modificado de tal forma que o usuário não perceba a transferência de dados da memória secundária para a principal, ou vice-versa, tendo a impressão de estar trabalhando com uma memória virtual infinita [KCGW88].

2.1.3 Facilidades obrigatórias de OO presentes no ORION

O ORION possui todas as facilidades de OO classificadas como obrigatórias em [ABD+92], com os mesmos significados. Entre os conceitos de tipo e classe o ORION utiliza apenas classe tendo esta os significados de tipo abstrato de dado e repositório de objetos. O ORION pode ser classificado como totalmente orientado a objetos pois tudo é representado como um objeto, incluindo classes.

O ORION oferece controle de versões tanto para classes como para objetos. A granulação de persistência é a classe, *i.e.*, todo objeto instância de uma classe definida pelo usuário é persistente.

As principais contribuições desse trabalho são: uma taxonomia de evolução dinâmica de esquema de banco de dados [BCG+87] (muito utilizada em outros trabalhos como o O2 e GemStone) e uma proposta detalhada de um modelo de consultas considerando o potencial e as restrições dos conceitos de orientação a objetos em banco de dados [BKK88, Kim89].

[11] Implementação do conceito de classe como repositório de objetos.
[12] O ORION tem completitude computacional.

2.2 O2

O O2 é um SBDOO desenvolvido no projeto francês Altaïr. O projeto Altaïr iniciou em 1986, teve uma duração de cinco anos e foi financiado por um consórcio de empresas privadas (IN2 e BULL) e de pesquisa (INRA e LRI). A meta do projeto foi projetar e implementar um sistema de banco de dados de próxima geração.

O Modelo de Dados do O2 é definido em [LRV88] e todo o sistema (modelo, arquitetura, linguagens e ferramentas) é descrito em [D+90]. As linguagens de definição de dados e de consulta são descritas mais detalhadamente em [BCD89] e [Alt89]. As possíveis modificações de esquema do banco de dados dentro da linguagem de programação LispO2 são descritas em [Bar90].

2.2.1 Composição e organização do O2

O O2 oferece um conjunto de linguagens de programação de banco de dados (CO2, LispO2 e BasicO2), uma linguagem de definição de dados e de consulta *ad hoc*, um conjunto de ferramentas de geração de interfaces para o usuário, denominado LOOKS e um ambiente de programação, denominado OOPE.

O O2 é composto por oito módulos básicos:

- o gerenciador de memória secundária: que gerencia I/O, posicionamento de dados, índices e *buffering*;
- o gerenciador de objetos: que mapeia o modelo de objetos para a representação em memória secundária;
- o gerenciador de esquemas: que manipula as informações associadas ao esquema do banco de dados;
- o processador de linguagem: que gerencia os comandos da linguagem de definição e a compilação de programas;
- o interpretador de consultas: que interpreta as consultas utilizando os gerenciadores de esquema e objeto;
- o LOOKS: interface gráfica para o sistema de banco de dados, que gerencia a tela, mostra objetos e valores e manipula a interação com o gerenciador de objetos;

- o OOPE: um ambiente de programação que utiliza o LOOKS como interface gráfica, e
- uma interface alfa-numérica: que provê acesso direto às várias linguagens do sistema de banco de dados sem utilizar recursos gráficos.

Programar uma aplicação de banco de dados usando o O2 envolve duas etapas: definir o esquema do banco de dados e em seguida implementar os métodos que descrevem o comportamento do esquema. Definir um esquema do banco de dados consiste em definir classes, utilizando-se a linguagem de definição de dados do O2. Nesta etapa a estrutura interna dos objetos de uma classe é definida por um tipo e os comportamentos desses objetos são declarados pelas assinaturas dos métodos na classe. Classes são interligadas por relacionamentos de herança e de composição. Uma vez definidas as classes passa-se para a segunda etapa, que envolve a implementação dos métodos das classes utilizando-se uma das linguagens de programação do O2. Estas linguagens são extensões de linguagens como C (CO2), Lisp (LispO2) e Basic (BasicO2) que permitem manipulação de objetos e valores complexos segundo o modelo de objetos do O2.

O O2 oferece dois modos para se executar uma aplicação: o modo de desenvolvimento e o modo de execução. No *modo de desenvolvimento* pode-se modificar o esquema do banco de dados de forma interativa (criar, eliminar ou alterar classes, métodos e objetos e valores nomeados). Nesse modo a aplicação é interpretada. No *modo de execução* considera-se que a aplicação está completa e testada. Nesse modo a aplicação é compilada e o esquema de banco de dados congelado.

2.2.2 O modelo de dados

Objetos e valores

No O2 a informação é representada por objetos e valores. Objetos são pares *(identificador , valor)* encapsulados[13]; portanto, manipulados somente por determinados métodos. Valores são dados não encapsulados representados pelo seu próprio estado e manipulados por primitivas. Um valor é construído usando-se valores atômicos, valores estruturados e/ou identificadores de objetos (denominados valores simples ou complexos).

[13] Deve-se notar a presença dos conceitos de identidade de objeto e encapsulamento no O2.

Classes e tipos

No O2 pode-se escolher entre dois tipos de organização de informação: classes e tipos. As instâncias de uma classe são objetos e as instâncias de um tipo são valores. Tipos são construídos recursivamente aplicando-se construtores `tuple`, `list` e `set` sobre tipos atômicos ou classes. Classes utilizam tipos para descrever a estrutura de suas instâncias[14].

Classes são interligadas por relacionamentos de composição e de herança. Relacionamentos de composição entre classes são definidos pelos tipos estruturados através dos construtores (`tuple`, `list` e `set`) associados à classe. Relacionamentos de hereditariedade são baseados em mecanismos de sub-tipicidade. No O2 um tipo é subtipo de outro se, e somente se, todas as instâncias deste tipo também forem instâncias de seu super-tipo. Um tipo *tupla-estruturado* é subtipo de outro se este contém todos os atributos do super-tipo e acrescenta e/ou redefine os tipos de alguns dos atributos e/ou métodos[15] do super-tipo. Um tipo *conjunto-estruturado* "set(T)" é um subtipo de "set(T')" se e somente se T for subtipo de T'. Da mesma forma, um tipo *lista-estruturado* "list(T)" é um subtipo de "list(T')" se e somente se T for subtipo de T'.

Além de herança simples, o O2 permite herança múltipla[16] na qual um subtipo é gerado a partir de um ou mais super-tipos, podendo resultar em ambigüidades quando um atributo ou nome de método é definido em mais de um super-tipo. No O2 esta ambigüidade é tratada explicitamente de duas formas pelo usuário, que pode: redefinir o atributo ou método em um ou mais dos super-tipos ou indicar o caminho de herança.

O O2 oferece uma classe predefinida chamada *Object*. Esta classe é a raiz da hierarquia de classes definida no esquema de banco de dados. Ela implementa métodos que são comuns a todos os objetos (*e.g.*, testes de identidade e igualdade e cópia de objetos complexos).

Persistência

No O2 a persistência é associada a nomes; desta forma, objetos, valores ou conjunto de objetos ou valores podem ser nomeados (*i.e.*, uma identificação dada pelo usuário adicional à dada pelo sistema). As regras de persistência são:

- todo objeto ou valor nomeado é persistente,

[14] Deve-se notar a presença do conceito de objetos complexos, dos construtores *tuple*, *list* e *set* e da facilidade e que o O2 é estendível.

[15] Deve-se notar a presença da facilidade de sobreposição (*overriding*) no O2.

[16] Deve-se notar a presença da facilidade de herança no O2.

- todo objeto ou valor que é parte de outro objeto persistente é também persistente.

A persistência também pode ser definida na classe anexando-se à classe a cláusula `with extension`, indicando persistência de todos os objetos da classe. A extensão de uma classe é o conjunto de todos objetos criados usando-se o comando `new` aplicado à classe. Objetos de classes definidas sem a cláusula `with extension` não têm persistência, a não ser que sejam nomeados.

Métodos

Um método é um procedimento anexado a uma classe. Uma classe pode ter vários métodos associados. Os objetos da classe são manipulados pelos métodos da classe. A definição dos métodos é feita em duas etapas:

- primeiramente o método é declarado na definição do esquema do banco de dados. Essa declaração é feita pela assinatura do método, que deve conter: o nome do método, à classe a qual o método será anexada, a classe ou tipo dos parâmetros de entrada e a classe ou tipo do resultado.
- posteriormente o método é implementado utilizando-se uma linguagem de programação do O2 para definir o corpo, ou a ação associada ao método.

Métodos podem ser privados ou públicos. Métodos privados são visíveis apenas dentro da classe. Métodos públicos são visíveis por todas as classes.

Atributos e métodos excepcionais

O O2 permite associar a um objeto específico atributos e/ou métodos excepcionais, *i.e.*, não definidos na classe. Métodos excepcionais são utilizados para modelar comportamentos excepcionais ou redefinir um comportamento de um objeto específico. Métodos excepcionais podem ser associados a objetos nomeados. Atributos excepcionais podem ser associados a objetos e valores, do tipo tupla, nomeados ou não.

2.2.3 As linguagens de programação de banco de dados

As Linguagens de Programação de Banco de Dados (LPBD) são utilizadas para implementar os métodos associados às classes e programas de aplicação de banco de dados[17]. Os projetistas das LPBDs do O2 optaram por projetar uma camada orientada a objetos (camada O2) dependente do modelo de dados do O2, responsável pela: declaração de objetos e valores, manipulação de objeto (passagem de mensagens) e manipulação de valores complexos (primitivas). Esta camada é acrescentada a uma linguagem de programação hospedeira L resultando na LPBD LO2. Esta absorção da camada orientada a objetos pela linguagem de programação hospedeira L é feita de tal forma que exista uma correspondência implícita entre os tipos atômicos do O2 com o da linguagem L.

Foram implementadas as LPBDs: CO2, LispO2 e BasicO2; baseadas nas linguagens hospedeiras C, Lisp e Basic respectivamente. A LPBD LispO2 tem funcionalidades específicas para manipulação de esquemas, *i.e.*, editar um esquema de banco de dados permitindo a evolução dinâmica do esquema do banco de dados.

2.2.4 Facilidades obrigatórias de OO presentes no O2

O O2 oferece todas as facilidades de OO classificadas como obrigatórias de acordo com [ABD+92]. No O2 tanto tipo como classe são vistos como tipos abstratos de dados com uma diferença, instâncias de tipos são dados não encapsulados denominados valores e instâncias de classes são dados encapsulados denominados objetos. Toda classe utiliza um tipo para descrever sua estrutura. Desta forma, como nem tudo é objeto no O2, este deve ser caracterizado como um sistema híbrido.

As inovações apresentadas no O2 são: os níveis de granulação de persistência (por valor, objeto, conjunto de valores ou objetos e classe), a possibilidade de especificar atributos e métodos excepcionais (definição de estrutura e comportamento no objeto individualmente além da definição na classe), uma potente e completa linguagem de consulta *ad hoc* orientada a objetos [BCD89, Alt89] e o sistema de múltiplas versões do banco de dados para gerenciamento de versões de objetos [CJ92].

[17] O O2 tem completitude computacional.

Tabela 2.4: Empresas com membros votantes no grupo ODMG.

AT&T Bell Latoratories
Barry & Associates
GemStone Systems
Hewlett-Packard
O2 Technology
Object Design
Objectivity
ONTOS
POET Software
Servio Corporation
SunSoft
Versant Object Technology

2.3 O padrão ODMG-93

O padrão ODMG-93 [Cat96] foi proposto pelo grupo ODMG formado a partir de um consórcio de empresas do ramo de SBDOO (ver Tabela 2.4), que se uniram para desenvolver um padrão de interface para seus produtos. O grupo ODMG é afiliado à OMG e foi formado em 1991. O padrão ODMG-93 inclui uma arquitetura e definição de SBDOO, um modelo de objetos com linguagem de definição de dados, linguagem de consulta e interfaces padronizadas para linguagens de programação (atualmente C++ e Smalltalk) [Cat94b]. Todos os membros votantes do grupo comprometeram-se a adaptar seus produtos para o padrão; portanto, espera-se que o ODMG-93 torne-se um padrão *de facto* para a indústria.

A referência mais completa, até o momento, do padrão ODMG-93 é [Cat96]. Este documento é dirigido a implementadores de SBDOOs. O documento inclui capítulos sobre o modelo de objetos, uma linguagem de manipulação de objetos (OML — *Object Manipulation Language*) e uma linguagem de consulta (OQL — *Object Query Language*) e suas respectivas interfaces (*bindings*) para as linguagens C++ e Smalltalk. Outra referência é o artigo [LAC+93] que descreve resumidamente o modelo de objetos do padrão.

2.3.1 O modelo de objetos

O modelo de objetos do ODMG (OM/ODMG) estende o modelo de objetos da OMG (OM/OMG). A seguir enumeram-se as interseções entre estes modelos segundo [Cat96]:

Tipos, Instâncias, Interfaces :

- objetos são instâncias de tipos,
- um tipo define o comportamento e estado de suas instâncias,
- o comportamento é especificado por um conjunto de operações,
- um objeto é instância imediata de apenas um tipo,
- o tipo de um objeto é determinado estaticamente no momento em que o objeto é criado e não pode ser modificado,
- tipos são organizados em grafos de subtipo/super-tipo,
- um tipo pode ter múltiplos super-tipos, e
- super-tipos são explicitamente especificados, *i.e.*, o relacionamento de subtipo/super-tipo não é deduzido da compatibilidade de assinatura entre tipos.

Operações :

- têm assinaturas que especificam o nome da operação e os valores dos argumentos e retorno da operação,
- são definidas sobre um único tipo,
- podem receber como argumentos literais ou objetos. A passagem de parâmetro para a operação é por referência,
- são acionadas (*invoked*), e
- podem ter efeitos colaterais.

O OM/ODMG estende o OM/OMG com as seguintes funcionalidades: objetos persistentes, dois tipos de propriedades de instância (a do tipo atributo e a do tipo relacionamento), exceções em operações, consultas e transações. Como consultas são efetuadas sobre conjuntos de objetos, acrescenta-se ao conceito de tipo os conceitos de extensão e chave. Extensão do tipo é o conjunto que contém todas instâncias do tipo. Chave é uma ou mais propriedades da instância do tipo que identifica um objeto pelo seu valor (estado ou conteúdo).

A hierarquia predefinida de tipos do OM/ODMG é mostrada na Figura 2.1. A hierarquia de tipo é estendida com subtipos do tipo Object, como mostra a Figura 2.2 e com subtipos do tipo Literal, como mostra a Figura 2.3.

A diferença entre instâncias do tipo Object e do tipo Literal está em como estas são identificadas. Uma instância é um par contendo identificador e estado. No caso de instâncias do tipo Object o identificador é imutável e é diferente do estado do objeto, sendo que o estado pode ser modificado mantendo-se o identificador. No caso de instâncias do tipo Literal, o identificador e o estado são iguais. Portanto, quando modificamos o estado modificamos seu identificador o que equivale a, na verdade, não ter modificado o objeto mas estar se referindo a outro objeto. Somente os tipos em itálico nas Figuras 2.1, 2.2 e 2.3 são instanciáveis.

Um tipo pode ter uma ou mais implementações; cada implementação equivale a uma classe. A implementação de um tipo consiste de uma representação e um conjunto de métodos. Uma representação é um conjunto de estruturas de dados. Os métodos são procedimentos. Para cada operação especificada na interface do tipo existe apenas um método correspondente em cada implementação. Pode haver estruturas de dados e métodos extras na implementação não especificados na interface do tipo. Na hierarquia de tipos predefinida do OM/ODMG implementações definidas pelos usuários só são permitidas para os tipos a partir de Object.

2.3.2 Definição de tipos

A definição de um tipo é feita descrevendo-se sua interface que inclui: propriedades do tipo, propriedades das instâncias do tipo e um conjunto de operações aplicáveis às instâncias do tipo.

As propriedades do tipo podem ser: *supertype*, *extent* e *key*. A propriedade *supertype* indica em que posição da hierarquia de tipos o tipo está incluído. A propriedade *extent* indica se o tipo é instanciável e qual é o nome do conjunto que conterá todas as instâncias do tipo, *i.e.*, sua extensão. A propriedade *key* indica uma propriedade ou conjunto de propriedades de instância que a identifica univocamente pelos valores associados a estas propriedades.

As propriedades das instâncias do tipo podem ser: *attribute* e *relationship*. A propriedade *attribute* define a composição da instância. A propriedade *relationship* define um relacionamento (1:1, 1:n, n:m) da instância com outras instâncias.

Denotable-Object :

- Object
- Literal

Characteristic :

- Property
  - *Attribute*
  - *Relationship*
- *Operation*

Figura 2.1: Hierarquia de tipos do ODMG-93 — visão macro.

Object :

- *Atomic-Object*
  - *Type*
  - *Exception*
  - *Iterator*
- Structured-Object
  - Collection $< T >$
    - *Set* $< T >$
    - *Bag* $< T >$
    - *List* $< T >$
    - *Array* $< T >$
  - *Structure* $< e1 : T1 \ldots en : Tn >$

Figura 2.2: Sub-hierarquia do tipo Object.

Literal :

- *Atomic-Literal*
  - *Integer*
  - *Float*
  - *Character*
  - *Boolean*
- Structured-Literal
  - Immutable-Collection $< T >$
    - *Immutable-Set* $< T >$
    - *Immutable-Bag* $< T >$
    - *Immutable-List* $< T >$
    - *Immutable-Array* $< T >$
    - *Enumeration*
  - *Immutable-Structure* $< e1 : T1 \ldots en : Tn >$
    - *Date*
    - *Time*
    - *Timestamp*
    - *Interval*

Figura 2.3: Sub-hierarquia do tipo Literal.

A diferença entre as propriedades do tipo e de instâncias do tipo está na extensão de que estas participam. As propriedades de instância descrevem a composição das instâncias agrupadas na extensão do tipo. As propriedades do tipo descrevem a composição das instâncias agrupadas na extensão do tipo *Type*. Isto porque um tipo também é um objeto, *i.e.*, uma instância do tipo atômico *Type*. Portanto, a extensão do tipo atômico *Type* contém uma instância para cada tipo definido na hierarquia de tipos. Por sua vez, cada tipo pode ou não ter uma extensão contendo as instâncias formadas pelas propriedades de instância.

As propriedades do tipo e de instâncias do tipo e as operações aplicáveis sobre as instâncias do tipo são definidas na interface do tipo através de assinaturas.

- A *assinatura de uma propriedade de tipo* é composta pelo **tipo e nome da propriedade.**
- A *assinatura de uma propriedade de instância* também é composta pelo **tipo e nome da propriedade.**
- A *assinatura de uma operação* aplicável a uma instância é composta pelo **nome da operação, nome e tipos dos argumentos, nome e tipo dos valores de retorno e nome das exceções** (condições de erro) que a operação pode levantar.

2.3.3 O meta-modelo

Uma vez descrito o modelo de objetos do ODMG-93, é possível visualizar o seu meta-modelo. A Figura 2.4, extraída de [LAC+93], mostra o meta-modelo simplificado do ODMG-93 utilizando a notação do OMT [RBP+91]. A figura especifica as propriedades (e/ou operações) de tipos, objetos, classes, atributos, relacionamentos e operações e relacionamentos entre estes. Todo o tipo definido pelo usuário é também uma instância do tipo *Type*; portanto, é possível obter através do meta-modelo informações como: o conjunto de operações do tipo, o conjunto de propriedades do tipo, quais são seus subtipos, quais são seus super-tipos, o nome de sua extensão e sua extensão.

Pode-se notar que objetos são instâncias de classes (implementações de tipos). Entretanto, como extensões estão associadas a tipos, uma extensão pode conter objetos de implementações diferentes de um mesmo tipo.

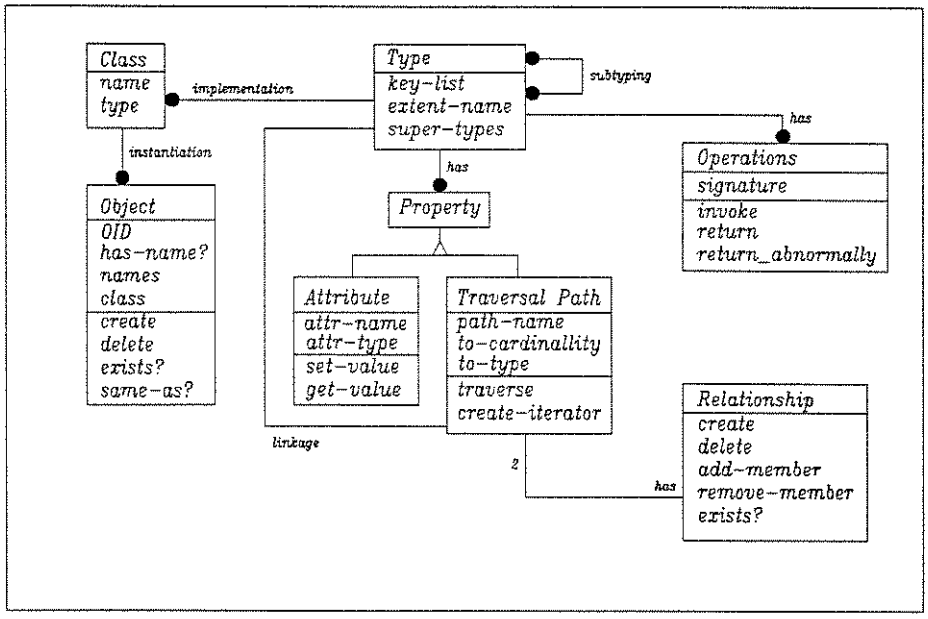

Figura 2.4: O meta-modelo simplificado do OM/ODMG.

2.4 Conclusão

2.4.1 Análise comparativa do O2 e ORION

A Tabela 2.5 mostra um resumo de como as facilidades de OO, classificadas como obrigatórias em [ABD+92, US90], estão presentes nos SBDOOs apresentados. Deve-se notar que um dos pontos de divergência entre os modelos de objetos destes SBDOOs está no significado e/ou escolha entre tipos e classes. Outro ponto de divergência está na semântica de objeto complexo que se reflete no recurso de compartilhamento de dados e controle de integridade. Daí vem a distinção entre os conceitos de objeto e valor, *i.e.*, objetos compartilham dados e valores não. SBDOOs que distinguem estes dois conceitos são denominados híbridos (*e.g.*, ObjectStore, O2 e EXODUS/EXTRA). Em contrapartida SBDOOs que utilizam apenas objetos são denominados puros (*e.g.*, ORION e GemStone). SBDOOs puros simulam o conceito de valor através de restrições de compartilhamento entre objetos (*e.g.*, ORION, ver Seção 2.1.1).

Tabela 2.5: Facilidades Obrigatórias de OO presentes nos SBDOOs apresentados.

Facilidade OO	**Orion**	**O2**
objeto complexo	puro	híbrido
identidade de objeto	sim	sim
encapsulamento	sim	sim/não
tipo(T)/classe(C)	T = C = repositório	$T \neq C$, $C_{\text{with extention}}$ = repositório
herança	múltipla	múltipla
polimorfismo e ligação tardia	sim	sim
completitude computacional	sim	sim
ser estendível	sim	sim

2.4.2 Contribuições do ODMG-93

Pode-se notar três principais contribuições do padrão ODMG-93 que tratam os conflitos encontrados entre os modelos de objetos analisados:

- Procura-se deixar bastante evidente a diferença entre tipo e classe, sendo ambos utilizados. Tipo é um conceito abstrato e classe uma implementação.
- Adota-se objeto complexo com aninhamento, existindo dois tipos de dados: o mutável (do tipo *Object*) e o imutável (do tipo *Literal*), que correspondem a objeto e valor, respectivamente. Entretanto, assume-se que ambos são encapsulados pois a aplicação é sempre descrita no nível abstrato, independente de implementação.
- Procura-se distinguir dois níveis de informação: o abstrato (intencional), que contém a definição de tipos organizados em hierarquia, e o concreto, que contém as extensões dos tipos, *i.e.*, conjuntos formados pelas instâncias dos tipos.

Uma quarta contribuição do padrão é a diferenciação de relacionamentos do tipo *é-composto-por* e de relacionamentos quaisquer com cardinalidades 1:1, 1:n e n:m. O relacionamento do tipo *é-composto-por* é caracterizado pela propriedade de instância do tipo *attribute*. Um relacionamento qualquer entre objetos é caracterizado pela propriedade de instância do tipo *relationship*. A propriedade de instância do tipo *relationship* tem funcio-

nalidades específicas que mantém a integridade deste tipo de relacionamento. Esta diferenciação tem origem em modelos semânticos.

Existem falhas no ODMG-93. Uma destas está na hierarquia predefinida de tipos que mistura tipos e geradores de tipos numa mesma hierarquia (*e.g.*, na Figura 2.2 o gerador de tipo *Collection* <*T*> é subtipo do tipo *Structured-Object*!). Os autores justificam esta abordagem como tentativa de simplificação do modelo.

2.4.3 Necessidade de um formalismo

Foi observado a partir da análise sobre o modelo de objetos de SBDOOs existentes que não existe um modelo padrão de objetos (ver Tabela 2.3). Ao mesmo tempo, viu-se que existem definições para SBDOOs [BK90, Kim90, US90] (ver Capítulo 1); entretanto, existem divergências entre estas ou pontos em aberto. Deve-se notar que as definições de [BK90] e [Kim90] indicam *classe* como uma das facilidades de OO obrigatórias em modelos de objetos para SBDOOs, e excluem *tipo*. Já a definição de [US90] especifica que é necessário *classe* ou *tipo*. Esse exemplo de divergência e/ou flexibilidade nas definições de SBDOOs dá margem ao surgimento de modelos de objetos divergentes. Duas divergências foram descritas na Seção 2.4.1. Conclui-se que é necessário não somente um modelo padrão de objetos mas também uma formalização matemática deste, que imponha correção entre implementações de um modelo compatível com o padrão. Portanto, todo padrão deve também ser formalizado. Pode-se tomar como exemplo o Sistema de Banco de Dados Relacional (SBDR), hoje amplamente adotado devido a esforços de padronização dos comitês ANSI e de formalização na álgebra relacional.

Neste contexto, este trabalho pretende completar esforços de padronização existentes propondo um modelo referência de objetos compatível com um padrão e um formalismo correspondente. O modelo referência sendo mais genérico que o padrão possibilita o mapeamento de modelos de objetos variados para o padrão. Ao mesmo tempo, o modelo referência estando de acordo com as definições de SBDOO é considerado completo.

Será adotado o padrão ODMG-93 [Cat96] como o padrão visado pelo modelo referência. Este padrão foi escolhido ao invés do SQL3 pelo fato deste segundo ser voltado a SBDRs e ainda estar em elaboração.

Será adotada a teoria de conjuntos como recurso matemático para a formalização do modelo referência. Escolheu-se a teoria de conjuntos pois visa-se como características do

formalismo: flexibilidade, uniformidade de tratamento e simplicidade de formalização.

Capítulo 3

Proposta de um Modelo Referência de Objetos para SBDOO

3.1 Introdução

Como vimos no capítulo anterior, não existe um modelo padrão de objetos, comum aos SBDOOs existentes. Esta falha tem origem na definição informal de SBDOOs. Visando uma solução para o problema, um modelo referência de objetos compatível com o padrão ODMG-93 é proposto neste capítulo e formalizado no capítulo que se segue.

O modelo referência de objetos proposto inclui todas as facilidades de OO (objetos complexos, identidade de objeto, encapsulamento, tipos e classes, herança, sobrecarga (*overloading*) e ligação tardia (*late binding*), ser estendível e completitude computacional). Sendo, portanto, considerado completo de acordo com as definições de SBDOOs em [BK90, Kim90, US90]. Ao mesmo tempo, o modelo referência adota a proposta de padronização do ODMG-93 visando o tratamento de divergências e/ou pontos em aberto das definições de SBDOO. Entretanto, o modelo referência generaliza conceitos e inclui conceitos não representados no padrão ODMG-93, de tal forma que possa mapear modelos de objetos variados para esse padrão.

3.2 Níveis de informação

Um modelo é um conjunto de abstrações com possibilidades e restrições de combinações que permitem descrever no nível intencional (abstrato) um conjunto limitado de informações

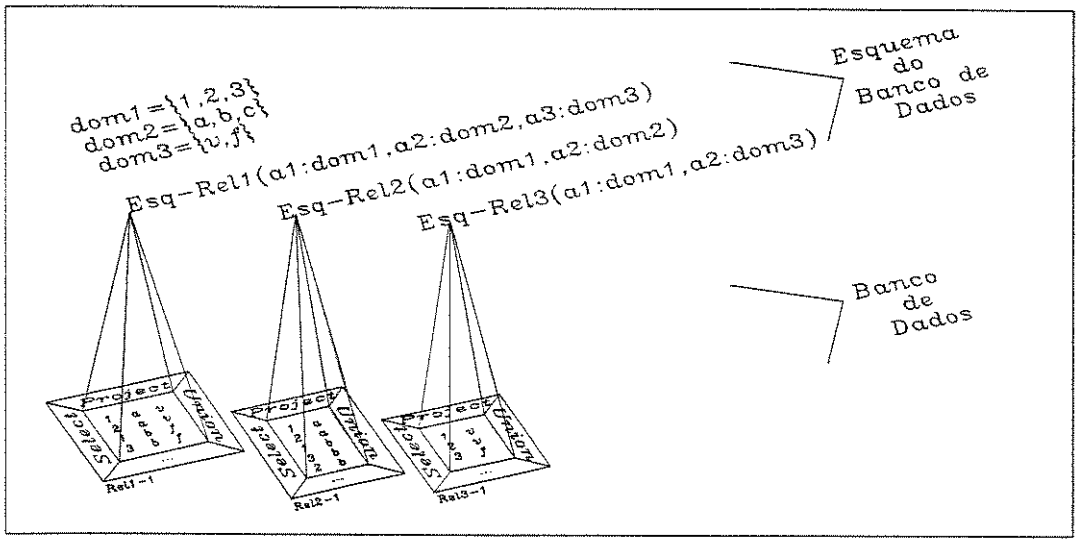

Figura 3.1: Níveis de informação num Sistema de Banco de Dados Relacional.

do nível concreto. Conseqüentemente um sistema de banco de dados, que segue um determinado modelo, manipula informações em dois níveis, o intencional (abstrato) e o concreto (dos dados), definindo uma correspondência entre ambos.

A Figura 3.1 mostra a separação desses dois níveis de informação no modelo relacional. Em Sistemas de Banco de Dados Relacionais (SBDRs) a única abstração permitida é a relação, sempre restrita a definir um novo domínio a partir do produto cartesiano de domínios básicos predefinidos. Os novos domínios são denominados esquemas de relação (*e.g.*, `Esq-Rel1`, `Esq-Rel2` e `Esq-Rel3`). Um esquema de banco de dados num SBDR é definido a partir do conjunto de esquemas de relações, sendo que o banco de dados é formado no nível concreto pelas extensões nomeadas (*e.g.*, `Rel-1`, `Rel-2` e `Rel-3`) desses esquemas de relações. Todas as extensões são manipuladas pelo mesmo conjunto de operações. Estas operações (*e.g.*, *select, project, cartesian-product, union, difference, intersection, joins* e *division*) são definidas pela álgebra relacional [Mai83].

SBDOOs suportam no nível intencional vários tipos de abstrações (*e.g.*, relação — neste trabalho estendida e renomeada para agregação[1], arranjo, potenciação) que definem novos

[1]Neste trabalho a abstração de agregação equivale à abstração de relação do modelo relacional não mais restrita a ser composta apenas por domínios básicos predefinidos e sim domínios quaisquer; portanto, também por outras relações. A abstração de relação estendida e renomeada para agregação é utilizada nesse trabalho tanto para representar relações do tipo associação qualquer (*e.g.*, pai-tem-filhos, país-tem-capital, pessoa-trabalha-em-empresa) como também associação do tipo agregação (*e.g.*, carro-é-composto-

domínios a partir de domínios quaisquer e o encapsulamento de abstrações. Este conjunto de abstrações[2] pode ser estendido. A abstração de agregação serve para descrever conjuntos heterogêneos de dados não ordenados rotulados com univocidade. A abstração de arranjo serve para descrever conjuntos homogêneos com repetição de dados indexados. A abstração de potenciação serve para descrever conjuntos homogêneos de dados não ordenados sem repetição. O encapsulamento de abstrações permite definir um novo domínio substituindo-se as operações padrão de manipulação da abstração (*i.e.*, operações que refletem a composição do domínio definido pela abstração) por um conjunto de operações que imprime ao domínio um novo comportamento, independente de sua composição. Desta forma pode-se implementar tipos abstratos de dados (*e.g.*, um domínio definido pela abstração de arranjo pode se comportar como uma pilha, lista ligada, fila e assim em diante).

Os domínios definidos pelas abstrações de agregação, arranjo e potenciação são aqui representados por grafos (veja a Figura 3.2). Para cada tipo de abstração utiliza-se um símbolo específico. O domínio novo sendo definido é o vértice raiz do grafo e os vértices intermediários e folhas do grafo especificam os domínios existentes que participam da abstração. Estes grafos são denominados grafos de composição. O domínio encapsulado é representado por um anel que envolve um domínio existente ou um grafo de composição. Este anel é formado pelos nomes das operações que definem o novo comportamento do domínio identificado no centro do anel.

Um esquema de banco de dados num SBDOO é definido por um conjunto de domínios organizados em grafos de composição (Figura 3.2) e hierarquias de herança (ver na Seção 3.3.4 a Figura 3.9). O banco de dados é formado pelo conjunto de extensões desses domínios no nível concreto. Essas extensões são manipuladas por operações do tipo: *select, project, cartesian-product, union, difference, intersection, joins* e *division* redefinidas para manipulação de conjuntos de valores ou objetos (*e.g.* como proposto em [AR91]).

Deve-se ter sempre em mente que quando se trabalha com Sistemas de Banco de Dados está-se a todo momento trabalhando com estes dois níveis de informação mencionados. Conseqüentemente, linguagens de definição de dados manipulam informações no nível intencional e linguagens de manipulação e consulta de dados manipulam informações no nível concreto. Quando em SBDOOs refere-se a tipos (e/ou classes) ou hierarquias destes está-se

por-{motor,chassis,eixo}, país-é-composto-por-estados).

[2]As abstrações de agregação e potenciação formam o conjunto mínimo de abstrações em SBDOOs que descrevem relacionamentos estruturais entre tipos e/ou classes.

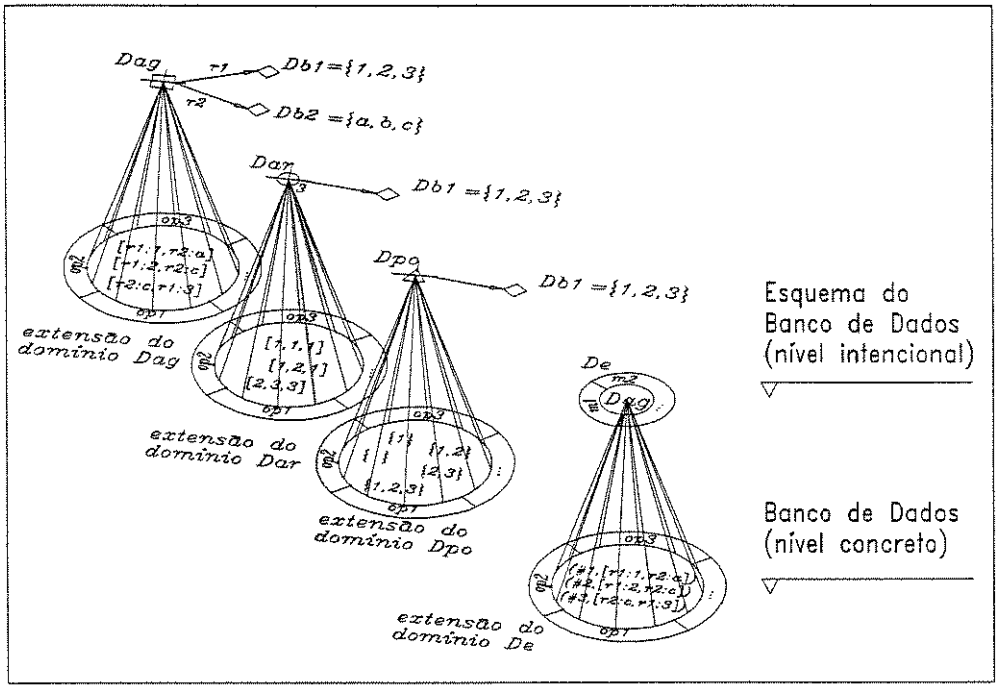

Figura 3.2: Níveis de informação num Sistema de Banco de Dados Orientado a Objetos.

referindo a informações no nível intencional. Quando se refere a dados (valores/objetos) está-se referindo a informações no nível concreto.

3.3 Nível intencional

Neste trabalho, um tipo equivale à associação de um domínio com um comportamento e uma classe equivale a implementação do comportamento associado ao domínio. Um domínio será representado por uma descrição de conjunto[3]. Um comportamento será representado por um conjunto de descrições de operações[4] (mapeamentos entre conjuntos).

Todo dado no nível concreto possui uma descrição no nível intencional, *i.e.*, uma associação domínio-comportamento. Portanto, diz-se que um dado pertence a um determinado domínio. Assim como existem categorias diferentes de dados (atômicos, compostos, encapsulados) existem categorias correspondentes de domínios. Considera-se que todas as possíveis categorias de dados pertencem a uma das seguintes categorias de domínios: básicos, com-

[3]Descrição da estrutura dos elementos do conjunto.
[4]Descrição do comportamento dos elementos do conjunto.

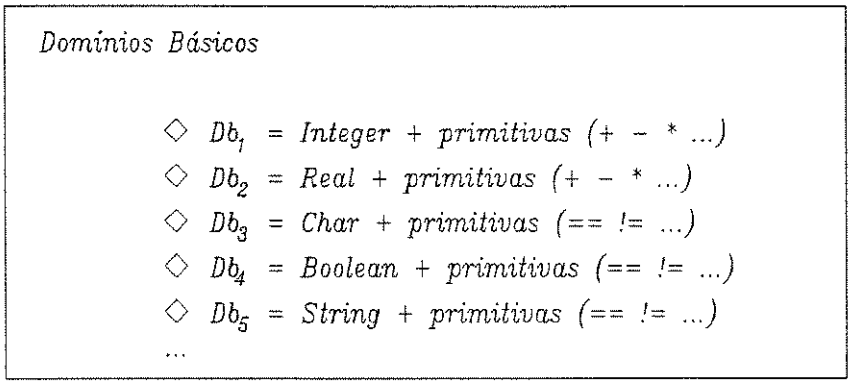

Figura 3.3: Exemplos de domínios básicos.

postos ou encapsulados.

Um esquema de banco de dados é composto pelo conjunto de associações domínios-comportamentos organizados segundo relacionamentos de composição (grafos de composição) e relacionamentos de herança (hierarquias *is-a*). As abstrações de agregação, arranjo e potenciação organizam os domínios em grafos de composição e serão apresentadas na Seção 3.3.2. Relacionamentos de herança entre domínios e comportamentos organizam estes em hierarquias de herança, *i.e.*, hierarquias *is-a*. Relacionamentos de herança serão apresentados na Seção 3.3.4.

3.3.1 Domínios básicos

Dados atômicos, *i.e.*, indivisíveis, pertencem a um número finito de domínios básicos aqui representados por $D_{b_1}, \ldots, D_{b_k}$ onde $k \geq 1$. As possíveis operações aplicáveis sobre os dados definem o comportamento destes. Todo domínio básico deve ter um único conjunto de operações que determina o comportamento dos dados no domínio. Alguns exemplos de domínios básicos associados a comportamentos específicos em linguagens de programação são os tipos *integer, real* e *boolean* (ver Figura 3.3).

3.3.2 Domínios compostos

Dados compostos por outros dados pertencem aos domínios compostos, aqui representados por $D_{ag}, D_{ar} ou D_{po}$. Três formas básicas de se combinar domínios são obtidas pelas abstrações de agregação, arranjo e potenciação. Estas abstrações são comumente represen-

tadas em linguagem de programação pelos construtores *struct/record, array* e *set* para tipos derivados ou compostos.

Estas abstrações do nível intencional levam a relacionamentos de composição entre dados do nível concreto. Esses relacionamentos são representados no nível intencional por esquemas de organização entre os domínios compostos denominados grafos de composição. Grafos de composição são representadas em forma gráfica como grafos dirigidos, em que os vértices são domínios e as arestas são relacionamentos de composição entre estes.

Domínio composto por agregação

A **abstração de agregação** de n domínios define um novo domínio a partir do produto cartesiano dos n domínios especificados. Cada elemento resultado do produto cartesiano é denominado uma n-upla. O novo domínio é dito ser composto pelos ou estar relacionado aos domínios que participam do produto cartesiano, denominados domínios componentes. A relação inversa também é verdadeira, *i.e.*, os domínios componentes são parte do ou estão relacionados ao domínio novo. Simplificando, diz-se que existem os relacionamentos *composto-por/parte-de* entre o domínio novo e os domínios componentes, respectivamente. Os domínios componentes são univocamente rotulados na agregação; conseqüentemente, dados componentes de uma n-upla do produto cartesiano também são rotulados. O comportamento padrão do domínio composto por agregação é descrito essencialmente pela operação de acesso a um componente de uma n-upla através do rótulo que identifica o componente.

A Figura 3.4 mostra uma representação gráfica do grafo de composição para um domínio composto pela abstração de agregação. Na parte (A) da figura dois domínios, D_{ag_i} e D_{ag_j}, compostos por agregação são definidos a partir de domínios existentes. O domínio D_{ag_i} é uma agregação dos domínios compostos D_{ag_j}, D_{ar_i} e D_{po_i} e do domínio básico D_{b_1} identificados na agregação pelos rótulos r_1, r_2, r_3 e r_4, respectivamente. O domínio D_{ag_j}, componente do domínio D_{ag_i}, também é composto por agregação, sendo composto pelos domínios básicos D_{b_2} e D_{b_3} identificados na agregação pelos rótulos r_1 e r_2, respectivamente. Deve-se notar que os rótulos podem ser repetidos desde que em diferentes definições de domínios. Na parte (B) da figura a definição do domínio D_{ag_j} é repetida somente para explicitar o relacionamento de composição entre o domínio composto e os domínios componentes.

Se na Figura 3.4 os domínios básicos D_{b_2} e D_{b_3} correspondessem aos conjuntos {3,4} e {a,b}, respectivamente, então o domínio D_{ag_j} definiria o conjunto {[r1:3,r2:a], [r1:3,

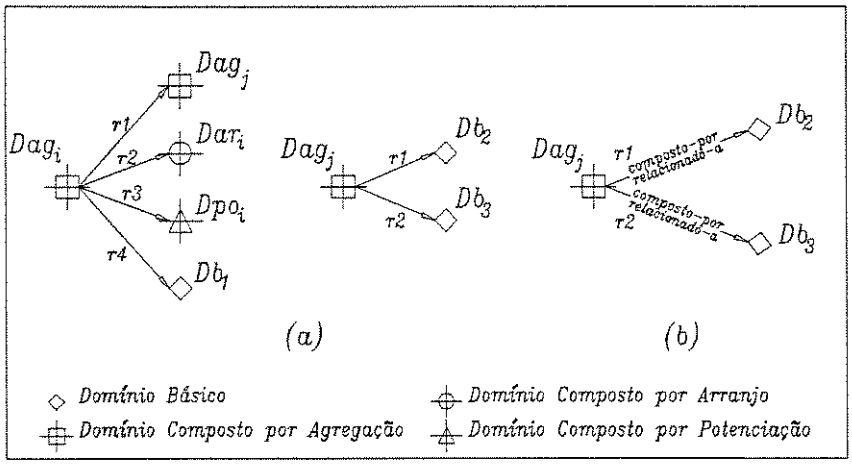

Figura 3.4: Grafos de composição de domínios compostos por agregação.

`r2:b]`, `[r1:4,r2:a]`, `[r1:4,r2:b]`}.

A Figura 3.5 mostra definições equivalentes do domínio D_{ag_i} composto por agregação, sendo que na parte (A) da figura o domínio é definido a partir de domínios existentes e na parte (B) da figura o mesmo domínio é definido por abstrações aninhadas. Uma vantagem de se definir domínios a partir de domínios previamente definidos, é que estes podem ser reutilizados na definição de outros domínios.

As arestas rotuladas no grafo de composição do domínio composto por agregação são comumente chamadas de

- atributos/variáveis de instância quando o domínio componente apontado pela aresta está fortemente acoplado ao domínio composto, ou
- relacionamentos do tipo associação qualquer ou agregação quando o domínio componente apontado pela aresta está fracamente acoplado ao domínio composto.

O acoplamento entre domínios é forte quando o domínio componente é básico ou composto e fraco quando o domínio componente é encapsulado (ver Seção 3.3.3). Quando os domínios composto e componente estão fortemente acoplados somente é possível definir o relacionamento de composição no sentido do domínio composto para o domínio componente. Quando os domínios composto e componente estão fracamente acoplados então é possível definir os relacionamentos em ambos os sentidos. Essa restrição tem origem na forma como os dados pertencentes a estes domínios são representados no nível concreto de informação, *i.e.* por

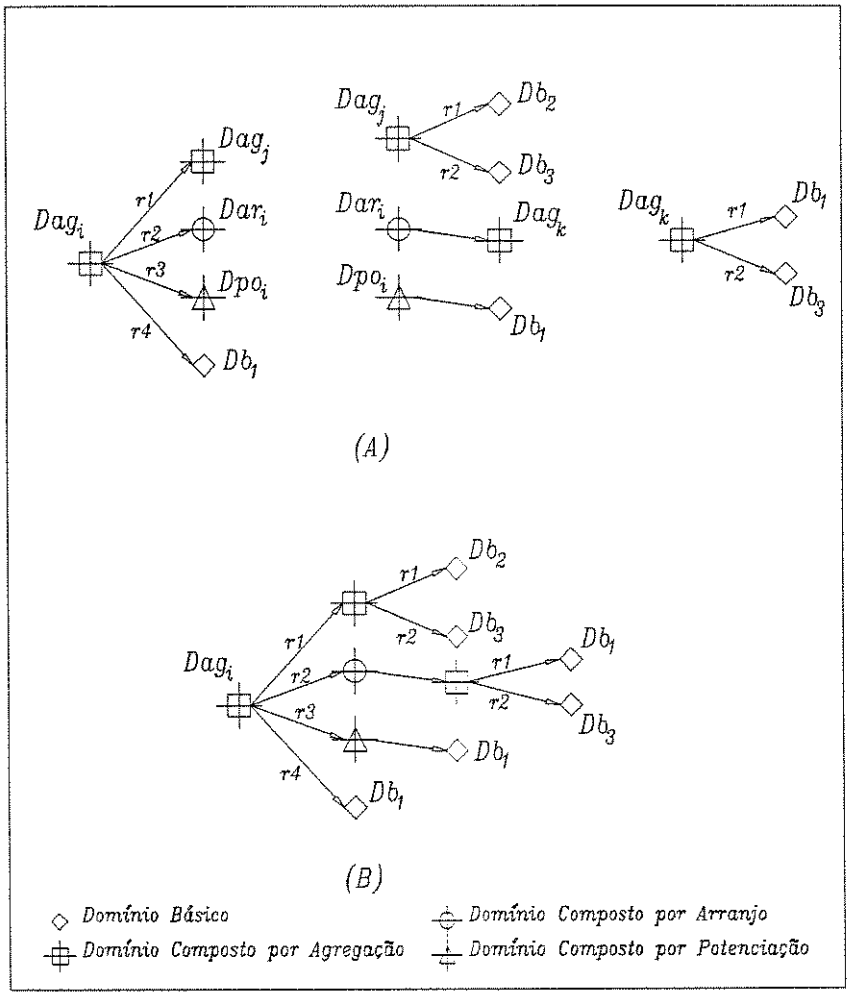

Figura 3.5: Grafos de composição de domínios sem e com abstrações aninhadas.

valor ou objeto (ver Seção 3.4).

Domínio composto por arranjo

A **abstração de arranjo** sobre um domínio existente define um novo domínio formado pela união de todos os possíveis produtos cartesianos do domínio existente original. Desta forma, cada elemento do domínio composto é um arranjo (ou uma lista homogênea de tamanho qualquer com repetição de elementos) do domínio original, denominado domínio membro. Portanto, diz-se que existe um relacionamento *arranjo-de/membro-de* entre o domínio novo e o domínio membro, respectivamente. O comportamento padrão do domínio composto por arranjo é essencialmente descrito pela operação de acesso a um elemento do arranjo pela posição (índice) deste no arranjo.

A Figura 3.6(A) mostra a definição de dois domínios, D_{ar_i} e D_{ar_j}, compostos por arranjo. O domínio D_{ar_i} define uma família de arranjos formados por elementos do domínio D_{ag_k} composto por agregação. O domínio D_{ar_j} define uma família de arranjos formados por elementos do domínio básico D_{b_1}. Deve-se notar que no grafo de composição por arranjo sempre existe apenas uma aresta partindo do domínio novo sendo definido (*e.g.*, D_{ar_i} e D_{ar_j}) para o domínio existente (*e.g.*, D_{ag_k} e D_{b_1}). Isto porque os dados compostos definidos pelo domínio composto por arranjos são homogêneos. Esta aresta representa o relacionamento de composição *arranjo-de* no sentido explicitado em forma gráfica do domínio novo para o existente. O relacionamento *membro-de* no sentido inverso não é possível de ser definido neste exemplo, pois como o domínio membro também é um domínio composto, este está fortemente acoplado ao domínio sendo definido. A Figura 3.6(B) repete a definição do domínio D_{ar_i} somente para explicitar o relacionamento de composição existente entre o domínio composto e o domínio membro.

Se na Figura 3.6 o domínio D_{b_1} correspondesse ao conjunto {1,2} então o domínio D_{ar_j} definiria o conjunto infinito { [1], [2], [1, 1], [1, 2], [2, 1], [2, 2], [1, 1, 1], [2, 2, 2], [1, 1, 2], [2, 2, 1], [1, 2, 1], ...}.

Domínio composto por potenciação

A **abstração de potenciação** sobre um domínio existente define um novo domínio formado pelo conjunto potência do domínio existente original. Cada elemento deste novo domínio é um subconjunto sem repetição de elementos do domínio existente original,

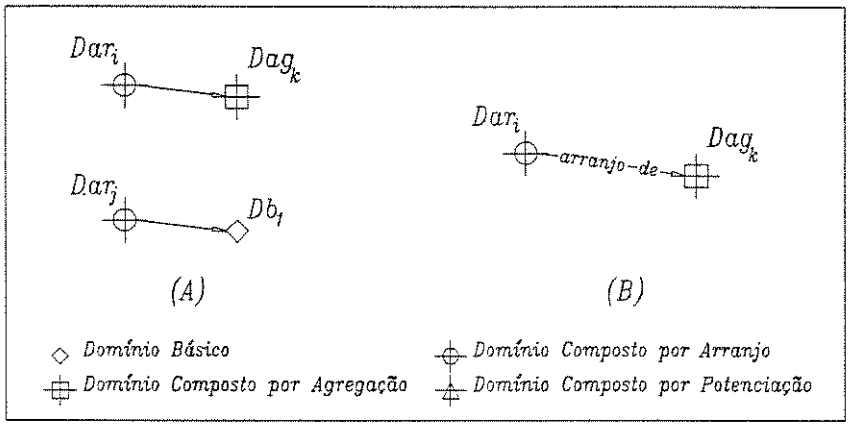

Figura 3.6: Grafos de composição de domínios compostos por arranjo.

denominado domínio membro. Portanto, diz-se que existe um relacionamento *conjunto-de/membro-de* entre o domínio novo e o domínio membro, respectivamente.

O comportamento padrão do domínio composto por potenciação é descrito pelas operações de união, interseção, diferença de conjunto e pertinência ao conjunto.

A Figura 3.7(A) mostra a definição de dois domínios compostos por potenciação D_{po_j} e D_{po_i}. O domínio D_{po_j} define um conjunto potência formado a partir do domínio D_{ag_j} composto por agregação. O domínio D_{po_i} define um conjunto potência formado a partir do domínio básico D_{b_1}. Deve-se notar que no grafo de composição por potenciação sempre vai existir apenas uma aresta partindo do domínio novo sendo definido (*e.g.*, D_{po_j} e D_{po_i}) para o domínio existente (*e.g.*, D_{ag_j} e D_{b_1}). Isto porque os conjuntos que compõem o domínio novo são homogêneos. Esta aresta representa o relacionamento de composição *conjunto-de* no sentido do domínio novo para o existente; entretanto, o relacionamento *membro-de* no sentido inverso, nesse exemplo, é impossível de ser definido, pois os domínios membros são do tipo básico ou composto. A Figura 3.7(B) repete a definição do domínio D_{po_i} somente para explicitar o relacionamento de composição existente entre o domínio composto e o domínio membro.

Se na Figura 3.7 o domínio D_{b_1} correspondesse ao conjunto {1,2} então o domínio D_{po_i} seria o conjunto { { },{1},{2},{1,2} }.

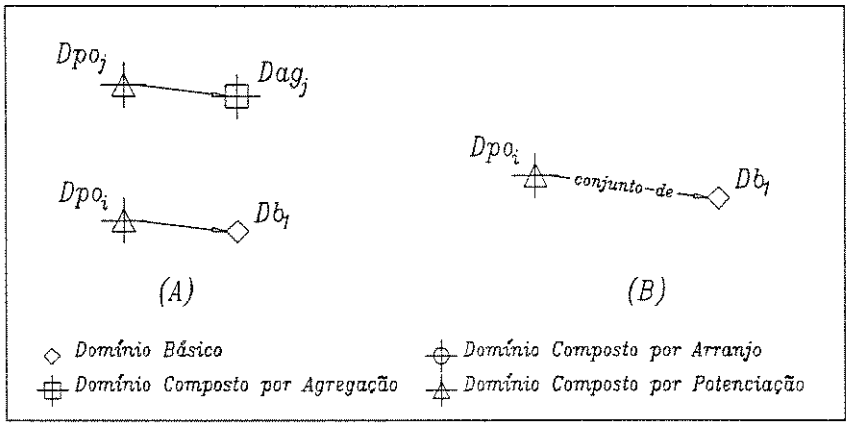

Figura 3.7: Grafos de composição de domínios compostos por potenciação.

3.3.3 Domínios encapsulados

Novos domínios podem ser definidos a partir de qualquer domínio substituindo-se seu comportamento por outro conjunto de operações que lhes imprime um novo comportamento. Este novo conjunto de operações esconde o grafo de composição do domínio original, *i.e.*, o encapsula, dando origem ao nome "domínio encapsulado".

Um domínio encapsulado é definido por um conjunto de assinaturas de operações[5], denominado interface. A assinatura de uma operação é composta pelo nome da operação e nome dos domínios dos parâmetros de entrada e resultado da operação. Um domínio encapsulado pode ter uma ou mais implementações. Uma implementação associa a interface do domínio encapsulado a uma definição de um domínio composto e ações (um corpo) para as operações[6] enumeradas na interface.

A Figura 3.8 propõe uma representação gráfica para domínios encapsulados. O encapsulamento é representado em forma gráfica envolvendo o domínio composto, que será utilizado na implementação do domínio encapsulado, por um anel formado pela interface do domínio encapsulado. Nesse exemplo a interface é representada apenas pelo nome das operações. O domínio composto que compõe o domínio encapsulado somente é conhecido internamente pelas operações na implementação do domínio. Procurou-se mostrar o conhecimento das operações sobre a composição do domínio sendo encapsulado separando ambos por um círculo tracejado.

[5] Operações no contexto da interface são denominadas na literatura de *mensagens*.
[6] Operações no contexto da implementação são denominadas na literatura de *métodos*.

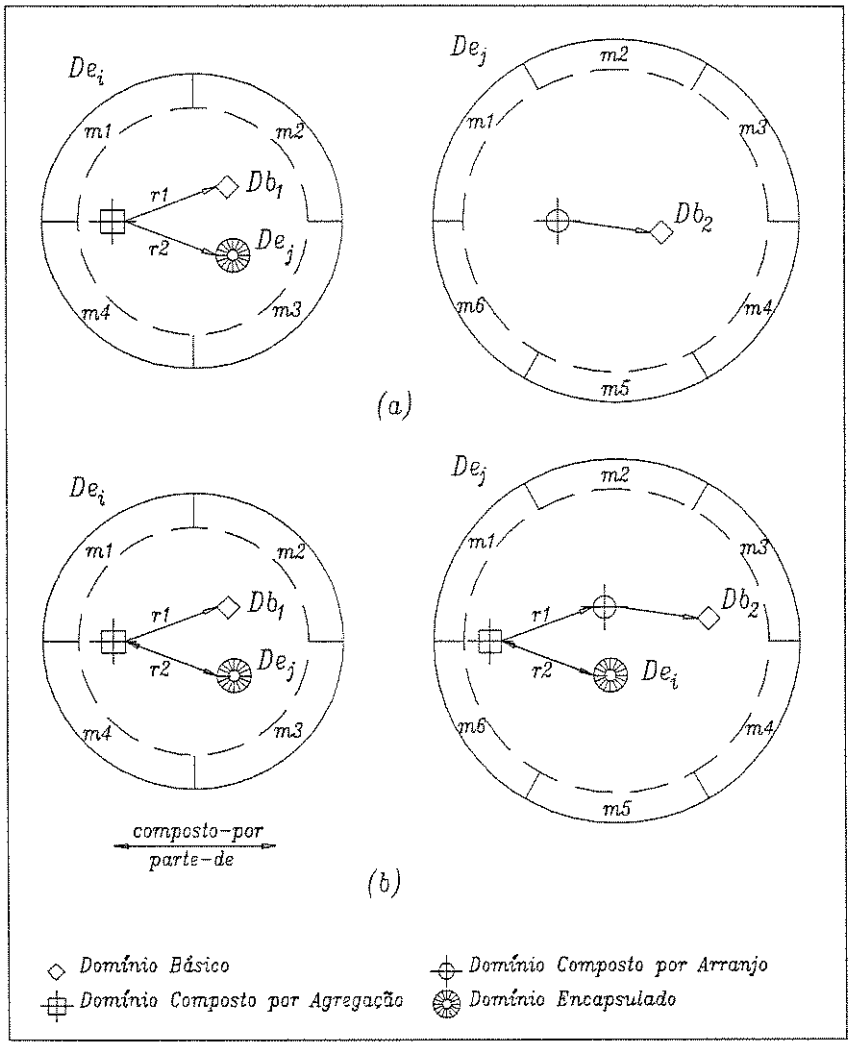

Figura 3.8: Domínios compostos e domínios encapsulados.

Nas partes (A) e (B) da Figura 3.8 os domínios encapsulados D_{e_i} e D_{e_j} são definidos, sendo que na parte (B) mostra-se como o relacionamento inverso de composição *parte-de* é definido. Deve-se notar que na parte (B) da figura a aresta r_2 especifica um relacionamento de composição em dois sentidos: do domínio composto para o componente e vice-versa. Isto somente é possível porque ambos são domínios encapsulados e porque entre ambos o relacionamento *composto-por* é definido explicitamente. Na parte (A) da figura o domínio D_{e_j} não tem conhecimento de que é parte do domínio D_{e_i}, o que implica em apenas ser possível percorrer o grafo de composição em um sentido, *i.e.*, do domínio composto para o componente. Na parte (B) da figura como o relacionamento de composição foi definido nos dois sentidos, cada domínio conhece sua composição e também está ciente de quais outros domínios fazem parte de.

3.3.4 Herança

O conceito de herança está relacionado à reutilização de definições. Com um mecanismo de herança é possível definir domínios encapsulados reutilizando definições de domínios encapsulados existentes. O novo domínio encapsulado derivado pelo mecanismo de herança é denominado subdomínio dos domínios encapsulados que o originaram (ou super-domínios). O relacionamento existente entre o subdomínio e seus super-domínios é denominado *is-a* (é-um) e é representado neste trabalho em forma gráfica como arestas pontilhadas de uma árvore ou um grafo dirigido acíclico. Quando se utiliza um mecanismo de herança simples, *i.e.*, restringe-se a derivação de um subdomínio a partir de um único super-domínio, gera-se uma árvore denominada *hierarquia is-a de domínios.* Quando se utiliza o mecanismo de herança múltipla, onde é permitindo derivar um subdomínio de múltiplos super-domínios, gera-se um grafo acíclico denominado *malha is-a de domínios.*

Existem várias combinações de transformações que derivam subdomínios de super-domínios, que vão desde transformações com regras mais livres até transformações com regras bastante rígidas, resultando em diferentes mecanismos de herança, descritos em [PS92]:

1. **derivação arbitrária**, onde no subdomínio operações podem ser adicionadas, eliminadas ou ter suas implementações redefinidas;

2. **derivação por compatibilidade de nomes**, onde o subdomínio deve possuir um conjunto de operações com os mesmos nomes com relação ao conjunto de operações

do super-domínio;

3. **derivação por conformidade de interface**, onde a interface (conjunto de assinatura de operações) do subdomínio deve ser equivalente à interface de seu super-domínio. Esta equivalência pode ser regida por duas regras de conformidade: a co-variante e a contra-variante[7];

4. **derivação monotônica**, onde no subdomínio é possível adicionar operações ou redefinir a implementação de operações herdadas;

5. **derivação comportamental**, onde a implementação das operações pode ser redefinida desde que atenda as pré- e pós-condições especificadas visando garantir a manutenção do comportamento das operações do super-domínio nas operações correspondentes do subdomínio;

6. **derivação estritamente monotônica**, onde somente é possível adicionar novas operações no subdomínio.

Deve-se notar que em todas estas variações de mecanismos de herança a derivação é feita em termos das operações (comportamento) do domínio.

A Figura 3.9 mostra uma malha *is-a* de domínios encapsulados formada segundo um mecanismo de herança múltipla com regra de derivação monotônica. Na malha *is-a* da figura o domínio encapsulado D_{e_0} é a raiz da malha onde os domínios D_{e_1}, D_{e_2} e D_{e_3} são seus subdomínios diretos, sendo que:

- o subdomínio D_{e_1} herda de D_{e_0} as operações m_1, m_2 e m_3 , e redefine o comportamento das operações m_1 e m_2 nas operações m_1' e m_2', respectivamente, e acrescenta a operação m_4;

- o subdomínio D_{e_2} herda de D_{e_0} as operações m_1, m_2 e m_3, e redefine o comportamento da operação m_1 na operação m_1', e também acrescenta as operações m_4 e m_5;

- o subdomínio D_{e_3} herda de D_{e_0} as operações m_1, m_2 e m_3, e redefine o comportamento da operação m_1 na operação m_1', e acrescenta as operações m_6 e m_7.

[7] A Seção 4.8.5 explica a diferença entre estes dois tipos de teste de equivalência entre interfaces de domínios.

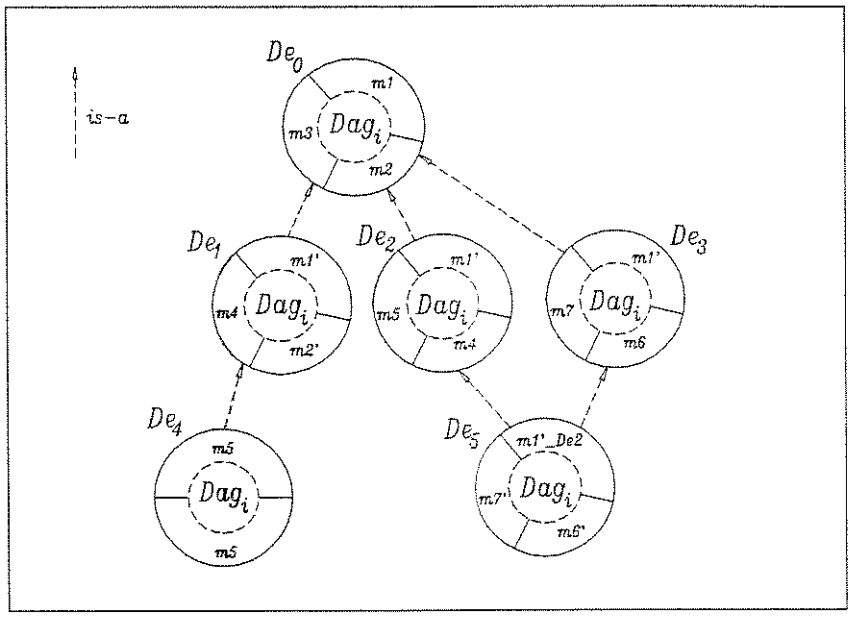

Figura 3.9: Hierarquia *is-a* de domínios encapsulados.

O recurso de redefinição, no subdomínio, da implementação de uma operação herdada é denominado sobreposição (*overriding*).

Por sua vez o domínio D_{e_4} é derivado do domínio D_{e_1} apenas acrescentando duas operações com o mesmo nome. Isto é possível desde que as assinaturas destas operações sejam diferentes. O recurso de definição de operações diferentes com o mesmo nome e assinaturas diferentes é denominado sobrecarga (*overloading*).

O domínio D_{e_5} é derivado dos domínios D_{e_2} e D_{e_3}, exemplificando um caso de herança múltipla. Na derivação de D_{e_5} nos deparamos com um conflito, já que nos super-domínios existem duas operações com o mesmo nome e provavelmente comportamentos diferentes. Esse conflito pode ser resolvido escolhendo-se de qual super-domínio a operação será herdada ou herdando ambas as operações e renomeando-as.

3.3.5 Sub-tipicidade

Mecanismos de herança definem domínios encapsulados com reutilização de definições e implementações gerando uma hierarquia *is-a* de domínios com relacionamentos de sub-tipicidade. Sub-tipicidade está relacionada com o **Princípio da Substituição**, que determina quando dados ou operações podem substituir outros dados ou operações, respec-

tivamente [Car84]. Segundo Paslberg e Schwartzbach, sub-tipicidade baseada em hierarquias/malhas *is-a* é denominada sub-tipicidade baseada em nomes [PS92]. Isso porque na definição de um subdomínio seus super-domínios são explicitamente enumerados (ou nomeados). Existe um segundo tipo de sub-tipicidade denominada estrutural, onde a relação de sub-tipicidade entre domínios não é explícita mas inferida da estrutura dos domínios.

Dependendo do mecanismo de herança utilizado para definir hierarquias *is-a* o princípio da substituição entre dados e operações varia em aplicabilidade. Na seção anterior apresentaram-se seis variações de mecanismos de herança para domínios encapsulados utilizando variados tipos de derivação: arbitrária, por compatibilidade de nomes, por conformidade de interface, monotônica, comportamental e estritamente monotônica. O mecanismo de herança que utiliza derivação estritamente monotônica é o mais perfeito em garantir o princípio da substituição entre os dados e operações da hierarquia *is-a* definida. No sentido oposto está o mecanismo de herança que utiliza derivação arbitrária que não permite o princípio da substituição entre os dados e operações da hierarquia *is-a* definida.

Portanto, neste trabalho a sub-tipicidade de domínios encapsulados:

- é definida a partir de hierarquias/malhas *is-a* e denominada sub-tipicidade baseada em nomes (pois é explicitado no sub-domínio quem são seus super-domínios) e
- pode ser utilizada por um dos seis mecanismos de herança apresentados (que enfatizam herança das operações ou do comportamento).

Como domínios encapsulados "encapsulam"(*i.e.*, são compostos por) domínios básicos ou compostos, para toda hierarquia *is-a* de domínios encapsulados existe uma hierarquia *is-a* equivalente e implícita de domínios básicos ou compostos. Implicando que para cada tipo de abstração (*e.g.*, agregação, arranjo e potenciação) deve existir uma regra de sub-tipicidade que determine quando um domínio composto é subdomínio de outro por comparação de sua composição. Portanto, conclui-se que:

- uma hierarquia/malha *is-a* de domínios encapsulados é organizada via sub-tipicidade baseada em nomes, onde o comportamento dos domínios é considerado, podendo estar ou não subentendida uma sub-tipicidade estrutural dependendo da regra de derivação utilizada.

3.3.6 Domínios parametrizados

É possível definir uma gama de domínios com comportamentos idênticos de uma única vez utilizando-se o recurso de definição de domínios parametrizados. Geralmente são utilizados para definir domínios com comportamento de coleções, *e.g.*, listas, pilhas, filas, bags, etc. Define-se a coleção genericamente, sem se especificar o domínio do elemento colecionado. Este é tratado na definição como um parâmetro. O nome do domínio parametrizado deve ser composto pelo nome do domínio e parâmetro que especifica o domínio do elemento colecionado, *e.g.*, `Lista<T>`. É na criação de um dado pertencente a este domínio que se substitui o parâmetro genérico T pelo domínio verdadeiro do elemento, *e.g.*, `Lista<integer> Li, Lista<float> Lf, Lista<Livro> Ll`.

3.3.7 Polimorfismo

As facilidades de sobrecarga[8] , sub-tipicidade, conversão entre domínios e domínios paramétricos dão uma característica polimórfica aos dados e a seu comportamento. Polimorfismo significa a capacidade de se apresentar sob inúmeras formas: ser multiforme. Cardelli e Wegner em [CW85] descrevem quatro tipos de polimorfismos em linguagem de programação com verificação de tipos (*i.e.*, fortemente tipadas):

1. polimorfismo paramétrico: comportamento (conjunto operações) uniforme sobre uma gama de domínios;
2. polimorfismo de inclusão: efeito obtido pelo Princípio da Substituição;
3. polimorfismo por sobrecarga: comportamentos diferentes com o mesmo nome, sendo o contexto que define a qual domínio cada comportamento pertence;
4. polimorfismo por coerção: dado é convertido para o domínio necessário antes de ser utilizado.

Os polimorfismos paramétrico e de inclusão são também classificados como *polimorfismo universal* e os polimorfismos por sobrecarga e por coerção são classificados como *polimorfismo ad-hoc*. O polimorfismo universal caracteriza comportamento similar sobre uma gama

[8] Com a sobreposição de operações herdadas obtém-se sobrecarga de operações numa mesma hierarquia de domínios. Também é possível sobrecarga entre operações em hierarquias diferentes. A Seção 3.3.4 inclui exemplos de ambos os casos.

de domínios que normalmente possuim uma composição em comum. O polimorfismo ad-hoc acontece entre domínios diferentes (com ou sem composição em comum) que podem ou não apresentar comportamento similar.

Este trabalho abordará somente os polimorfismos de inclusão e por sobrecarga, inicialmente tratando as formas mais expressivas de polimorfismo universal e ad-hoc.

3.4 Nível concreto

Nas seções anteriores apresentou-se como descrever domínios, comportamentos e hierarquias destes no nível intencional. Nesta seção apresentar-se-á como dados, pertencentes a estes domínios, são representados no nível concreto em termos de instâncias.

Em Sistemas de Banco de Dados Relacionais, a extensão de um esquema de relação é um subconjunto do produto cartesiano especificado no esquema de relação (Figura 3.1). Toda extensão é automaticamente persistente[9]. Desta forma o banco de dados é o conjunto de todas as extensões persistentes. São sobre estas extensões que se aplicam as consultas ao banco de dados. Portanto, linguagens de consulta manipulam informação do nível concreto.

Em Sistemas de Banco de Dados Orientados a Objetos adota-se uma abordagem mais flexível. A extensão de um domínio não é necessariamente persistente, permitindo uma granulação variável de persistência. As extensões são compostas por instâncias em forma de valores ou objetos. Objetos são dados com identidade independente de seu conteúdo. Sistemas que manipulam instâncias somente em forma de objetos são classificados como puramente orientados a objetos. Sistemas que manipulam instâncias tanto em forma de objetos quanto em forma de valores são classificados como híbridos.

3.4.1 Valor e objeto

Toda instância é um par *(identificador, estado)* onde o identificador identifica com univocidade a instância e o estado representa o conteúdo da instância. O estado pode ser um identificador ou um conjunto estruturado de identificadores. Esta estruturação é definida no nível intencional pelo domínio associado à extensão da qual à instância pertence.

Uma instância é caracterizada como um valor quando seu identificador e estado são iguais e imutáveis. Uma instância é caracterizada como um objeto quando seu identificador

[9]Na Seção 3.4.2 apresenta-se o conceito de persistência.

e estado são diferentes e o estado é mutável[10]. No caso de objetos o identificador pertence a um conjunto especial de identificadores. A seguir enumeram-se quais categorias de domínios têm extensões compostas por valores e quais têm extensões compostas por objetos:

- Domínios básicos possuem extensões compostas por valores atômicos.
- Domínios compostos possuem extensões compostas por valores simples ou complexos.
- Domínios encapsulados possuem extensões compostas por objetos atômicos, simples ou complexos.

Valores atômicos

Valores atômicos possuem estado indivisível (veja exemplos na Tabela 3.1). Instâncias pertencentes às extensões de domínios básicos são sempre valores atômicos. A Figura 3.10(A) mostra os domínios básicos e suas respectivas extensões segundo a Tabela 3.1. Deve-se notar que cada domínio básico define explicitamente todos os possíveis estados das instâncias de sua extensão.

Deve-se notar que não existe o compartilhamento de estados entre valores atômicos. Isto porque identificadores não podem ser repetidos e no caso de valores estado e identificador são iguais. Dois exemplos de valores atômicos são números e impressões digitais. Ambos são únicos e por si próprio identificados.

Valor simples

Valores simples possuem estado divisível, *i.e.*, o estado é composto por um conjunto de identificadores de valores atômicos ou valores simples. A Tabela 3.1 mostra exemplos de valores simples pertencentes aos domínios compostos D_{ag_j}, D_{ar_j} e D_{po_i}, anteriormente apresentados nas Figuras 3.4, 3.6 e 3.7 respectivamente.

[10] A diferenciação de valor e objeto adotada neste trabalho é baseada no padrão ODMG-93 [Cat94a].

Tabela 3.1: Dados e domínios correspondentes.

Nível Concreto **Dado** *(identificador, estado)*	**Nível Intencional** **Domínio**
Valor Atômico	**Domínio Básico**
(1,1) e (2,2) (3,3) e (4,4) (a,a) e (b,b)	$D_{b_1} = \{1, 2\}$ $D_{b_2} = \{3, 4\}$ $D_{b_3} = \{a, b\}$
Valor Simples	**Domínio Composto**
$([r1 : 3, r2 : b], [r1 : 3, r2 : b])$ $([r1 : 3, r2 : a], [r1 : 3, r2 : a])$	D_{ag_j} (Figura 3.10(B))
$([1, 1, 1], [1, 1, 1])$	D_{ar_j} (Figura 3.10(B))
$(\{1, 2\}, \{1, 2\})$ $(\{1\}, \{1\})$	D_{po_i} (Figura 3.10(B))
Valor Complexo	**Domínio Composto**
$([r1 : 1, r2 : \#1], [r1 : 1, r2 : \#1])$ $([r1 : 2, r2 : \#1], [r1 : 2, r2 : \#1])$ $([r1 : 1, r2 : \#2], [r1 : 1, r2 : \#2])$	D_{ag_k} (Figura 3.11)
Objeto Atômico	**Domínio Encapsulado**
$(\#1, 1)$ $(\#2, 1)$ $(\#3, 2)$	D_{e_k} (encapsula D_{b_1})
Objeto Simples	**Domínio Encapsulado**
$(\#1, [3, 3, 3])$ $(\#2, [3, 3, 3])$ $(\#3, [3, 4])$	D_{e_j} (Figura 3.11)
Objeto Complexo	**Domínio Encapsulado**
$(\#1, [r1 : [3, 3, 3], r2 : \#4])$ $(\#2, [r1 : [3, 3, 3], r2 : \#5])$ $(\#3, [r1 : [3, 4], r2 : \#4])$	D_{e_j} (Figura 3.12)
$(\#4, [r1 : 1, r2 : \#1])$ $(\#5, [r1 : 1, r2 : \#2])$ $(\#6, [r1 : 2, r2 : \#1])$	D_{e_i} (Figura 3.12)

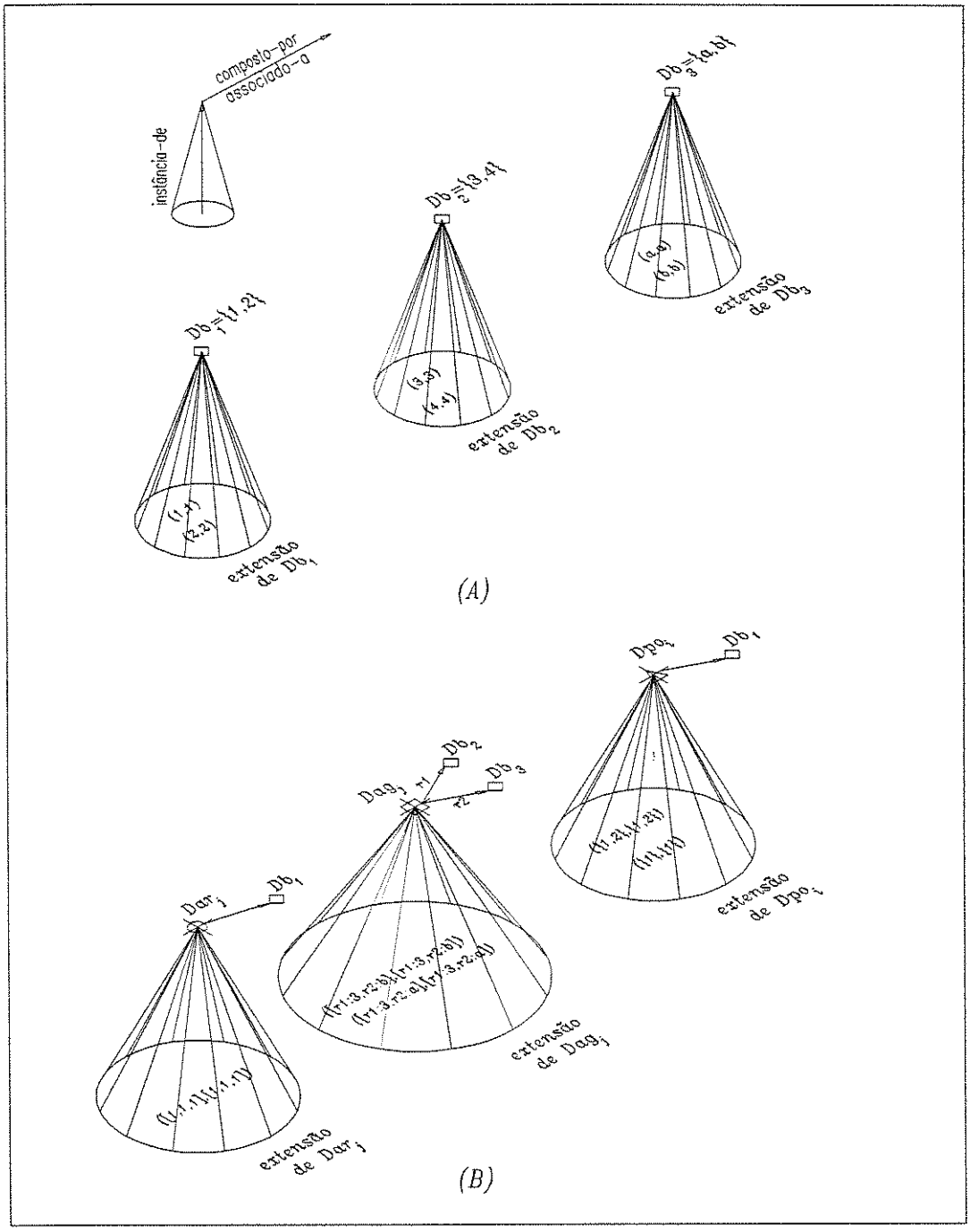

Figura 3.10: Domínios básicos e compostos e extensões com valores atômicos e simples.

A Figura 3.10(B) reapresenta os grafos de composição dos domínios compostos D_{ag_j}, D_{ar_j} e D_{po_i} acrescentando suas respectivas extensões segundo a Tabela 3.1. No nível intencional cada domínio composto é definido descrevendo-se seu grafo de composição. Esta definição representa implicitamente o conjunto de todos os possíveis estados do valor simples, que neste caso são:

- D_{ag_j} = {[r1:3, r2:a], [r1:3, r2:b], [r1:4, r2:a], [r1:4, r2:b]}
- D_{po_i} = { { },{1},{2},{1,2} }
- D_{ar_j} = { [1], [2], [1, 1], [1, 2], [2, 1], [2, 2], [1, 1, 1], [2, 2, 2], [1, 1, 2], [2, 2, 1], [1, 2, 1], [2, 1, 2], ...}

Deve-se notar que existe entre estados de valores simples o compartilhamento de identificadores. Por exemplo, o estado dos valores ([r1:3, r2:b], [r1:3, r2:b]) e ([r1:3, r2:a], [r1:3,r2:a]) compartilham o valor (3,3). Entretanto, não existe compartilhamento de modificações sobre o valor compartilhado, pois valores são imutáveis.

Um exemplo de valor simples poderia ser o documento de um carro. Por exemplo, dois carros têm cada um um documento especificando número do chassis e cor. Se os carros forem da mesma cor ambos os documentos compartilham cor. Deve-se notar que se um carro for pintado não implica na mudança de cor do outro carro. Entretanto, qualquer modificação de cor ou número de chassis implica num novo documento.

Valor complexo

Valores complexos incluem em seu estado identificadores de valores e de objetos. A Tabela 3.1 mostra exemplos de valores complexos. Neste texto adotar-se-á o símbolo # seguido de um inteiro para representar um identificador de objeto.

A Figura 3.11 mostra (segundo a Tabela 3.1): no nível intencional a definição do domínio composto D_{ag_k} e um de seus componentes (*e.g.*, o domínio encapsulado D_{e_j}), e mostra no nível concreto a extensão desses domínios. No nível intencional, o domínio D_{ag_k} representa implicitamente o seguinte conjunto infinito:

- D_{ag_k} = {[r1:1, r2:[3]], [r1:1, r2:[4]], [r1:1, r2:[3,4]], ..., [r1:2, r2:[3]], [r1:2, r2:[4]], [r1:1, r2:[3,4]], ...}

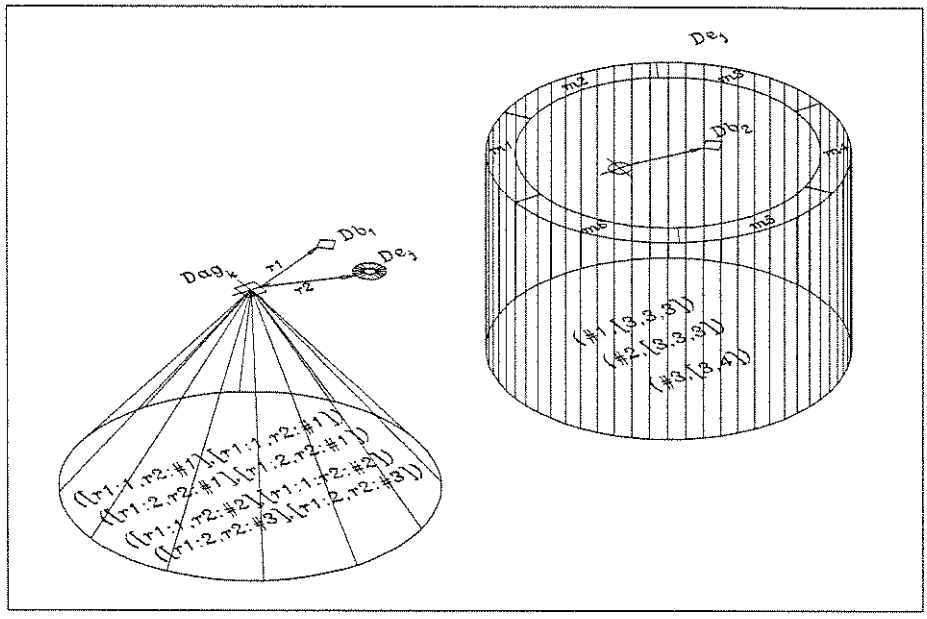

Figura 3.11: Domínio composto e extensão com valores complexos e domínio encapsulado e extensão com objeto simples.

Os elementos deste conjunto são os possíveis estados finais (*i.e.*, "desreferenciados"[11] das instâncias contidas na extensão do domínio D_{ag_k}. Por exemplo, o estado do valor `([r1:1, r2:#1], [r1:1, r2:#1])` está representado neste conjunto pelo elemento `[r1:1,r2:[3,3,3]]`.

Deve-se notar que os valores `([r1:1, r2:#1], [r1:1, r2:#1])` e `([r1:2, r2:#1], [r1:2, r2:#1])` compartilham o identificador `#1`; portanto, ambos referenciam o objeto `(#1, [3,3,3])`. Estes valores, além de compartilharem um objeto, também compartilham modificações, pois objetos são mutáveis. Por exemplo, o objeto `(#1, [3,3,3])` poderia ter seu estado `[3,3,3]` substituído por `[3,4]`. Os valores, que fazem referência a este objeto, automaticamente compartilhariam esta alteração.

Um exemplo de valor complexo é o projeto de um carro. Por exemplo, dois projetos de carros possuem especificações individuais e comuns. Modificações sobre as especificações em comum são compartilhadas entre os mesmos projetos. Modificações sobre especificações individuais implicam num novo projeto.

[11] Os identificadores de objetos no estado do objeto são substituídos pelos estados dos objetos referenciados até se obter identificadores de valores básicos ou simples.

Objeto atômico

Objetos atômicos são instâncias onde identificador e estado são desassociados e o estado é o identificador de um valor atômico. A Tabela 3.1 mostra exemplos de objetos atômicos. Identificadores não podem ser repetidos (*e.g.*, os objetos `(#1,1)` e `(#1,2)` não podem coexistir); entretanto, objetos podem compartilhar estados (*e.g.*, os objetos `(#1,1)` e `(#2,1)` são válidos).

Deve-se notar que objetos atômicos compartilham valores atômicos porém não compartilham modificações sobre valores compartilhados pois valores são imutáveis. No exemplo da Tabela 3.1 os objetos `(#1,1)` e `(#2,1)` compartilham o valor `(1,1)` mas como este valor é imutável é impossível compartilhar modificações sobre o valor compartilhado.

Um exemplo de objeto atômico poderia ser uma pessoa descrita apenas por uma característica, por exemplo seu nome. Deve-se notar que duas pessoas diferentes podem ter nomes iguais. Se o nome de uma mudar isto não implica na mudança de nome da outra pessoa com o mesmo nome. As pessoas continuam as mesmas mas com nomes diferentes, apenas o nome foi modificado.

Objeto simples

Objetos simples são objetos onde o estado é um identificador de um valor simples. A diferença entre objeto atômico e simples está na composição do estado do objeto: em objetos atômicos o esta é indivisível e em objetos simples o estado é composto. A Tabela 3.1 mostra exemplos de objetos do domínio D_{e_j} apresentados anteriormente nas Figuras 3.8(A) e 3.11.

Assim como valores simples, objetos simples compartilham valores atômicos e/ou simples; portanto, não compartilham modificações. Considere-se o exemplo dos objetos `(#1, [3,3,3])` e `(#2, [3,3,3])` que compartilham o valor simples `([3,3,3], [3,3,3])`. Como o dado compartilhado é um valor e este é imutável, não existe compartilhamento de modificações. A única diferença entre um valor simples e objeto simples é que no segundo é possível modificar o estado mantendo-se o mesmo identificador, por exemplo o objeto `(#1, [3,3,3])` pode ser alterado para `(#1, [3,4])`.

Um exemplo de objeto simples poderia ser alunos descritos por um conjunto de notas. Por exemplo, alunos são avaliados individualmente no semestre e possuem um conjunto de notas. Estes alunos podem ter um conjunto de notas iguais mas não as compartilham pois os trabalhos são individuais; portanto, se qualquer aluno for reavaliado esta alteração não

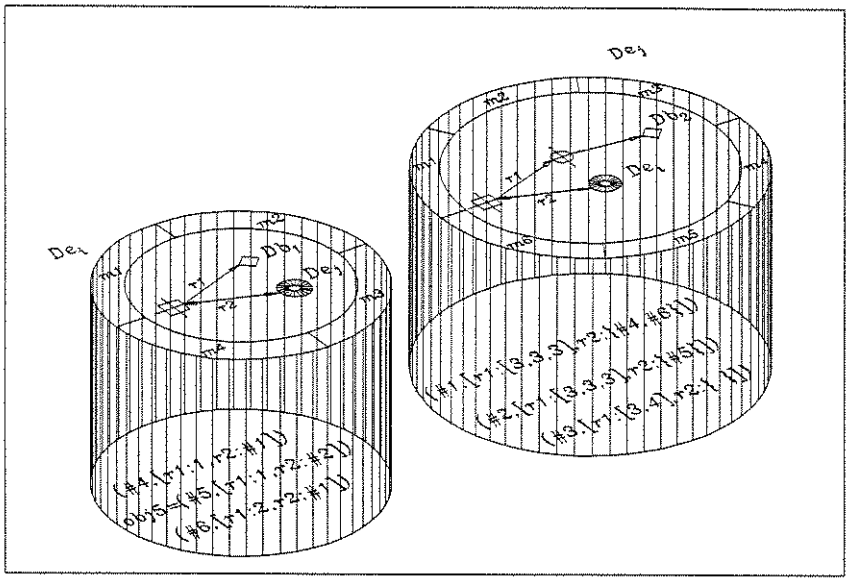

Figura 3.12: Domínios encapsulados e extensões com objetos complexos.

se propaga para outros. A modificação nas notas de um aluno não implica num novo aluno, mas sim no mesmo aluno com um novo conjunto de notas.

Objeto complexo

Objetos complexos são objetos onde o estado é um identificador de um valor complexo. A Tabela 3.1 mostra exemplos de objetos dos domínios D_{e_i} e D_{e_j} apresentados anteriormente na Figura 3.8(B).

A Figura 3.12 reapresenta os domínio D_{e_i} e D_{e_j} acrescentando extensões destes segundo dados da Tabela 3.1. Considere-se o exemplo dos objetos `(#4, [r1:1, r2:#1])` e `(#6, [r1:2, r2:#1])` ambos fazem referência ao objeto `(#1,[3,3,3])`; se o estado deste objeto for alterado o identificador permanece o mesmo e os objetos `(#4, [r1:1, r2:#1])` e `(#6, [r1:2, r2:#1])` automaticamente compartilham esta mudança.

Um exemplo de objeto complexo poderia ser duas pessoas, marido e mulher, que compartilham filhos. Ambos possuem várias propriedades, entre as quais um conjunto de filhos. Se o conjunto de filhos do casal for alterado, esta alteração deve ser também compartilhada por ambos. Além do mais, uma mudança no conjunto de filhos não implica numa nova pessoa.

É possível dar uma identificação extra a dados (valores ou objetos), além do identificador interno, através de uma nomeação pontual. A Figura 3.12 mostra um exemplo no objeto `(#5, [r1:1, r2:#2])`, que recebeu o nome `obj5`.

3.4.2 Persistência

Toda instância (objeto ou valor) tem um tempo de vida, que equivale ao tempo que esta ocupa espaço na memória (principal ou secundária[12]) de um computador. Existem vários espectros de existência, ou persistência, de instâncias:

1. existência durante a avaliação de uma expressão;
2. existência durante a execução de uma operação;
3. existência durante a execução de um programa;
4. existência entre execuções do programa;
5. existência entre várias versões do programa;
6. existência que sobrevive a programas.

Instâncias cujo tempo de vida é descrito pelas três primeiras formas de existência são denominadas *transientes* e residem na memória principal do sistema. Instâncias com o tempo de vida descrito pelas três últimas formas são denominadas *permanentes* e residem na memória secundária do sistema. Linguagens de programação enfatizam a manipulação de dados transientes deixando sob responsabilidade do programador o controle da persistência dos dados. Sistemas de bases de dados manipulam instâncias transientes e permanentes deixando transparente para o usuário o controle da persistência destas. Segundo [KA90] existem duas estratégias para identificar instâncias permanentes: por extensão e por alcance.

Persistência por extensão significa tornar qualquer instância que compõe uma extensão automaticamente permanente. Se na Figura 3.12 a extensão do domínio D_{e_i} fosse definida como permanente, então todos os objetos que compõem a extensão seriam permanentes, inclusive os objetos de outras extensões sendo referenciadas. Neste caso, todos os objetos da Figura 3.12 são permanentes exceto o objeto identificado por #3, pois não está sendo referenciado por ninguém.

[12]Por memória secundária entende-se não somente discos magnéticos, mas também memória terciária, ROMs, *i.e.*, qualquer forma de armazenagem que também seja persistente.

Persistência por alcance associa persistência a instâncias nomeadas. Se uma instância nomeada é especificada como permanente esta passa a ser uma raiz de persistência, instâncias que fazem parte desta recursivamente vão formando uma árvore de instâncias permanentes. Se na Figura 3.12 fosse adotada persistência por alcance então apenas a instância nomeada seria permanente. Conseqüentemente, os objetos identificados por `#5` e `#2` também seriam permanentes; sendo os restantes transientes.

3.4.3 Controle de integridade de dados

Um sistema de controle de integridade de dados assegura a consistência de dados reforçando restrições de integridade, que são assertivas que devem sempre ser mantidas verdadeiras [ZM90a]. Alguns tipos comuns de restrições de integridade são: de domínio, de chaves, de referência e de dependência de existência. Em muitos modelos de objetos o controle de integridade de dados está embutido na semântica de objeto complexo do modelo (ver exemplo na Seção 2.1.1).

Entretanto, ainda não se chegou a um consenso sobre a inclusão do controle de integridade entre as facilidades obrigatórias ou opcionais de um SBDOO [ABD+92]. Por esta razão, este trabalho limitar-se-á a distinguir acoplamentos forte e fraco entre domínios compostos e seus componentes tratando o compartilhamento de dados sem entrar na questão da dependência de existência de dados compartilhados.

O compartilhamento de dados será abordado da seguinte forma:

- Existe um acoplamento forte entre domínio composto e componente se o domínio componente for um domínio atômico ou composto. O acoplamento forte entre domínios no nível intencional implica no não compartilhamento de modificações sobre valores compartilhados entre instâncias destes domínios no nível concreto.

- Existe um acoplamento fraco entre domínio composto e componente se o domínio componente for um domínio encapsulado. O acoplamento fraco entre domínios no nível intencional implica no compartilhamento de modificações sobre objetos compartilhados entre instâncias destes domínios no nível concreto.

O acoplamento forte entre domínios compostos indica posse do valor compartilhado que resulta no não compartilhamento de modificações sobre o valor compartilhado e na dependência de existência do valor compartilhado. Veja o exemplo dos objetos (`#1, [3,3,3]`)

e (#2, [3,3,3]) (da Tabela 3.1 ou Figura 3.11); ambos estão fortemente acoplados ao valor ([3,3,3], [3,3,3]), que por sua vez também está fortemente acoplado ao valor (3,3). Deve-se notar que, nestes dois exemplos, existe a falsa sensação de propriedade sobre o valor compartilhado pois mesmo havendo compartilhamento, não havendo possibilidade de modificações sobre o valor compartilhado o compartilhamento não é percebido.

A característica de dependência de existência pode ser tratada de várias formas dentro de um sistema de controle de integridade, independentemente da forma de representação de acoplamento apresentado no modelo de referência. Por exemplo, pode-se optar por eliminar um objeto com dependência de existência relacionado a mais de um objeto somente quando o último objeto que o referencia é eliminado, ou pode-se optar por permitir e tratar referências a objetos inexistentes.

Num modelo de objetos dito puro todo dado é um objeto. Portanto, existindo compartilhamento de dados existe compartilhamento de modificações; caso contrário, é necessário acrescentar restrições de compartilhamento de dados. É a semântica de objeto complexo anexada ao modelo de objetos dito puro que possibilita o acréscimo deste tipo de restrição.

Igualdade de instâncias

O teste de igualdade de instâncias é necessário na criação e manipulação de instâncias, atividades que envolvem um sistema de controle de integridade. Existem vários níveis de igualdade entre instâncias: instâncias idênticas, com igualdade rasa (*shallow-equal*) e com igualdade total (*all-equal*) [KC86]. Estas diferentes nuanças de igualdade são conseqüência da distinção feita entre valor e objeto, combinada com a possibilidade de compartilhamento de objetos. Instâncias idênticas têm o mesmo identificador. Instâncias com igualdade rasa têm identificadores diferentes e estados iguais. Instâncias totalmente iguais possuem identificadores diferentes e estados iguais em todos os objetos componentes.

O teste de igualdade rasa ou total somente é possível entre valores e objetos complexos. No caso de valores atômicos ou simples o teste será sempre de identidade, pois pela definição adotada de valor nunca existirão dois valores com estados iguais e identificadores diferentes. Entretanto, isto é possível entre valores complexos desreferenciando-se os estados, por exemplo, os valores complexos ([r1:1, r2:#1], [r1:1, r2:#1]) e ([r1:1, r2:#2], [r1:1, r2:#2]) (da Figura 3.11). Estes valores são totalmente iguais não tendo igualdade rasa. Os objetos (#1, [3,3,3]) e (#2, [3,3,3]), também na mesma figura,

são totalmente iguais e têm igualdade rasa. Pode-se notar que instâncias com igualdade rasa são conseqüentemente totalmente iguais, sendo que o inverso não é necessariamente verdadeiro.

3.5 Estudos de caso

No Apêndice A são apresentados três estudos de caso, para mostrar a aplicabilidade do modelo referência proposto:

- modelando-se com os recursos do nível intencional conceitos como: classe, propriedades de classes (atributos e operações), relacionamento entre classes (associações quaisquer e do tipo agregação), hierarquia *is-a* de classes (com *sobrecargar/sobreposição* de operações) e encapsulamento; e
- obtendo-se conseqüentemente no nível concreto um conjunto de extensões compostas por instâncias que refletem um compartilhamento de dados e polimorfismo adequado às necessidades especificadas no estudo de caso.

Os estudos de caso são apresentados utilizando-se a notação gráfica apresentada neste capítulo.

3.6 Resumo

Este capítulo descreveu informalmente um modelo referência de objetos para SBDOOs. O modelo foi inspirado nos recursos de modelagem presentes nos SBDOOs e no padrão ODMG-93 descritos no Capítulo 2.

A apresentação do modelo referência iniciou-se com a descrição de abstrações para a definição de domínios. Primeiramente a abstração de relação do modelo relacional, restrita a definir um novo domínio de dados a partir do produto cartesiano de domínios básicos predefinidos, foi estendida e renomeada na abstração de agregação. A abstração de agregação proposta permite a definição de um novo domínio a partir do produto cartesiano de qualquer domínio existente.

Além da abstração de agregação, também foram propostas novas abstrações: de arranjo e de potenciação. Estas três abstrações definem domínios de dados compostos. O modelo

referência associa a todo domínio definido, por uma destas três abstrações, um comportamento padrão (operações ou funcionalidade) que reflete a forma de composição de seus dados.

Uma quarta forma de abstração, o encapsulamento, que dá origem a domínios denominados encapsulados, também foi proposta. A abstração de encapsulamento envolve associar a um ou mais domínios existentes um comportamento independente da composição dos dados envolvidos.

Uma quinta forma de definição de domínio foi finalmente apresentada: derivação por um mecanismo de herança de um novo domínio encapsulado a partir de domínios encapsulados existentes. Para tal, foram analisadas variações do mecanismo de herança. No modelo referência proposto é adotada uma combinação de derivação monotônica com compatibilidade de assinaturas do mecanismo de herança para a definição de novos domínios, denominados subdomínios, a partir de domínios existentes, denominados super-domínios.

A regra de derivação de novos domínios encapsulados a partir de domínios encapsulados existentes adotada garante a propriedade de polimorfismo por inclusão e por sobrecarga entre domínio derivado (subdomínio) e domínio original (super-domínio). Desta forma, o princípio da substituição entre dados e comportamentos do domínio derivado em relação ao domínio original é preservado, dando origem a uma hierarquia *is-a* entre estes domínios.

O comportamento associado a um domínio é definido por um conjunto de assinaturas de operações, que especifica somente a sintaxe do comportamento. A associação da semântica à sintaxe do comportamento, para um determinado domínio, é denominada implementação do comportamento (ou, simplesmente, implementação do domínio). O comportamento associado a um domínio básico ou composto é único; portanto, estes domínios têm uma única implementação. Entretanto, um domínio encapsulado pode ter múltiplas implementações; isto significa que um domínio encapsulado pode associar um único comportamento (conjunto de assinatura de operações) a múltiplos domínios.

Os seguintes conceitos foram abordados acima:

- domínio composto por qualquer domínio;
- domínio com comportamento dependente ou independente da composição;
- derivação, por um mecanismo de herança, de domínio (e comportamento) com a propriedade de polimorfismo por inclusão ou sobrecarga; e

- implementação do comportamento associado ao domínio.

Nestes conceitos as seguintes facilidades foram tratadas direta ou indiretamente no modelo referência proposto: encapsulamento, tipos, classes, herança, sobrecarga, ligação tardia e ser estendível.

A facilidade de completitude computacional não foi abordada. Essa facilidade está relacionada com a forma de se especificar a semântica do comportamento associado ao domínio. Entretanto, veremos na formalização do modelo referência, desenvolvida no próximo capítulo, que esta facilidade não foi esquecida. Esta facilidade será incluída no modelo referência de forma limitada; ou seja, sem detalhamento ou como uma "caixa preta".

Os conceitos enumerados acima estão relacionados a informações abstratas; portanto, no modelo referência, são informações do nível intencional. O modelo referência proposto também abordou informações concretas sob formas de instâncias. Instâncias são agrupadas no modelo referência em extensões no nível concreto de informações; sendo que toda extensão é associada a um domínio do nível intencional. Entretanto, o inverso não é exigido, *i.e.*, nem todo domínio do nível intencional tem associado uma extensão no nível concreto.

Instâncias foram representadas pelo par *(identificador, estado)*, sendo que:

- o identificador e estado da instância são iguais e imutáveis ou
- o estado é independente do identificador e mutável.

A primeira forma de instância foi denominada valor e a segunda objeto. Valores e objetos foram ainda subclassificados em atômicos, simples e complexos. Esta subclassificação leva em consideração a composição do estado da instância, sendo que a denominação valor ou objeto complexo reflete a capacidade de compartilhamento de modificações sobre dados compartilhados. Este tratamento de instâncias foi baseado no padrão ODMG-93. Esta abordagem tem a vantagem de não necessitar que se acrescente ao modelo de objetos uma semântica extra para o tratamento de restrições de compartilhamento entre objetos complexos.

Na abordagem dada a instâncias no modelo referência procurou-se tratar as facilidades de OO obrigatórias tais como: objeto complexo e identidade de objeto. Verifica-se finalmente que todas as facilidades de OO obrigatórias em um modelo de objetos foram tratadas e considera-se o modelo referência proposto completo com relação as definições de SBDOOs de [BK90, Kim90, US90].

A correspondência entre o modelo referência e o modelo de objetos do padrão ODMG-93 é mostrada na Tabela 3.2. Deve-se notar que no modelo referência, assim como no ODMG-93, diferenciam-se as facilidades de tipo, classe e extensão (repositório). No ODMG-93 dados são caracterizados por tipos; tipos têm interface; classes implementam tipos, sendo que tipos definidos pelo usuário podem ter múltiplas implementações (*i.e.*, múltiplas classes); e, finalmente, tipos têm extensão. No modelo referência, tipos equivalem a domínios; a interface do tipo equivale a descrição de um comportamento por um conjunto de assinaturas de operações; classes equivalem à implementação do comportamento associado ao domínio; e domínios têm extensão.

Deve-se notar também uma quase equivalência entre ambos. O modelo referência é mais genérico na descrição de assinaturas; representa assinaturas de atributos, relacionamentos e operações do ODMG-93 todas da mesma forma, *i.e.*, como assinaturas de operações. No modelo referência as assinaturas fazem parte do comportamento do domínio, sendo que no ODMG-93 estas fazem parte da interface do tipo. Conseqüentemente, o modelo referência é mais genérico, com relação ao ODMG-93, na especificação da interface do tipo. Isto permite que o modelo referência possa ser utilizado para mapear, para o padrão ODMG-93, modelos de objetos que não diferenciem atributos de relacionamentos na especificação de classes.

Entretanto, o modelo referência não é menos completo que o ODMG-93 por ser mais genérico neste contexto. O tratamento de atributos e relacionamentos (associações 1:1, 1:n, n:m ou de agregação) foram incluídos no modelo referência implicitamente na escolha de uma forma de composição de domínios (*e.g.*, agregação, arranjo ou potenciação). O Apêndice A mostra e discute exemplos.

As facilidades do ODMG-93 não abordadas no modelos referência são incluídas no Capítulo 5 na seção que discute trabalhos futuros.

Tabela 3.2: O modelo referência de objetos proposto × o padrão ODMG-93.

Modelo Referência Proposto	**ODMG-93**
instâncias como objeto/valor	instâncias como objeto/literal
caracterização de dados por domínio	caracterização de dados por tipos
comportamento do domínio: conjunto de assinaturas	interface do tipo: conjunto de assinaturas
assinatura de: operação operação operação *não abordado*	assinatura de: propriedade de instância *attribute* propriedade de instância *relationship* operação exceções
propriedade de domínio: polimorfismo por inclusão *não abordado* extensão de domínio	propriedade de tipo: *supertype* *key* extensão de tipo
implementação do comportamento do domínio	classe = implementação de tipo

Capítulo 4

Formalismo do Modelo Referência

4.1 Notação utilizada

A nomenclatura adotada neste capítulo é apresentada no Apêndice B nas Tabelas B.1, B.2, B.3, B.4, B.5, B.6 e B.7. A simbologia é apresentada na Tabela B.8.

4.2 Introdução

No capítulo anterior foi apresentado um modelo referência de objetos; entretanto, é também necessário sua fundamentação teórica. A fundamentação teórica via um formalismo matemático fornecerá recursos e parâmetros:

- para a caracterização de modelos de objetos de SBDOOs existentes; e
- que garantam a correta interpretação do modelo referência assim como do padrão ODMG-93.

Visando um formalismo simples, flexível e de tratamento uniforme, o modelo referência será formalizado pela teoria de conjuntos segundo as correspondências apresentadas na Tabela 4.1.

O ponto fundamental da abordagem aqui utilizada é assumir um domínio como um conjunto de dados associado a mapeamentos que descrevem o comportamento dos dados. Esta abordagem evoluiu das seguintes analogias: abstrações de dados são descritas por tipos e classes que, por sua vez, definem propriedades de dados. Segundo [Wan89], propriedades

Tabela 4.1: Conceitos fundamentais do formalismo representados na teoria de conjuntos.

Formalismo	**Teoria de Conjunto**
domínio	conjuntos
composições de domínios	produtos cartesianos, união e/ou permutação de produtos cartesianos, e potenciação
relação (operação)	mapeamento entre conjuntos
assinatura de uma relação	sintaxe do mapeamento
implementação de uma relação	semântica do mapeamento: uma sentença aberta, uma lei, ou um conjunto solução
implementação de comportamento de domínios	associação 1:1 ou 1:n entre conjuntos e relações
propriedade de polimorfismo de domínios e relações	inclusão de conjunto
conjunto de identificadores	conjunto associado a um conjunto de dados equivalente a um domínio
instância	par ordenado resultado da aplicação de funções específicas entre conjunto de identificadores
extensão	conjunto solução da função entre conjunto de identificadores

são representadas por predicados definidos sobre domínios. Portanto, tipos e classes caracterizam domínios por predicados, ou seja, associam a um conjunto de dados um conjunto de relações e/ou funções.

Por sua vez, em Sistemas de Banco de Dados, tipos e classes não só denotam associações entre domínios e comportamento como também extensões[1], ou seja, um conjunto de instâncias.

Instâncias serão formalizadas como pares obtidos através de mapeamentos de funções específicas entre conjuntos de identificadores associados a domínios. Esta abordagem é ba-

[1] Equivalente ao conceito de *relation extension* em [Mai83] ou *domain extent* em [Cat94a].

seada no padrão ODMG-93 [Cat94a] que diferencia instâncias em valor (*literal*) e objeto. O suporte a valores e objetos é importante pois possibilita a representação do compartilhamento ou não de modificações entre instâncias, sem a necessidade de acrescentar ao modelo de objetos semânticas extras de restrições.

Sub-tipicidade entre tipos ou classes está relacionada com a possibilidade de substituição consistente de dados ou funções. A substituição consistente entre dados na teoria de conjuntos existe entre dados que pertencem a conjuntos comparáveis. Dois conjuntos são comparáveis se houver um relacionamento de inclusão entre ambos; ou seja, para que um domínio D' seja comparável ao domínio D, o conjunto de dados que o domínio D' representa deve estar contido no conjunto de dados que o domínio D representa. Desta forma, o domínio D' será considerado um subdomínio do domínio D e qualquer dado que pertença ao domínio D' poderá substituir consistentemente um dado do domínio D.

Uma vez introduzidas as idéias que regem os conceitos a serem formalizados, inicia-se a conceituação matemática dos mesmos.

4.3 Domínios básicos

Como já introduzido na Seção 3.3.1, considera-se a existência de um número n de domínios finitos (e/ou infinitos), denominados domínios básicos definidos a seguir.

Definição 4.1 *Um domínio básico é um conjunto composto por dados atômicos.*

Um domínio básico qualquer será representado por D_{b_i}. Assim,

$$D_{b_i} = \{d \mid d \text{ é atômico}\} \tag{4.1}$$

O conceito de dado atômico, *i.e.*, indivisível, não é absoluto, depende do contexto em que está sendo utilizado. Por exemplo, para uma determinada aplicação uma imagem pode ser um dado atômico e para outra um dado composto por pixels, onde pixel é o dado atômico. Entretanto, a seguinte regra deve prevalecer: uma vez que um dado é considerado atômico dentro de um determinado contexto, neste contexto o dado é indivisível. Um exemplo de domínio básico a partir do qual qualquer outro domínio pode ser composto é o conjunto que representa o tipo *bit*.

Definição 4.2 *Dois domínios básicos com o mesmo número de dados são iguais (=) se ambos possuírem os mesmos dados.*

Axioma 4.1 (Princípio da substituição) *Um dado pode substituir outro dado se ambos pertencerem a domínios iguais.*

Veremos na Seção 4.8.1 que a restrição de igualdade no princípio da substituição apresentado acima pode ser relaxada. O princípio da substituição aplicado entre dados será representado pela propriedade de polimorfismo que será representada neste texto pelo símbolo $\preceq$; portanto:

- *Propriedade de polimorfismo de dados atômicos.*

$$d_i \preceq d_j \text{ se } d_i \in D_{b_i},\ d_j \in D_{b_j} \text{ e } D_{b_i} = D_{b_j} \tag{4.2}$$

Definição 4.3 *O conjunto de operações M_{b_i} é definido por $M_{b_i} = \{R \mid (A, B, P(a,b))\}$ onde R é uma relação de A em B sendo que:*

- *A é o domínio básico D_{b_i} ou um produto cartesiano de domínios básicos onde o domínio básico D_{b_i} é o primeiro domínio do produto,*
- *B é um domínio básico ou um produto cartesiano de domínios básicos, e*
- *$P(a,b)$ é uma sentença aberta, onde $a \in A$ e $b \in B$.*

O conjunto solução R^* da relação R é um subconjunto de $A \times B$ constituído pelos dados (a,b) em $A \times B$, para os quais a sentença aberta $P(a,b)$ é verdadeira:

$$R^* = \{(a,b) \mid a \in A, b \in A, P(a,b) \text{ é verdadeira}\}.$$

Se $(a,b) \in R^*$ então diz-se que a está relacionado com b e é denotado por $a\ R\ b$.

A relação, em M_{b_i}, onde a cada dado do domínio corresponde um único dado do contradomínio é denominada função e será expressa por $f : A \rightarrow B$, onde f é a lei que especifica o mapeamento entre os domínios:

$$f : A \rightarrow B$$

$$a \mapsto b = f(a)$$

Para cada $a \in A$ o único dado $b \in B$ tal que $(a,b) \in f$ é chamado de imagem de a e denotado por $f(a)$. O dado b é também chamado o valor que a função assume no argumento a.

Definição 4.4 *A assinatura de uma relação é composta pelo nome, domínio e contra-domínio da relação.*

Definição 4.5 *Duas assinaturas são iguais se possuírem nomes, domínios e contradomínios iguais.*

A assinatura de uma relação $R = (A, B, P(a, b))$ será representada por:

$$R : A \rightarrow B$$

sendo que a sentença aberta $P(a, b)$ que define a correspondência entre um dado a do domínio e um dado b do contradomínio não faz parte da assinatura da relação.

Da mesma forma a assinatura da função $f : A \rightarrow B$ onde $a \mapsto b = f(a)$ será representada por:

$$f : A \rightarrow B$$

sendo que também a lei $f(a)$ que define a correspondência única entre todo dado a do domínio em um dado b do contradomínio não faz parte da assinatura de uma função.

Neste trabalho interessa apenas especificar relações ou funções nos conjuntos M_{b_i} por suas assinaturas sem especificar o mapeamento específico entre o domínio e contradomínio da relação. A definição do mapeamento específico entre domínio e contradomínio da relação será incluída no conjunto $I_{M_{b_i}}$ definido a seguir e denominado conjunto implementação de relações.

Definição 4.6 *O conjunto $I_{M_{b_i}}$ define o mapeamento específico entre domínio e contra-domínio de uma relação ou função em M_{b_i} por:*

$$I_{M_{b_i}} = \{Imp(R) \mid Imp(R) = P(a, b) \text{ ou } Imp(R) = f(a)\}$$

O conjunto $I_{M_{b_i}}$ será tratado como uma "caixa preta"de tal forma que relações de igualdade e de inclusão entre conjuntos de operações considerem apenas a assinatura das operações. Para tal a Definição 4.3 de um conjunto de operações será simplificada para:

Definição 4.7 *O conjunto de operações M_{b_i} é o conjunto de assinaturas de relações $M_{b_i} = \{R \mid R : A \rightarrow B\}$ onde R é o nome de uma relação (ou função) de A em B sendo que, A é*

o domínio básico D_{b_i} ou um produto cartesiano de domínios básicos onde o domínio básico D_{b_i} é o primeiro domínio do produto e B é um domínio básico ou um produto cartesiano de domínios básicos.

Todo conjunto de operações M_{b_i} é associado a um único domínio básico D_{b_i} e a um único conjunto $I_{M_{b_i}}$. Esta associação operações-domínio equivale a impingir ao domínio um comportamento (ou funcionalidade). Esta associação será denominada implementação do comportamento de um domínio D_{b_i} e representada por:

$$Imp(M_{b_i}) = (D_{b_i}, I_{M_{b_i}}) \tag{4.3}$$

onde $I_{M_{b_i}}$ é o conjunto implementação das relações ou funções do conjunto de operações M_{b_i} para dados do domínio D_{b_i}, que neste caso são dados atômicos.

Definição 4.8 *Dois conjuntos de operações segundo a Definição 4.7 são iguais se possuírem as mesmas assinaturas.*

Portanto, $M_{b_i} = M_{b_j}$ se

$$\forall R_i \in M_{b_i} \text{ onde } R_i : A \rightarrow B$$

$$\exists R_j \in M_{b_j} \text{ onde } R_j : C \rightarrow D$$

$$\text{tal que } R_i = R_j, A = C \text{ e } B = D$$

Definição 4.9 *A sobrecarga (overlaoding) de relações existe quando duas ou mais relações têm o mesmo nome e assinaturas diferentes.*

Axioma 4.2 *Relações com sobrecarga pertencentes ao mesmo conjunto de operações têm conjunto solução igual.*

A propriedade de substituição consistente de relação por sobrecarga, é denominada polimorfismo por sobrecarga e também será representada neste texto pelo símbolo $\preceq$; portanto:

- *Propriedade de polimorfismo de relações por sobrecarga.*

$$R_i \preceq R_j \text{ se } \{R_i, R_j\} \in M_{b_i} \text{ e } R_i \text{ for uma sobrecarga de } R_j \tag{4.4}$$

Definição 4.10 *O universo $\mathcal{DB}$ de domínios básicos é a família de domínios básicos definida por*

$$\mathcal{DB} = \{D_{b_i}\}_{i \in I} \text{ onde } I = \{1, 2, \ldots, n\}$$

Definição 4.11 *O universo $\mathcal{MB}$ de operações sobre os domínios básicos é a família de conjuntos de operações definida por*

$$\mathcal{MB} = \{M_{b_i}\}_{i \in I}$$

Axioma 4.3 *A relação entre o universo $\mathcal{DB}$ de domínios básicos e o universo $\mathcal{MB}$ de operações sobre os domínios básicos é biunívoca.*

4.4 Domínios compostos por domínios básicos

Como já introduzido na Seção 3.3.2, assumem-se aqui três formas básicas de composição (agregação, arranjo e potenciação) para formar um novo domínio a partir de domínios existentes (neste caso domínios básicos). Os dados do novo domínio são dados compostos. Conseqüentemente o domínio é denominado domínio composto e, dependendo da forma de composição utilizada, será denominado domínio composto por agregação, por arranjo ou por potenciação.

O comportamento associado a estes domínios será expresso pelo conjunto de três características:

- a forma de acesso a um componente do dado composto,
- a propriedade de polimorfismo[2] do dado composto, e
- a relação de igualdade entre dados compostos.

Deve-se notar que o comportamento associado a cada categoria de domínio composto inclui a forma de acesso a um dado componente; portanto, torna explícito a composição do domínio.

[2] A Seção 3.3.7 apresenta todos os tipos de polimorfismos.

Conseqüentemente, em domínios compostos ao se conhecer seu comportamento conhece-se sua composição. Também se observa que não foram incluídas no comportamento do domínio as funcionalidades de criação e de destruição de dados, tendo em vista que as referidas funcionalidades estão relacionadas à facilidade de controle de integridade, a qual não é considerada obrigatória (ver Seção 3.4.3).

4.4.1 Domínio composto por agregação

Definição 4.12 *Um domínio composto por agregação é formado por qualquer um dos m! produtos cartesianos de m domínios básicos univocamente rotulados.*

Um domínio composto por agregação representado por D_{ag_i} e produto cartesiano de domínios básicos univocamente rotulados será representado por:

$$D_{ag_i} = \times_m \, r_j : D_{bj} \tag{4.5}$$

$$\text{onde} \;\; 1 \leq j \leq m, \;\; r_1 \neq r_2 \neq \ldots \neq r_j \neq \ldots \neq r_m, \;\; \forall j \;\; D_{b_j} \in \mathcal{DB}$$

o que equivale a

$$D_{ag_i} = r_1 : D_1 \times r_2 : D_2 \times \ldots \times r_j : D_j \times \ldots \times r_m : D_m$$

Um dado $a \in D_{ag_i}$ é uma agregação ou n-upla e será representado por:

$$a = [r_1 : d_1, \; \ldots \; , r_j : d_j, \; \ldots \; , r_m : d_m] \text{ onde } \;\; d_j \in D_{b_j} \tag{4.6}$$

Definição 4.13 *A função* `Comp` *sobre um domínio ou dado composto retorna um conjunto sem repetição de elementos formado pelos elementos que compõem o domínio ou dado composto.*

Assim,

$$\texttt{Comp}(D_{ag_i}) = \{D_{b_1}, \ldots, D_{b_j}, \ldots, D_{b_m}\} \text{ onde } \;\; D_{b_1} \neq D_{b_j} \neq \ldots \neq D_{b_m} \tag{4.7}$$

$$\text{ou } \; \texttt{Comp}(a) = \{d_1, \; \ldots \; , d_j, \; \ldots \; , d_m\} \text{ onde } \;\; d_1 \neq \ldots \neq d_j \neq \ldots \neq d_m$$

Portanto, a função `Comp` representa o relacionamento *composto-por* entre o domínio ou dado composto e seus componentes.

Definição 4.14 *A função $r_j(a)$ acessa o dado d_j rotulado por r_j na n-upla a.*

A assinatura da função $r_j(a)$ será representada por $r_j : D_{ag_i} \rightarrow D_{b_j}$. Desta forma, o nome r_j é tanto um rótulo, *i.e.*, um identificador de um domínio que participa do produto cartesiano e de um dado componente da n-upla, quanto um nome de função.

A propriedade de polimorfismo é caracterizada pelo princípio da substituição (Axioma 4.1). Como uma n-upla é um dado composto, a n-upla b pode substituir a n-upla a se os dados componentes de b puderem substituir os dados componentes correspondentes[3] de a. Assim:

$$\text{para } a = [r_1 : d_1, \ \ldots \ , r_j : d_j, \ \ldots \ , r_m : d_m]$$

$$\text{e } b = [r'_1 : d'_1, \ \ldots \ , r'_i : d'_i, \ \ldots \ , r'_n : d'_n],$$

- *Propriedade de polimorfismo de n-uplas.*

$$b \preceq a \text{ se } n \geq m, \ \forall (r_j : d_j) \ \exists_1 \ (r'_i : d'_i) \mid (r_j : d_j) \in a, \ (r'_i : d'_i) \in b, \ r_j = r'_i \text{ e } d'_i \preceq d_j \tag{4.8}$$

Definição 4.15 *Duas n-uplas do mesmo tamanho são iguais se ambas possuírem os mesmos dados.*

Assim $a = b$ se

$$n = m, \ \forall (r_j : d_j) \ \exists_1 \ (r'_i : d'_i) \mid (r_j : d_j) \in a, \ (r'_i : d'_i) \in b, \ r_j = r'_i \text{ e } d'_i = d_j.$$

Deve-se notar que a relação de igualdade é obtida de restrições sobre a propriedade de polimorfismo.

Definição 4.16 *O conjunto de operações M_{ag_i} é o conjunto de assinaturas*

$$M_{ag_i} = \{\{r_j : D_{ag_i} \rightarrow D_{b_j}\}_m, \preceq : D_{ag_i} \rightarrow D_{ag_j}, = : D_{ag_i} \rightarrow D_{ag_i}\}$$

O conjunto M_{ag_i} representa, em termos de assinatura de relações, o comportamento essencial dos dados de um domínio composto por agregação; portanto, inclui pelo menos: as funções

[3] A palavra *correspondente* é necessária pois n-uplas (ou agregações) são dados compostos onde os dados componentes não são ordenados; porém são rotulados (identificados univocamente).

de acesso aos componentes de uma n-upla (Definição 4.14), a propriedade de polimorfismo (Equação 4.8) e a relação de igualdade entre n-uplas (Definição 4.15).

A implementação do comportamento de um domínio composto por agregação será representada por:

$$Imp(M_{ag_i}) = (D_{ag_i}, I_{M_{ag_i}}) \tag{4.9}$$

onde $I_{M_{ag_i}}$ é o conjunto implementação das relações em M_{ag_i} para dados segundo a composição especificada em D_{ag_i}.

Definição 4.17 *O universo $\mathcal{DAG}$ dos domínios compostos por agregação é a família de domínios compostos por agregação definida por*

$$\mathcal{DAG} = \{D_{ag_1}, D_{ag_2}, \ldots\}$$

Definição 4.18 *O universo $\mathcal{MAG}$ dos comportamentos dos domínios compostos por agregação é a família de conjuntos de operações definida por*

$$\mathcal{MAG} = \{M_{ag_1}, M_{ag_2}, \ldots\}$$

Axioma 4.4 *A relação entre o universo $\mathcal{DAG}$ dos domínios compostos por agregação e o universo $\mathcal{MAG}$ dos comportamentos dos domínios compostos por agregação é biunívoca.*

4.4.2 Domínio composto por arranjo

Definição 4.19 *Um domínio composto por arranjo é formado pela união de todos os possíveis produtos cartesianos de um mesmo domínio básico.*

Um domínio composto por arranjo representado por D_{ar_i} e união de todos os possíveis produtos cartesianos do domínio básico D_{b_i} será representado por:

$$D_{ar_i} = \bigcup_{j=1}^{\infty} \times_j D_{b_i} \text{ onde } D_{b_i} \in \mathcal{DB} \tag{4.10}$$

ou seja

$$D_{ar_i} = \times_1 D_{b_i} \cup \times_2 D_{b_i} \cup \ldots \cup \times_\infty D_{b_i}$$

$$\text{onde} \quad \times_1 D_{b_i} = D_{b_i},\ \times_2 D_{b_i} = D_{b_i} \times D_{b_i},\ \times_3 = D_{b_i} \times D_{b_i} \times D_{b_i},\ \ldots$$

A função Comp (Definição 4.13) aplicada sobre um domínio composto por arranjo representa o relacionamento *arranjo-de* entre o domínio composto e seu domínio componente; portanto:

$$\texttt{Comp}(D_{ar_i}) = D_{b_i} \tag{4.11}$$

Um dado $a \in D_{ar_i}$ é um arranjo de tamanho qualquer e será representado por:

$$a = [d_1, \ldots, d_j] \text{ onde } d_j \in D_{b_i} \text{ e } j = 1 \ldots \infty \tag{4.12}$$

A função Comp também pode ser aplicada sobre um arranjo, ou seja:

$$\texttt{Comp}(a) = \{d_1, \ldots, d_j\} \text{ onde } d_1 \neq \ldots \neq d_j$$

Num arranjo a ordem dos dados componentes é considerada; portanto, o acesso a estes se dá pela indicação da posição destes no arranjo.

Definição 4.20 *A função $k(a)$ acessa o k-ésimo dado componente de um arranjo a de tamanho j.*

A assinatura de função $k(a)$ será representada por $k : \times_j D_{b_i} \rightarrow D_{b_i}$. Desta forma, o nome k será tanto o índice que identifica a posição de um dado componente do arranjo quanto um nome de função.

A propriedade de polimorfismo é caracterizada pelo princípio da substituição (Axioma 4.1). Como um arranjo é um dado composto, o arranjo b pode substituir o arranjo a se os dados componentes de b puderem substituir os dados componentes correspondentes[4] de a; portanto:

$$\text{para } a = [d_1, \ldots, d_k] \text{ e } b = [d'_1, \ldots, d'_l],$$

- *Propriedade de polimorfismo de arranjos.*

$$b \preceq a \text{ se } l \geq k \text{ e } \forall k,\ d'_k \preceq d_k \tag{4.13}$$

Definição 4.21 *Dois arranjos do mesmo tamanho são iguais se possuírem os mesmos elementos na mesma ordem.*

[4]Deve-se notar que a palavra *correspondente* é necessária pois arranjos são dados compostos por outros dados onde os dados componentes são ordenados.

Assim $a = b$ se

$$l = k \text{ e } \forall k,\ d'_k = d_k.$$

Definição 4.22 *O conjunto de operações M_{ar_i} é o conjunto de assinaturas*

$$M_{ar_i} = \{\{\{k : \times_j D_{b_i} \rightarrow D_{b_i}\}_{k=1}^{j}\}_{j=1}^{\infty}, \preceq: D_{ar_i} \rightarrow D_{ar_j}, =: D_{ar_i} \rightarrow D_{ar_i}\}$$

O conjunto M_{ar_i} representa, em termos de assinatura de relações, o comportamento essencial dos dados de um domínio composto por arranjo; portanto, inclui pelo menos: as funções de acesso aos componentes de um arranjo (Definição 4.20), a propriedade de polimorfismo (Equação 4.13) e a relação de igualdade entre n-uplas (Definição 4.21).

A implementação do comportamento de um domínio composto por arranjo será representada por:

$$Imp(M_{ar_i}) = (D_{ar_i}, I_{M_{ar_i}}) \tag{4.14}$$

onde $I_{M_{ar_i}}$ é o conjunto implementação das relações em M_{ar_i} para dados segundo a composição especificada em D_{ar_i}.

Definição 4.23 *O universo $\mathcal{DAR}$ dos domínios compostos por arranjo é a família de domínios compostos por arranjo definida por*

$$\mathcal{DAR} = \{D_{ar_1}, D_{ar_2}, \ldots\}$$

Definição 4.24 *O universo $\mathcal{MAR}$ dos comportamentos dos domínios compostos por arranjo é a família de conjuntos de operações definida por*

$$\mathcal{MAR} = \{M_{ar_1}, M_{ar_2}, \ldots\}$$

Axioma 4.5 *A relação entre o universo $\mathcal{DAR}$ dos domínios compostos por arranjo e o universo $\mathcal{MAR}$ dos comportamentos dos domínios compostos por arranjo é biunívoca.*

4.4.3 Domínio composto por potenciação

Definição 4.25 *Um domínio composto por potenciação é formado a partir do conjunto potência de um domínio básico.*

Um domínio composto por potenciação representado por D_{po_i} e conjunto potência de D_{b_i} será representado por:

$$D_{po_i} = 2^{D_{b_i}} \text{ onde } D_{b_i} \in \mathcal{DB} \tag{4.15}$$

A função Comp (Definição 4.13) sobre um domínio composto por potenciação representa o relacionamento *composto-por* entre o domínio composto e seu domínio componente; portanto:

$$\texttt{Comp}(D_{po_i}) = D_{b_i} \tag{4.16}$$

Um dado $a \in D_{po_i}$ é um conjunto e será representado por:

$$a = \{d_1, \ldots, d_j, \ldots, d_k\} \tag{4.17}$$

onde

$$k = 0, \ldots, cardinalidade(D_{b_i}),$$

$$d_1 \neq d_2 \neq \ldots \neq d_k$$

$$\text{e } \forall k, d_k \in D_{b_i}$$

Deve-se notar que se $k = 0$ então $a = \{\emptyset\}$.

A função Comp também pode ser aplicada sobre um conjunto; entretanto, retornará o próprio conjunto pois segundo a Equação 4.17 o conjunto não possui repetição de dados, ou seja:

$$\texttt{Comp}(a) = a = \{d_1, \ \ldots \ , d_j, \ \ldots, d_k\} \text{ onde } \ d_1 \neq d_2 \neq \ldots \neq d_k$$

Como num conjunto a ordem de seus dados não é considerada, a única consulta plausível sobre um componente do conjunto é a de pertinência.

Definição 4.26 *A função contem(a, d_i) retorna verdadeiro se o dado d_i pertence ao conjunto a, caso contrário retorna falso.*

A assinatura da função *contem*(a, d_i) será representada por *contem* : $D_{po_i} \times D_{b_i} \rightarrow \{v, f\}$.

A propriedade de polimorfismo é caracterizada pelo princípio da substituição (Axioma 4.1). Como um conjunto é um dado composto, o conjunto b pode substituir o conjunto a se os dados de b puderem substituir os dados de a. Assim

$$\text{para } a = \{d_1, d_2, \ldots, d_k\} \text{ e } b = \{d'_1, d'_2, \ldots, d'_l\}$$

- *Propriedade de polimorfismo de conjuntos.*

$$b \preceq a \text{ se } \forall d_i \exists_1 d'_j \text{ tal que } d'_j \preceq d_i \tag{4.18}$$

Definição 4.27 *Dois conjuntos do mesmo tamanho são iguais se possuírem os mesmos dados.*

Assim $a = b$ se

$$k = l \text{ e } \forall d_i \exists_1 d'_j \text{ tal que } d'_j = d_i.$$

Definição 4.28 *O conjunto de operações M_{po_i} é o conjunto de assinaturas*

$$M_{po_i} = \{contem : D_{po_i} \times D_{b_i} \rightarrow \{v, f\}, \preceq : D_{po_i} \rightarrow D_{po_j}, = : D_{po_i} \rightarrow D_{po_i}\}$$

O conjunto M_{po_i} representa, em termos de assinatura de relações, o comportamento essencial dos dados de um domínio composto por potenciação; portanto, inclui pelo menos: as funções de pertinência de um dado no conjunto (Definição 4.26), a propriedade de polimorfismo (Equação 4.18) e a relação de igualdade entre conjuntos (Definição 4.27).

A implementação do comportamento de um domínio composto por potenciação será representada por:

$$Imp(M_{po_i}) = (D_{po_i}, I_{M_{po_i}}) \tag{4.19}$$

onde $I_{M_{po_i}}$ é o conjunto implementação das relações em M_{po_i} para dados segundo a composição especificada em D_{po_i}.

Definição 4.29 *O universo $\mathcal{DPO}$ dos domínios compostos por arranjo é a família de domínios compostos por potenciação definida por*

$$\mathcal{DPO} = \{D_{po_1}, D_{po_2}, \ldots\}$$

Definição 4.30 *O universo $\mathcal{MPO}$ dos comportamentos dos domínios compostos por potenciação é a família de conjuntos de operações definida por*

$$\mathcal{MPO} = \{M_{po_1}, M_{po_2}, \ldots\}$$

Axioma 4.6 *A relação entre o universo $\mathcal{DPO}$ dos domínios compostos por potenciação e o universo $\mathcal{MPO}$ dos comportamentos dos domínios compostos por potenciação é biunívoca.*

4.4.4 Universo dos domínios compostos

Definição 4.31 *O universo dos domínios compostos é a união dos universos dos domínios compostos por agregação, arranjo e potenciação.*

O universo dos domínios compostos será representado por $\mathcal{DC}$. Assim,

$$\mathcal{DC} = \mathcal{DAG} \cup \mathcal{DAR} \cup \mathcal{DPO} \tag{4.20}$$

Um domínio qualquer do universo dos domínios compostos será representado por D_{c_i}. Assim,

$$D_{c_i} \in \mathcal{DC} \tag{4.21}$$

D_{c_i} pode ser um domínio composto por agregação, arranjo ou potenciação, ou seja

$$D_{c_i} = D_{ag_i} \text{ ou } D_{c_i} = D_{ar_i} \text{ ou } D_{c_i} = D_{po_i}$$

Definição 4.32 *O universo dos comportamentos dos domínios compostos é a união dos universos de comportamentos dos domínios compostos por agregação, arranjo e potenciação.*

O universo dos comportamentos dos domínios compostos será representado por $\mathcal{MC}$. Assim,

$$\mathcal{MC} = \mathcal{MAG} \cup \mathcal{MAR} \cup \mathcal{MPO} \tag{4.22}$$

Um comportamento qualquer do universo dos comportamentos dos domínios compostos será representado por M_{c_I}. Assim,

$$M_{c_i} \in \mathcal{MC} \tag{4.23}$$

M_{c_i} pode ser o comportamento de um domínio composto por agregação, arranjo ou potenciação, ou seja

$$M_{c_i} = M_{ag_i} \text{ ou } M_{c_i} = M_{ar_i} \text{ ou } M_{c_i} = M_{po_i}$$

Como a relação entre os universos dos domínios compostos $\mathcal{DAG}, \mathcal{DAR}$ e $\mathcal{DPO}$ e seus respectivos universos de comportamentos $\mathcal{MAG}, \mathcal{MAR}$ e $\mathcal{MPO}$ é biunívoca temos que:

$$\text{se } M_{c_i} = M_{ag_i} \text{ então } D_{c_i} = D_{ag_i} \text{ ou}$$

$$\text{se } M_{c_i} = M_{ar_i} \text{ então } D_{c_i} = D_{ar_i} \text{ ou}$$

$$\text{se } M_{c_i} = M_{po_i} \text{ então } D_{c_i} = D_{po_i}.$$

4.5 Domínios encapsulados

Definição 4.33 *Um domínio encapsulado é definido por um único conjunto de operações, associado a múltiplos domínios básicos ou compostos.*

Um domínio encapsulado será representado por D_{e_i}. Assim,

$$D_{e_i} = \{D_{b_i} \text{ ou } D_{c_i}\} \tag{4.24}$$

Axioma 4.7 *Os múltiplos domínios que o domínio encapsulado engloba têm a mesma composição.*

Portanto,

$$\text{se } D_{e_i} = \{D_1, \ldots, D_j\} \text{ então } \texttt{Comp}(D_1) = \ldots = \texttt{Comp}(D_j)$$

Definição 4.34 *O comportamento de um domínio encapsulado é definido por* $M_{e_i} = \{R | R : D_{e_i} \rightarrow D_i\}$ *onde* D_{e_i} *pode ser qualquer um dos domínios enumerados em* D_{e_i} *e* D_i *é um domínio qualquer.*

Axioma 4.8 *O comportamento do domínio encapsulado é sempre diferente do comportamento dos domínios por ele englobado.*

Portanto,

$$\text{se } D_{e_i} = \{D_1, \ldots, D_j\} \text{ então } \forall j, M_{e_i} \neq M_j$$

Diferentemente dos domínios básicos e compostos, o domínio encapsulado possui múltiplas implementações que serão representadas por:

$$Imp(M_{e_i}) = \{(D_j, I_{M_{e_{i,j}}})\} \tag{4.25}$$

onde $I_{M_{e_{i,j}}}$ é o j-ésimo conjunto implementação das relações em M_{e_i} para dados segundo a composição especificada em D_j.

Axioma 4.9 *Nas múltiplas implementações de um domínio encapsulado o conjunto de operações é o mesmo; entretanto, o conjunto implementação das relações é sempre diferente.*

Assim,

$$\forall k \neq j,\ I_{M_{e_{i,j}}} \neq I_{M_{e_{i,k}}}$$

Definição 4.35 *O universo $\mathcal{DE}$ dos domínios encapsulados é a família de domínios encapsulados definida por*

$$\mathcal{DE} = \{D_{e_1}, D_{e_2}, \ldots\}$$

Definição 4.36 *O universo $\mathcal{ME}$ dos comportamentos dos domínios encapsulados é a família de conjuntos de operações definida por*

$$\mathcal{ME} = \{M_{e_1}, M_{e_2}, \ldots\}$$

Axioma 4.10 *A relação entre o universo $\mathcal{DE}$ dos domínios encapsulados e o universo $\mathcal{ME}$ dos comportamentos dos domínios encapsulados é biunívoca.*

4.6 Domínio genérico

Definição 4.37 *O universo genérico de domínios é formado pela união dos universos dos domínios básicos, compostos e encapsulados.*

O universo genérico de domínios será representado por $\mathcal{D}$. Assim,

$$\mathcal{D} = \mathcal{DB} \cup \mathcal{DC} \cup \mathcal{DE} \tag{4.26}$$

Um domínio qualquer do universo genérico $\mathcal{D}$ será representado por D_i. Assim,

$$D_i \in \mathcal{D} \tag{4.27}$$

D_i pode ser um domínio básico, composto ou encapsulado, ou seja

$$D_i = D_{b_i} \text{ ou } D_i = D_{c_i} \text{ ou } D_i = D_{e_i}.$$

Definição 4.38 *O universo genérico de comportamentos é formado pela união dos universos de comportamento dos domínios básicos, compostos e encapsulados.*

O universo genérico de comportamentos será representado por $\mathcal{M}$. Assim,

$$\mathcal{M} = \mathcal{MB} \cup \mathcal{MC} \cup \mathcal{ME} \tag{4.28}$$

Um comportamento qualquer do universo genérico de comportamento de domínios será representado por M_i. Assim,

$$M_i \in \mathcal{M} \tag{4.29}$$

M_i pode ser o comportamento de um domínio básico, composto ou encapsulado, ou seja

$$M_i = M_{b_i} \text{ ou } M_i = M_{c_i} \text{ ou } M_i = M_{e_i}$$

Como a relação entre os universos dos domínios básicos, compostos e encapsulado e seus respectivos universos de comportamentos é biunívoca temos que:

$$\text{se } M_i = M_{b_i} \text{ então } D_i = D_{b_i},$$

$$\text{ou se } M_i = M_{c_i} \text{ então } D_i = D_{c_i},$$

$$\text{ou se } M_i = M_{e_i} \text{ então } D_i = D_{e_i} = \{D_j\}.$$

4.7 Revisitando domínios compostos

As Definições 4.12, 4.19 e 4.25 apresentam domínios que são compostos por domínios básicos. Esta restrição pode ser relaxada e domínios compostos podem ser definidos também a partir de domínios compostos ou encapsulados, *i.e.*, sobre domínios do universo genérico $\mathcal{D}$. Portanto, nesta seção estas definições serão revistas.

Axioma 4.11 *Não será permitida recursão na definição de um domínio composto.*

Uma definição recursiva de um domínio composto resultaria numa composição infinita do mesmo.

Definição 4.39 *O domínio composto por agregação é formado por qualquer um dos m! produtos cartesianos de m domínios univocamente rotulados.*

Assim,

$$D_{ag_i} = \times_m r_j : D_j \text{ onde } D_j \in \mathcal{D} \text{ e } D_j \neq D_{ag_i}$$

Definição 4.40 *Um domínio composto por arranjo é formado pela união de todos os possíveis produtos cartesianos de um mesmo domínio.*

Assim,

$$D_{ar_i} = \bigcup_{j=1}^{\infty} \times_j D_i \text{ onde } D_i \in \mathcal{D} \text{ e } D_i \neq D_{ar_i}$$

Definição 4.41 *Um domínio composto por potenciação é formado a partir do conjunto potência de um domínio qualquer.*

Assim,

$$D_{po_i} = 2^{D_i} \text{ onde } D_i \in \mathcal{D} \text{ e } D_i \neq D_{po_i}$$

4.8 Subdomínios

Conforme exposto na Seção 4.2, deseja-se definir domínios polimórficos, *i.e.*, domínios em que o princípio da substituição é válido entre seus dados; ou seja:

- *Propriedade de polimorfismo ($\preceq$[5]) de domínios.*

$$D_i' \preceq D_i \text{ se } b \in D_i',\ a \in D_i \text{ e } \forall b, b \preceq a \tag{4.30}$$

$D_i' \preceq D_i$ lê-se "o domínio D_i' é subdomínio do domínio D_i"ou ainda "o domínio D_i é super-domínio do domínio D_i'".

A seguir a relação de subdomínio obtida da propriedade de polimorfismo ($D_i' \preceq D_i$) será detalhada para cada categoria de domínio definida anteriormente, assim como a relação correspondente para o comportamento associado ao domínio.

[5] *O símbolo $\preceq$ usado para representar a propriedade de polimorfismo de dados (Equação 4.2) também será utilizado para representar domínios polimórficos.*

4.8.1 Subdomínios de domínios básicos

O princípio da substituição foi apresentado no Axioma 4.1 considerando domínios comparáveis, sendo que para domínios básicos esta comparabilidade foi restrita a igualdade de domínio. Portanto, dados atômicos eram considerados comparáveis se pertencessem a domínios iguais. Esta restrição pode ser relaxada para dados que pertencem a domínios com relação de inclusão de conjunto.

Definição 4.42 *Um conjunto A é dito ser subconjunto do conjunto B se todo dado $a \in A$ é também um dado do conjunto B.*

Assim,

$$A \subseteq B \text{ se } a \in A \text{ e } \forall a, a \in B$$

Se o conjunto A é subconjunto do conjunto B, então o conjunto B é denominado super-conjunto de A.

Axioma 4.12 (Princípio da Substituição - Revisitado) *Todo dado de um subconjunto pode ser utilizado no lugar de um dado de seu super-conjunto.*

A Definição 4.42 e o Axioma 4.12 permitem um relaxamento da propriedade de polimorfismo de dados atômicos expresso na Equação 4.2 para:

- *Propriedade de polimorfismo por inclusão de dados atômicos.*

$$d_i \preceq d_j \text{ se } d_i \in D_{b_i},\ d_j \in D_{b_j} \text{ e } D_{b_i} \subseteq D_{b_j} \tag{4.31}$$

Da propriedade de polimorfismo de domínios (Equação 4.30) e da propriedade de polimorfismo por inclusão de dados atômicos (Equação 4.31) temos a:

- *Propriedade de polimorfismo por inclusão de domínios básicos.*

$$D'_{b_i} \preceq D_{b_i} \text{ se } D'_{b_i} \subseteq D_{b_i} \tag{4.32}$$

$$\text{onde } D'_{b_i} = \{ \text{ subintervalos e/ou enumerações de dados de } D_{b_i} \}$$

4.8.2 Subdomínio do domínio composto por agregação

Da propriedade de polimorfismo de domínios (Equação 4.30), da propriedade de polimorfismo de n-uplas (Equação 4.8) e do princípio de substituição revisitado (Axioma 4.12) temos a:

- *Propriedade de polimorfismo por inclusão de domínios compostos por agregação.*

$$D'_{ag_i} \preceq D_{ag_i} \text{ se } D'_{ag_i} \subseteq D_{ag_i} \text{ tal que}$$

$$D'_{ag_i} = \times_n r'_k : D'_k,\ D_{ag_i} = \times_m r_j : D_j,$$

$$n \geq m \text{ e } \forall(r_j : D_j)\exists_1(r'_k : D'_k) \text{ tal que } r'_k = r_j \text{ e } D'_k \preceq D_j \tag{4.33}$$

Devido à relação biunívoca entre domínio composto por agregação e comportamento (Axioma 4.4) temos que os conjuntos de operações correspondentes serão:

$$M_{ag_i} = \{\{r_j : D_{ag_i} \rightarrow D_j\}_m, \preceq: D_{ag_i} \rightarrow D_{ag_i}, =: D_{ag_i} \rightarrow D_{ag_i}\}$$

$$M'_{ag_i} = \{\{r_k : D'_{ag_i} \rightarrow D'_k\}_n, \preceq: D'_{ag_i} \rightarrow D_{ag_i}, =: D'_{ag_i} \rightarrow D'_{ag_i}\}$$

Deve-se notar que existe uma relação co-variante entre domínios das relações deste conjuntos de operações, bem como entre os contradomínios, *i.e.*, tantos os domínios quanto os contradomínios das relações em M'_{ag_i} são iguais aos domínios e contradomínios das relações correspondentes em M_{ag_i}, ou subdomínios destes.

Deve-se notar também que um subdomínio composto por agregação:

- estende a definição do super-domínio, *i.e.*, acrescenta novos pares *rótulo:domínio* na definição do produto cartesiano pois $n \geq m$, o que implica em acrescentar também o comportamento correspondente, e/ou
- especializa (restringe) um domínio componente, substituindo-o por um subdomínio do mesmo pois $D'_k \preceq D_j$, o que implica na sobreposição (*overriding*) de comportamento.

Como um subdomínio composto por agregação pode estender a representação do seu super-domínio a Definição 4.39 de domínio composto por agregação deve ser ampla o suficiente para incluir qualquer possibilidade de extensão, de tal forma que a Equação 4.33 seja sempre verdadeira.

Definição 4.43 (Domínio composto por agregação - Revisitado) *O domínio composto por agregação é formado por qualquer um dos m! produtos cartesianos que inclua os m domínios univocamente rotulados.*

Assim, a representação de domínios compostos por agregação será modificada para:

$$D_{ag_i} = \times_m r_j : D_j \times \ldots \tag{4.34}$$

A equação acima representa qualquer produto cartesiano onde os domínios $D_1, \ldots, D_m$ participam identificados pelos rótulos $r_1, \ldots, r_m$ respectivamente, ou seja

$$D_{ag_i} = r_1 : D_1 \times r_2 : D_2 \times \ldots \times r_m : D_m \times \ldots$$

Segundo a Definição 4.43 o domínio abaixo é super-domínio de qualquer domínio composto por agregação pois não restringe quais domínios rotulados devem participar do produto cartesiano:

$$D_{ag_{sup}} = \times \ldots$$

4.8.3 Subdomínio do domínio composto por arranjo

Da propriedade de polimorfismo de domínios (Equação 4.30), da propriedade de polimorfismo de arranjos (Equação 4.13) e do princípio de substituição revisitado (Axioma 4.12) temos a:

- *Propriedade de polimorfismo por inclusão de domínios compostos por arranjo.*

$$D'_{ar_i} \preceq D_{ar_i} \text{ se } D'_{ar_i} \subseteq D_{ar_i} \text{ tal que}$$

$$D_{ar_i} = \bigcup_{j=1}^{\infty} \times_j D_i,\ D'_{ar_i} = \bigcup_{j=1}^{\infty} \times_j D'_i,$$

$$\forall l, (\times_l D'_i) \preceq (\times_k D_i), l \geq k \text{ e } D'_i \preceq D_i \tag{4.35}$$

Devido à relação biunívoca entre domínio composto por arranjo e comportamento (Axioma 4.5) temos que os conjuntos de operações correspondentes serão:

$$M_{ar_i} = \{\{\{k : \times_j D_i \rightarrow D_i\}_{k=1}^{j}\}_{j=1}^{\infty}, \preceq : D_{ar_i} \rightarrow D_{ar_j}, =: D_{ar_i} \rightarrow D_{ar_i}\}$$

$$M'_{ar_i} = \{\{\{k : \times_j D'_i \rightarrow D'_i\}_{k=1}^{j}\}_{j=1}^{\infty}, \preceq : D'_{ar_i} \rightarrow D_{ar_i}, =: D'_{ar_i} \rightarrow D'_{ar_i}\}$$

Deve-se notar também em subdomínios compostos por arranjo uma relação co-variante entre os conjuntos de operações associados aos sub e super-domínios.

Assim como observado em domínios compostos por agregação, não se restringindo o domínio componente na definição de um domínio composto por arranjo se está na verdade definindo o super-domínio de qualquer domínio composto por arranjo, *e.g.*:

$$D_{ar_{sup}} = \bigcup_{j=1}^{\infty} \times_j \ldots$$

4.8.4 Subdomínio do domínio composto por potenciação

Da propriedade de polimorfismo de domínios (Equação 4.30), da propriedade de polimorfismo de conjuntos (Equação 4.18) e do princípio de substituição revisitado (Axioma 4.12) temos a:

- *Propriedade de polimorfismo por inclusão de domínios compostos por potenciação.*

$$D'_{po_i} \preceq D_{po_i} \text{ se } D'_{po_i} \subseteq D_{po_i} \text{ tal que}$$

$$D_{po_i} = 2^{D_i},\ D'_{po_i} = 2^{D'_i},$$

$$\text{e } D'_i \preceq D_i \tag{4.36}$$

Devido à relação biunívoca entre domínio composto por potenciação e comportamento (Axioma 4.6) temos que os conjuntos de operações correspondentes serão:

$$M_{po_i} = \{contem : D_{po_i} \times D_i \rightarrow \{v, f\}, \preceq : D_{po_i} \rightarrow D_{po_j}, = : D_{po_i} \rightarrow D_{po_i}\}$$

$$M'_{po_i} = \{contem : D'_{po_i} \times D'_i \rightarrow \{v, f\}, \preceq : D'_{po_i} \rightarrow D_{po_i}, = : D'_{po_i} \rightarrow D'_{po_i}\}$$

Deve-se notar também em subdomínios compostos por potenciação uma relação co-variante entre os conjuntos de operações associados aos sub e super-domínios.

Da mesma forma, como foi ; portanto: em domínios compostos por agregação e arranjo, também em domínios compostos por potenciação não se restringindo o domínio componente na definição se está na verdade definindo o super-domínio de qualquer domínio composto por arranjo, *e.g.*:

$$D_{po_{sup}} = 2^{\cdots}$$

4.8.5 Subdomínio do domínio encapsulado

Um domínio encapsulado é definido por um comportamento que pode ser associado a múltiplos domínios. Sendo assim é o conjunto de operações que diferencia dois domínios encapsulados. Portanto, para se definir o conceito de subdomínio dos domínios encapsulados, *i.e.*, comparabilidade entre domínio encapsulados, é necessário basear-se no princípio da substituição para tipos funcionais.

Axioma 4.13 (Princípio da Substituição para tipos funcionais segundo [Car84]) *O espaço funcional F, composto pelo conjunto de relações do domínio D no contradomínio C, está contido no (ou é comparável ao) espaço funcional F', composto pelo conjunto de relações do domínio D' no contradomínio C', se $D' \subseteq D$ e $C' \supseteq C$.*

Um espaço funcional F que contém relações do domínio D no contradomínio C será representado por:

$$F = \{f | f : D \rightarrow C\} \tag{4.37}$$

Do Axioma 4.13 temos a:

- *Propriedade de polimorfismo de inclusão de relações/funções.*

$$f \preceq f' \text{ se } f \in F, f' \in F' \text{ e } F \subseteq F' \tag{4.38}$$

O relacionamento $F \subseteq F'$ é dito ser contra-variante pois as relações de comparabilidade entre domínios e contradomínios dos espaços funcionais são inversas. Esta inversão justifica-se ao se analisar o polimorfismo de relações, ou seja:

> *Se $f' \in F'$ sendo $F' = \{f' | f : D' \rightarrow C'\}$ e $f \in F$ sendo $F = \{f | f : D \rightarrow C\}$ e $F \subseteq F'$ então $f(a')$ pode substituir $f'(a')$. A aplicação de $f(a')$ é válida se $a' \in D'$ e $D' \subseteq D$. O resultado esperado da aplicação de $f'(a')$ seria do contradomínio C'; entretanto, $f(a')$ mapeia para o contradomínio C, mas como $C \subseteq C'$ a substituição levou a resultados de domínios comparáveis; portanto, estaticamente corretos.*

O conjunto de operações M_i, associado a um domínio D_i, pode ser caracterizado como um conjunto de espaços funcionais onde o domínio das relações é o próprio domínio D_i. Assim temos a:

- *Propriedade de polimorfismo por inclusão contra-variante de conjunto de operações.*

$$M_i \preceq M_i'$$

$$\text{para} \quad M_i = \{F_j | F_j = \{f_j | f_j : \{D_i \rightarrow D_j\}\}\}$$

$$\text{e} \quad M_i' = \{F_k' | F_k' = \{f_k' | f_k' : \{D_i' \rightarrow D_k'\}\}$$

$$\text{se} \quad k \geq j \text{ e} \quad \forall F_j \exists_1 F_k' | F_j \subseteq F_k' \tag{4.39}$$

Deve-se notar que a propriedade de polimorfismo por inclusão de comportamentos (*i.e.*, conjunto de operações) é inversa com relação a propriedade de polimorfismo por inclusão de seu respectivos domínios:

$$\text{se} \quad M_i \preceq M_i' \text{ então} \quad D_i' \preceq D_i \tag{4.40}$$

Enquanto um dado do subdomínio D_i' pode ser utilizado em qualquer lugar em que um dado do seu super-domínio D_i é esperado, uma relação $f_j \in F_j$ do comportamento M_i associado ao super-domínio D_i pode substituir uma relação do espaço funcional $F_k' \in M_i'$ associado subdomínio D_i', desde que $F_j \subseteq F_k'$.

Entretanto, como já observado nas Seções 4.8.2, 4.8.3 e 4.8.4, a relação existente entre conjuntos de operações de sub e super-domínios compostos não é contra-variante e sim co-variante. Portanto, temos a:

- *Propriedade de polimorfismo por inclusão co-variante de conjunto de operações.*

$$M_i \preceq M_i'$$

$$\text{para} \quad M_i = \{F_j | F_j = \{f_j | f_j : \{D_i \rightarrow D_j\}\}\}$$

$$\text{e} \quad M_i' = \{F_k' | F_k' = \{f_k' | f_k' : \{D_i' \rightarrow D_k'\}\}$$

$$\text{se} \quad k \geq j \text{ e} \quad \forall F_j \exists_1 F_k' | D_i' \preceq D_i \text{ e} \quad D_k' \preceq D_k \tag{4.41}$$

Uma relação contra-variante entre comportamentos polimórficos implica em uma verificação de tipos estática, sendo que a relação co-variante implica em uma verificação de tipos dinâmica. O raciocínio de Cardelli sobre o princípio da substituição entre tipos funcionais (Axioma 4.13) serve para demonstrar a relação inversa de polimorfismo existente entre domínio e subdomínio e seus respectivos conjuntos de operações (Equação 4.40).

Finalmente a propriedade de polimorfismo por inclusão entre domínios encapsulados pode ser deduzida, ou seja:

- *Propriedade de polimorfismo por inclusão de domínios encapsulados.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \preceq M'_{e_i} \text{ segundo a Equação 4.41} \tag{4.42}$$

4.9 Extensões de domínios e instâncias

Como visto na Seção 3.4, instâncias compõem as extensões dos domínios no nível concreto. Instâncias são pares ordenados (a, b) resultado do mapeamento de funções entre conjuntos de identificadores associados aos domínios. A seguir apresentaremos primeiramente as definições para conjunto de identificadores, em seguida extensões como o conjunto solução de funções que fazem os mapeamentos entre estes conjuntos de identificadores e finalmente instâncias como valores e objetos, bem como os possíveis testes de igualdade entre instâncias.

4.9.1 Conjunto de identificadores

Definição 4.44 *O conjunto de identificadores associado a um domínio básico é igual ao próprio domínio básico.*

O conjunto de identificadores associado ao domínio básico D_{b_i} será representado por ID_{b_i}. Assim,

$$\forall D_{b_i} \exists_1 ID_{b_i} \text{ tal que } ID_{b_i} = D_{b_i} \tag{4.43}$$

Definição 4.45 *O conjunto de identificadores associado a um domínio encapsulado é único, varia com o tempo e é subconjunto do conjunto de símbolos* $IDO = \{\#1, \#2, \#3, \ldots\}$.

Axioma 4.14 *Em nenhum instante existe interseção entre os conjuntos de identificadores associados aos domínios encapsulados.*

O conjunto de identificadores associado ao domínio encapsulado D_{e_i}, no instante t, será representado por $ID_{e_{it}}$. Assim,

$$\text{em qualquer instante } t, \forall D_{e_i} \exists_1 ID_{e_{it}} \text{ tal que} \tag{4.44}$$

$$ID_{e_{it}} \subset IDO \text{ e } ID_{e_{1t}} \cap ID_{e_{2t}} \cap ID_{e_{3t}} \ldots = \emptyset$$

Definição 4.46 *O conjunto de identificadores de um domínio composto é igual a um conjunto com o mesmo fator de composição aplicado sobre o conjunto de identificadores dos domínios que o compõem.*

O conjunto de identificadores do domínio composto D_{c_i} será representado por ID_{c_i}. Como um domínio composto pode ser composto por agregação, arranjo e potenciação temos que:

$$D_{c_i} = D_{ag_i} \text{ ou } D_{c_i} = D_{ar_i} \text{ ou } D_{c_i} = D_{po_i}$$

Assim,

$$\forall D_{c_i} \exists_1 ID_{c_i} \text{ tal que} \tag{4.45}$$

$$\text{se } D_{c_i} = D_{ag_i} = \times_m r_j : D_{comp_j} \text{ então } ID_{c_i} = \times_m r_j : ID_{comp_j},$$

$$\text{se } D_{c_i} = D_{ar_i} = \bigcup_{j=1}^{\infty} \times_j D_{comp} \text{ então } ID_{c_i} = \bigcup_{j=1}^{\infty} \times_j ID_{comp},$$

$$\text{se } D_{c_i} = D_{po_i} = 2^{D_{comp}} \text{ então } ID_{c_i} = 2^{ID_{comp}}$$

onde

$$\text{se } D_{comp} \in \mathcal{DB} \text{ então } ID_{comp} = ID_b \text{ segundo a Definição 4.44}$$

$$\text{se } D_{comp} \in \mathcal{DE} \text{ então } ID_{comp} = ID_e \text{ segundo a Definição 4.45}$$

$$\text{se } D_{comp} \in \mathcal{DC} \text{ então } ID_{comp} = ID_c \text{ segundo a Definição 4.46}$$

Um dado de um conjunto de identificadores será denominado *id*. Assim,

$$id \in ID_b \text{ ou } id \in ID_e \text{ ou } id \in ID_c \tag{4.46}$$

As relações de igualdade de dados apresentadas nas Seções 4.3, 4.4.1, 4.4.2 e 4.4.3 também são válidas entre os *ids* dos conjuntos de identificadores. Assim,

$id_i = id_j$ equivale a igualdade de dados atômicos, se $\{id_i, id_j\} \in \{ID_b \text{ ou } IDO\}$

$id_i = id_j$ segundo as Definições 4.15, 4.21 ou 4.27, se $\{id_i, id_j\} \in ID_c$

Definição 4.47 *A função recursiva* `CompID` *reescreve um id em termos de ids pertencentes a um conjunto de identificadores associado aos domínios básicos e encapsulados.*

$$\text{se } id \in \{ID_b, ID_e\} \text{ então } \texttt{CompID}(id) = id$$

$$\text{se } id \in ID_c = ID_{ag} \text{ então } \texttt{CompID}([r_i : id_i, \ldots]) = [\texttt{CompID}(id), \ldots)$$

$$\text{se } id \in ID_c = ID_{ar} \text{ então } \texttt{CompID}([id_1, \ldots]) = [\texttt{CompID}(id_1), \ldots]$$

$$\text{se } id \in ID_c = ID_{po} \text{ então } \texttt{CompID}(\{id_i, \ldots\}) = \{\texttt{CompID}(id_i), \ldots\}$$

4.9.2 Extensão de domínios básicos

Definição 4.48 *A extensão de um domínio básico é o conjunto solução resultado da aplicação da função identidade sobre o conjunto de identificadores associado ao domínio básico.*

A extensão do domínio básico D_{b_i} será representada por ED_{b_i}. A função identidade aplicada sobre o conjunto de identificadores associado ao domínio básico será representada por fid_{b_i}. O conjunto solução da função fid_{b_i} será representado por $fid^*_{b_i}$. Assim,

$$\forall D_{b_i} \exists_1 ED_{b_i} \text{ tal que} \tag{4.47}$$

$$ED_{b_i} = fid^*_{b_i},$$

$$fid^*_{b_i} = \{(a, a) | a \in ID_{b_i}, a = fid_{b_i}(a)\},$$

$$fid_{b_i} : ID_{b_i} \rightarrow ID_{b_i}$$

4.9.3 Extensão de domínios encapsulados compostos por um único domínio básico

Definição 4.49 *A extensão de um domínio encapsulado composto por um único domínio básico é igual ao conjunto solução de uma função, que varia com o tempo, do conjunto de identificadores associado ao domínio encapsulado no conjunto de identificadores associados ao domínio básico que compõe o domínio encapsulado.*

A extensão do domínio encapsulado D_{e_i}, no instante t, será representada por $ED_{e_{it}}$. A função, que varia com o tempo, e faz o mapeamento do conjunto de identificadores associado ao domínio encapsulado no conjunto de identificadores associado ao domínio básico será

representada por $foa_{e_i t}$. O conjunto solução da função $foa_{e_i t}$ será representado por $foa^*_{e_i t}$. Assim,

$$\text{no instante } t, \forall(D_{e_i} = D_{b_j})\exists_1 ED_{e_{it}} \text{ tal que} \tag{4.48}$$

$$ED_{e_{it}} = foa^*_{e_i t},$$

$$foa^*_{e_i t} = \{(a,b)|a \in ID_{e_i t}, b = foa_{e_i t}(a), b \in ID_{b_j}\}$$

$$\text{e } foa_{e_i t} : ID_{e_i t} \rightarrow ID_{b_j}$$

4.9.4 Extensão de domínios compostos

Definição 4.50 *A extensão de um domínio composto é o conjunto solução resultado da aplicação da função identidade sobre o conjunto de identificadores associado ao domínio composto.*

A extensão do domínio composto D_{c_i} será representada por ED_{c_i}. A função identidade aplicada sobre o conjunto de identificadores associado ao domínio composto será representada por fid_{c_i}. O conjunto solução da função fid_{c_i} será representado por $fid^*_{c_i}$. Assim,

$$\forall D_{c_i}\exists_1 ED_{c_i} \text{ tal que} \tag{4.49}$$

$$ED_{c_i} = fid^*_{c_i},$$

$$fid^*_{c_i} = \{(a,a)|a \in ID_{c_i}, a = fid_{c_i}(a)\},$$

$$fid_{c_i} : ID_{c_i} \rightarrow ID_{c_i}$$

4.9.5 Extensão de domínios encapsulados compostos por domínios compostos

Definição 4.51 *A extensão de um domínio encapsulado composto por domínios compostos é igual ao conjunto solução de uma função, que varia com o tempo, do conjunto de identificadores associado ao domínio encapsulado na união dos conjuntos de identificadores associados aos domínio que compõem o domínio encapsulado.*

A função, que varia com o tempo, e faz o mapeamento do conjunto de identificadores associado ao domínio encapsulado nos conjuntos de identificadores associados aos domínios

que compõem o domínio encapsulado será representada por $fo_{e_i t}$. O conjunto solução da função $fo_{e_i t}$ será representado por $fo^*_{e_i t}$. Assim,

$$\text{no instante } t, \forall(D_{e_i} = \{D_{c_j}\})\exists_1 ED_{e_i t} \text{ tal que} \tag{4.50}$$

$$ED_{e_i t} = fo^*_{e_i t},$$

$$fo^*_{e_i t} = \{(a,b) | a \in ID_{e_i t}, b = fo_{e_i t}(a), b \in \{\cup\, ID_{c_j}\}\}$$

$$\text{e } fo_{e_i t} : ID_{e_i t} \rightarrow \{\cup\, ID_{c_j}\}$$

4.9.6 Instâncias

Definição 4.52 *Instâncias são pares ordenados (a, b) resultado do mapeamento das funções $fid_{b_i}, fid_{c_i}, foa_{e_i}$ e fo_{e_i} entre conjuntos de identificadores associados aos domínios.*

Portanto, as extensões de domínios no nível concreto são compostas por instâncias.

Definição 4.53 *A função* `identificador` *aplicada sobre uma instância retorna o primeiro dado do par ordenado (a, b).*

Assim,

$$\texttt{identificador}((a,b)) = a$$

Definição 4.54 *A função* `estado` *aplicada sobre uma instância retorna o segundo dado do par ordenado (a, b).*

Assim,

$$\texttt{estado}((a,b)) = b$$

Desta forma, o primeiro dado de uma instância será denominado *identificador* e o segundo *estado*. Da Definição 4.52 temos que tanto o *identificador* como o *estado* de uma instância são *ids*.

Sub-classificações

Como visto na Seção 3.4.1, instâncias são classificadas em valor ou objeto, sendo que estes são ainda classificados em atômicos, simples ou complexos. A classificação de instância em valor ou objeto está relacionada a igualdade entre *identificador* e *estado* da instância. A sub-classificação de valor e objeto em atômico, simples e complexo está relacionada com a composição do *estado* da instância.

Definição 4.55 *O par ordenado* $(a, a) \in fid^*_{b_i}$ *é denominado valor atômico.*

Definição 4.56 *O par ordenado* $(a, b) \in foa^*_{b_i}$ *é denominado objeto atômico.*

Definição 4.57 *O par ordenado* $(a, a) \in fid^*_{c_i}$ *é denominado valor complexo* **se qualquer** *id em* `CompID(estado((a, a)))` *pertencer ao conjunto de símbolos IDO.*

Definição 4.58 *O par ordenado* $(a, a) \in fid^*_{c_i}$ *é denominado valor simples* **se nenhum** *id em* `CompID(estado((a, a)))` *pertencer ao conjunto de símbolos IDO.*

Definição 4.59 *O par ordenado* $(a, b) \in fo^*_{e_i}$ *é denominado objeto complexo* **se qualquer** *id em* `CompID(estado((a, a)))` *pertencer ao conjunto de símbolos IDO.*

Definição 4.60 *O par ordenado* $(a, b) \in fo^*_{c_i}$ *é denominado objeto simples* **se nenhum** *id em* `CompID(estado((a, a)))` *pertencer ao conjunto de símbolos IDO.*

Igualdade

Como visto na Seção 3.4.3 existem variações nos testes de igualdade de instâncias que podem ser idênticas ou ter igualdade rasa ou total. A diferenciação entre identidade e igualdade está relacionada ao fato de uma instância ser um par *(identificador, estado)* e do *identificador* poder estar desassociado do *estado*. A diferenciação entre igualdade rasa e total está relacionada à aplicação do teste de igualdade sobre uma instância com o seu estado desreferenciado ou não pela função `Desref` definida a seguir.

Definição 4.61 *A função recursiva* `Desref` *reescreve um id em termos de ids pertencentes a conjuntos de identificadores associados a domínios básicos.*

Assim,

$$\text{se } id \in ID_b \text{ então } \texttt{Desref}(id) = id$$

$$\text{se } id \in ID_c = ID_{ag} \text{ então } \texttt{Desref}([r_i : id_i, \ldots]) = [r_i : \texttt{Desref}(id_i), \ldots]$$

$$\text{se } id \in ID_c = ID_{ar} \text{ então } \texttt{Desref}([id_1, \ldots]) = [\texttt{Desref}(id_1), \ldots]$$

$$\text{se } id \in ID_c = ID_{po} \text{ então } \texttt{Desref}(\{id_i, \ldots\}) = \{\texttt{Desref}(id_i), \ldots\}$$

$$\text{se } id \in ID_e \text{ então } \texttt{Desref}(id) = \texttt{Desref}(\texttt{estado}((a, b))) \text{ tal que}$$

$$\texttt{id} = \texttt{identificador}((a, b))$$

Desta forma, temos que as possíveis propriedades relacionadas a igualdade entre instâncias são:

1. *Propriedade de identidade (=) entre instâncias.*

$$(a_i, b_i) = (a_j, b_j) \text{ se} \tag{4.51}$$

$$\texttt{identificador}((a_i, b_i)) = \texttt{identificador}((a_j, b_j))$$

2. *Propriedade de igualdade rasa ($\simeq$) entre instâncias.*

$$(a_i, b_i) \simeq (a_j, b_j) \text{ se} \tag{4.52}$$

$$\texttt{identificador}((a_i, b_i)) \neq \texttt{identificador}((a_j, b_j)) \text{ e}$$

$$\texttt{estado}((a_i, b_i)) = \texttt{estado}((a_j, b_j))$$

3. *Propriedade de igualdade total ($\sim$) entre instâncias.*

$$(a_i, b_i) \sim (a_j, b_j) \text{ se} \tag{4.53}$$

$$\texttt{identificador}((a_i, b_i)) \neq \texttt{identificador}((a_j, b_j)) \text{ e}$$

$$\texttt{Desref}(\texttt{estado}((a_i, b_i))) = \texttt{Desref}(\texttt{estado}((a_j, b_j)))$$

4.10 Formalização de um estudo de caso

Com o intuito de mostrar uma aplicação da sintaxe adotada neste formalismo o primeiro estudo de caso apresentado no Apêndice A foi reescrito no Apêndice D. Neste apêndice para cada domínio do estudo de caso foi gerada uma tabela apresentando-se relações de subtipo (subclasse) do domínio, a definição matemática do domínio, o conjunto de assinaturas que define o comportamento do domínio, a equação de implementação do comportamento do domínio, o conjunto de instâncias do domínio e o conjunto de identificadores associado ao domínio.

4.11 Resumo

Neste capítulo o modelo referência proposto foi formalizado matematicamente usando teoria de conjuntos. A formalização é composta pelos seguintes conceitos base.

Tipos e classes

No modelo referência formalizado tipos equivalem a domínios, sendo que a interface de tipos equivale à descrição de um comportamento associado ao domínio. Comportamentos foram formalizados como mapeamentos genéricos, representados por assinaturas de relações ou funções. O mapeamento específico, *i.e.*, a implementação da relação ou função, foi incluído como uma "caixa preta", *i.e.*, não é detalhado. Desta forma, o tratamento de comportamento foi simplificado visando somente regras de sintaxe e não de semântica. O mapeamento específico é reassociado ao mapeamento genérico, na implementação do comportamento do domínio. A implementação do comportamento equivale ao conceito de classe.

Entretanto, como existem comportamentos que refletem a composição do domínio e comportamentos independentes da composição do domínio temos representado no modelo referência, via seu formalismo, classes sem e com encapsulamento, respectivamente. Por mais esta razão, o modelo referência é considerado mais abrangente que o padrão ODMG-93, não deixando de também ser compatível com o mesmo.

Exemplos de classes sem encapsulamento são as implementações de tipos básicos e estruturados em linguagens de programação procedimentais. Esses tipos, e classes correspondentes, poderiam ser representados no modelo referência pelo formalismo proposto da seguinte forma:

- a implementação do tipo básico Tb_i (*e.g.*, bit, integer, real) equivaleria a implementação do comportamento de um domínio básico D_{b_i} (Equação 4.3):

$$Tb_i = Imp(M_{b_i}) = (D_{b_i}, I_{M_{b_i}})$$

- a implementação do tipo estruturado Tc_i (*e.g.*, record, array, set) equivaleria a implementação do comportamento de um domínio composto D_{c_i} (Equações 4.9, 4.14 e 4.19):

$$Tc_i = Imp(M_{c_i}) = (D_{c_i}, I_{M_{c_i}})$$

Deve-se notar que em linguagens de programação procedimentais não se distingue classe, *i.e.* implementação do tipo, do próprio tipo. Esse fato é representado acima definindo-se tipo igual a implementação do comportamento (*e.g.*, $Tb_i = Imp(M_{b_i})$ e $Tc_i = Imp(M_{c_i})$).

Uma classe com encapsulamento seria representada no modelo referência via o formalismo proposto da seguinte forma:

- A C_i (*e.g.*, pessoa, carro, projeto) equivale a uma ou mais implementações do comportamento de um domínio encapsulado D_{e_i} (Equação 4.25):

$$C_i = Imp(M_{e_i}) = \{(D_j, I_{M_{e_{i,j}}})\}$$

O conceito de encapsulamento é representado no domínio encapsulado D_{e_i} pela associação múltipla de domínios a um único comportamento sem vínculo com a composição destes.

Subtipo e subclasse

Para representar o conceito de sub-tipicidade (subtipo ou subclasse) foram definidas regras para a definição de domínios e comportamentos com a propriedade de polimorfismo por inclusão estabelecendo-se relacionamentos de inclusão entre domínios e comportamentos de tal forma que:

- o tipo básico $Tb'_i = Imp(M'_{b_i}) = (D'_{b_i}, I_{M'_{b_i}})$ é um subtipo do tipo básico $Tb_i = Imp(M_{b_i}) = (D_{b_i}, I_{M_{b_i}})$ se

$$D'_{b_i} \preceq D_{b_i} \text{ (Equação 4.32)}$$

- o tipo estruturado $Tc'_i = Imp(M'_{c_i}) = (D'_{c_i}, I_{M'_{c_i}})$ é um subtipo do tipo estruturado $Tc_i = Imp(M_{c_i}) = (D_{c_i}, I_{M_{c_i}})$ se

$$D'_{c_i} \preceq D_{c_i} \text{ (Equações 4.33, 4.35 e 4.36)}$$

Conseqüentemente

$$M_{c_i} \preceq M'_{c_i} \text{ (Equação 4.41)}$$

$D'_{c_i} \preceq D_{c_i}$ representa um polimorfismo por inclusão onde a composição do dado é enfatizada. $M_{c_i} \preceq M'_{c_i}$ representa um polimorfismo por inclusão onde a conformidade co-variante de assinaturas de comportamentos é enfatizada.

- a classe $C'_i = Imp(M'_{e_i}) = \{(D'_j, I'_{M_{e_{i,j}}})\}$ é subclasse da classe $C_i = Imp(M_{e_i}) = \{(D_j, I_{M_{e_{i,j}}})\}$ se

$$\text{se } M_{e_i} \preceq M'_{e_i} \text{ (Equação 4.41)}$$

A relação inversa de polimorfismo por inclusão entre super-/subdomínio e seus comportamentos correspondentes é coerente com o fato de instâncias de sub-tipos/classes poderem substituir instâncias de seus super-tipos/classes e ao contrário, operações, dos super-tipos/classes poderem ser aplicadas sobre instâncias de seus sub-tipos/classes.

O Apêndice C apresenta, na sintaxe adotada neste capítulo, as variações no mecanismo de derivação de domínio, segundo [PS92], que geram diferentes níveis de relacionamentos de inclusão entre domínios e comportamentos. Estas variações foram apresentadas informalmente na Seção 3.3.4. O Apêndice C demonstra a potencialidade de expressão do formalismo, sendo adotado para caracterizar outros modelos.

Instâncias como valor e objeto

Como domínios equivalem a conjuntos de intenções de dados inclui-se estes num nível intencional de informações. Portanto, domínios, mapeamentos entre domínios (comportamentos) e implementações compõem uma descrição abstrata de uma porção do mundo real sendo modelada. Uma realização do mundo real, segundo esta descrição abstrata, é incluída no nível concreto de informações. O nível concreto de informações é formado por instâncias agrupadas em extensões associadas a domínios. Portanto, neste trabalho separa-se informações abstratas de concretas em dois níveis de informação o intencional e o concreto. Faz-se uma associação entre estes níveis associado-se domínios a instâncias via extensões.

A cada domínio foi associado um conjunto de identificadores (Definições 4.44, 4.46 e 4.45). Para cada categoria de domínio instâncias são definidas a partir de mapeamentos entre os respectivos conjuntos de identificadores. Os mapeamentos possíveis são de identidade (Definições 4.48 e 4.50) ou dinâmicos (que variam com o tempo) (Definições 4.49 e 4.51). Estes dois tipos de mapeamentos classificam instâncias em valores ou objetos. Dependendo ainda da composição do estado da instância, esta ainda é subclassificada como atômica, simples e complexa. Esta diferenciação separa instâncias que compartilham ou não modificações sobre objetos compartilhados, representando assim o conceito de valor e objeto complexo.

Capítulo 5

Conclusão

5.1 Contribuições, Vantagens e Desvantagens

No Capítulo 2 foi feita uma análise (ver Tabela 2.3) da origem dos modelos de objetos de SBDOO. Confirmou-se então a falta de um modelo de objetos padrão para SBDOO. Tal falha prejudica a portabilidade e interoperabilidade dos SBDOOs, desmotivando a aceitação e adoção destes pelos seus potenciais usuários. A partir de uma comparação mais detalhada de uma amostra de modelos de objetos de SBDOOs existentes, detectaram-se os principais pontos de divergência, tendo-se verificado que o problema tem origem na definição informal de SBDOOs.

Existem dois esforços na busca da padronização de SBDOOs; os trabalhos desenvolvidos pelos comitês ANSI X3H2 e ISO/IEC JTC1/SC21 e pelo grupo ODMG. O primeiro grupo está desenvolvendo o SQL3 [Kul94], uma proposta de padronização (ainda em elaboração) para SBDRs estendidos com funcionalidades de OO. O segundo grupo está desenvolvendo o padrão ODMG-93, uma proposta de padronização para SBDs que adotam um modelo de objetos como modelo de dados, tendo duas versões já publicadas [Cat94a, Cat96]. Entretanto, se estes esforços de padronização não forem acompanhados de desenvolvimento formal o problema persistirá.

No presente trabalho foi proposto um modelo referência de objetos e formalismo correspondente compatível com o padrão ODMG-93, visando contribuir para a garantia da correção entre modelos de objetos ditos padronizados. A proposta de padrão SQL3 foi preterida não só por ser voltada a extensões orientadas a objetos do modelo de dados relacional, e não a modelos de objetos por si, mas também por não ter nenhuma versão publicada.

O padrão ODMG-93 não foi formalizado diretamente e sim o modelo referência proposto. Esta abordagem é vantajosa; como o modelo referência é mais genérico e compatível com o padrão, o primeiro possibilita o mapeamento de modelos de objetos variados para o padrão, ampliando a aplicabilidade do formalismo.

Um dos pontos onde o modelo referência é mais genérico comparando-se ao modelo de objetos do ODMG-93 é na descrição de assinaturas. As assinaturas de atributos, relacionamentos e operações do ODMG-93 são todas representadas no modelo referência da mesma forma, *i.e.*, como assinaturas de relações. Isso permite que o modelo referência possa ser utilizado para mapear, para o padrão ODMG-93, modelos de objetos que não diferenciem atributos de relacionamentos na especificação de classes. Outro exemplo de generalidade no modelo referência é a possibilidade de representar classes com e sem encapsulamento.

Desta forma, o modelo referência proposto e formalismo correspondente podem ser utilizados para:

- mapear modelos de objetos para o padrão ODMG-93;
- caracterizar modelos de objetos variados; e
- verificar a correção de modelos de objetos adaptados para o padrão ODMG-93.

Portanto, nota-se que o modelo referência proposto, via seu formalismo, contribui para uma melhoria na portabilidade e interoperabilidade entre SBDOOs.

O Apêndice C apresenta uma amostra da potencialidade de caracterização de modelos com o formalismo desenvolvido neste trabalho. No apêndice, o formalismo foi utilizado para descrever variações do mecanismo de herança. A partir desta descrição, pode-se avaliar a validade do princípio da substituição em cada variação do mecanismo de herança.

O modelo referência foi descrito utilizando a teoria de conjuntos; visando um formalismo simplificado, flexível e com uniformidade de tratamento. O formalismo do modelo referência foi construído a partir dos conceitos básicos resumidos na Tabela 4.1. Um exemplo de flexibilidade no formalismo é a possibilidade de separação da sintaxe e semântica da relação e da recombinação variada destas para a definição de uma relação (ver Apêndice C). O formalismo inclui vários níveis de detalhamento de uma relação: somente pela assinatura (relativo a sintaxe), e pela assinatura e mapeamento específico (relativo a semântica); sendo que o mapeamento específico ainda pode ser representado por uma sentença aberta ou lei, e/ou um conjunto solução.

A característica genérica do modelo referência implica numa dificuldade de compreensão das aplicações por ele modeladas, *i.e.*, descritas pelos grafos e hierarquias de domínios na notação gráfica proposta (ver Apêndice A). Entretanto, deve-se ter em mente que o modelo referência representa verdadeiramente o modelo de objetos, portanto conclui-se que tal dificuldade é inerente ao paradigma.

5.2 Trabalhos futuros

Esse trabalho motiva inúmeros desenvolvimentos futuros a curto, médio e longo prazo, na sua maioria extensões. A curto prazo o formalismo pode ser estendido para incluir a identificação de instâncias por chaves e excessões em operações (facilidade do ODMG-93 não abordada, ver Tabela 3.2). A inclusão de chaves no formalismo é simples, envolvendo apenas a especificação de um novo operador de igualdade entre instâncias como os especificados na Seção 4.9.6.

A médio prazo, seria oportuno desenvolver uma interface para o modelo referência utilizando-se as linguagens ODL e OML do padrão ODMG-93. A sintaxe utilizada na formalização matemática do Capítulo 4 pode também ser utilizada como uma linguagem de definição de dados do formalismo (ver exemplo de aplicação no Apêndice D).

Também a médio prazo o modelo referência poderia ser estendido para incorporar restrições de integridade. Para tal, seria necessário especificar o esquema de criação e destruição de dados; via operações de criação e destruição não incluídas nesse trabalho. Isso implica em especificar no modelo referência o esquema de dependência de existencia entre valores e objetos complexos.

A semântica de relações foi tratada de forma limitada neste trabalho; no entanto, visualiza-se, a longo prazo, a extensão do formalismo incluindo a verificação da igualdade de semântica em relações. Neste trabalho, quando esta verificação foi necessária, foi incluída sem detalhamento (*e.g.*, no Apêndice C nas expressões $Imp(R_i) = Imp(R_j)$ ou $R_i^* = R_j^*$ não foram descritas). Este assunto é bastante complexo e sua problemática é abordada em trabalhos como o de [CMT91].

BIBLIOGRAFIA

[ABD+91] T. R. Ayers, D. K. Barry, J. D. Dolejsi, J. R. Galareneau, and R. V. Zoeller. Development of ITASCA. *Journal of Object-Oriented Programming*, 4(4):46–49, Julho 1991.

[ABD+92] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. *The Object-Oriented Database System Manifesto*, chapter 1, páginas 2–20. Em Bancilhon et al. [BDK92], 1992.

[AG89] R. Agrawal and N. H. Gehani. ODE (Object Database and Environment): The Language and the Data Model. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 36–45, 1989.

[AGOO85] A. Albano, G. Gheli, G. Occhiuto, and R. Orsini. Galileu: A Strongly Typed Interactive Conceptual Language. *ACM Transactions on Database Systems*, 10(2):230–261, 1985.

[AH87] T. Andrews and C. Harris. Combining Language and Database Advances in an Object-Oriented Development Environment. Em *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, páginas 430–440, 1987.

[Alt89] Altäir, Le Chesnay Cedex – France. *The O2 Query Language User's Manual*, Outubro 1989. Version 0.8, Release 16.

[AR91] J. Andersen and T. Reenskaug. Operations on Sets in an OODB. *ACM SIGPLAN Messenger*, 2(4):26–39, 1991.

[AWSL92] S. Ahmed, A. Wong, D. Sriram, and R. Logcher. Object-oriented database management systems for engineering: A comparion. *Journal of Object-Oriented Programming*, 5(3):27–44, Junho 1992.

[Bar90] G. Barbedette. Schema Manipulation in the LispO2 Persistent Object-Oriented Language. Technical Report 56-90, Altäir, Le Chesnay Cedex – France, Agosto 1990.

[Bau90] P. Baumann. Approaching a Formal Framework for the Object-Oriented Paradigm. Em *Proceedings of the IFIP Conference on CAD/CAM Technology Transfer*, páginas 31–38, 1990.

[BB90] F. Bancilhon and P. Buneman, editores. *Advances in Database Programming Languages*. ACM Press. Addison-Wesley Publishing Company, 1990.

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. A Query Language for the O2 Object-Oriented Database System. Technical report, Altair, Le Chesnay Cedex – France, Agosto 1989.

[BCG+87] J. Banerjee, H. Chou, J. F. Garza, W. Kim, D. Woelk, and N. Ballou. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Database Systems*, 5(1):3–26, Janeiro 1987.

[BDK92] F. Bancilhon, C. Delobel, and P. Kanellakis, editores. *Building an Object-Oriented Database System - The Story of O2*. Morgan Kaufmann Publishers, San Mateo, California, 1992.

[Bee89] C. Beeri. Formal Models for Object Oriented Databases. Em *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*, páginas 370–395, 1989.

[BK90] F. Bancilhon and W. Kim. Object-Oriented Database Systems: In Transition. *ACM SIGMOD Record*, 19(4):49–53, Dezembro 1990.

[BKK88] J. Banerjee, W. Kim, and K. Kim. Queries in Object-Oriented Databases. Em *Proceedings of the 4th. International Conference on Data Engineering*, páginas 31–38, 1988.

[BS92] K. Baclawski and D. A. Simovici. An Algebraic Approach to Databases with Complex Objects. *Information Systems*, 17(1):33–47, 1992.

[C+90] M. J. Carey et al. *The EXODUS Extensible DBMS Project: An Overview*, páginas 474–499. Em Zdonik and Maier [ZM90b], 1990.

[Car84] L. Cardelli. A Semantics of Multiple Inheritance. *Lecture Notes On Computer Science*, 173:51–67, 1984.

[Cat94a] R. G. G. Cattell, editor. *The Object Database Standard: ODGM-93, Release 1.1.* Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[Cat94b] R. G. G. Cattell. ODMG-93: A Standard for Object-Oriented DBMSs. *ACM SIGMOD Record*, 23(2):480, Junho 1994.

[Cat96] R. G. G. Cattell, editor. *The Object Database Standard: ODGM-93, Release 1.2.* Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[CDV88] M. J. Carey, D. J. DeWitt, and S. L. Vandenberg. A Data Model and Query Language for the EXODUS. *ACM SIGMOD Record*, 17(3):413–423, Setembro 1988.

[CJ92] W. Cellary and G. Jomier. *Consistency of Versions in Object-Oriented Databases*, páginas 447–462. Em Bancilhon et al. [BDK92], 1992.

[CMT91] I. Choi, M. V. Mannino, and V. P. Tseng. Graph Interpretation of Methods: A Unifying Framework for Polymorphism in Object-Oriented Programming. *ACM SIGPLAN Messenger*, 2(1):38–54, Janeiro 1991.

[CS90] M. Caruso and E. Sciore. *The VISION Object-Oriented Database Management System*, páginas 147–322. Em Bancilhon and Buneman [BB90], 1990.

[CW85] L. Cardelli and P. Wegner. On Understanding Type, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4):471–522, Dezembro 1985.

[D+87] K. Dittrich et al. DAMOKLES - The Database System for the UNIBASE Software Engineering Environment. *IEEE Database Engineering Bulletin*, 10(1):37–47, 1987.

[D+90] O. Deux et al. The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, Março 1990.

[DBB+88] U. Dayal, B. Blaustein, A. Bucmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC Project: Combining Active Databases and Timing Constraints. *ACM SIGMOD Record*, 17(1):51–70, 1988.

[DKT88] H. Duchene, M. Kaul, and V. Turau. VODAK Kernel Data Model. *Lecture Notes On Computer Science*, 334:242–261, 1988.

[DMB+90] U. Dayal, F. Manola, A. Buchmann, U. Chakravarthy, D. Goldhirsch, S. Heiler, J. Orenstein, and A. Rosenthal. *Simpliflying Complex Objects: The PROBE Approach to Modelling and Quering Them*, páginas 390–399. Em Zdonik and Maier [ZM90b], 1990.

[FAB+89] D. H. Fishman, J. Annevelink, D. Beech, E. Chow, T. Connors, J. W. Davis, W. Hasan, C. G. Hoch, W. Kent, S. Leichner, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. Risch, M. C. Shan, and W. K. Wilkinson. *Overview of the Iris DBMS*, páginas 219–250. Em Kim and Lochovsky [KL89], 1989.

[FJL+88] S. Ford, J. Joseph, D. E. Langworthly, G. Pathak D. F. Lively, E. R. Perez, R. W. Peterson, D. M. Sparacin, S. M. Thatte, D. L. Well, and S. Agarwala. ZEITGEIST: Database Support for Object-Oriented Programming. *Lecture Notes On Computer Science*, 334:23–42, 1988.

[Gor87] K. E. Gorlen. An Object-Oriented Class Library for C++ Programs. *Software Practice and Experience*, 17(2):899–922, Dezembro 1987.

[HK86] S. E. Hudson and R. King. CACTIS: A Database System for Specifying Functionally-Defined Data. Em *Proc. 1st. International Workshop on OODBS*, páginas 26–37, 1986.

[HS88] M. Hardwick and D. L. Spooner. ROSE: An Object-Oriented Database System for Interactive Computer Graphics Application. *Lecture Notes On Computer Science*, 334:340–345, 1988.

[Hul89] R. Hull. Four Views of Complex Objects: A Sophisticate's Introduction. *Lecture Notes On Computer Science*, 361:87–116, 1989.

[KA90] S. Khoshafian and R. Abnous. *Object Orientation - Concepts, Languages, Databases, User Interfaces*. John Wiley & Sons, Inc., 1990.

[KBC+87] W. Kim, J. Banerjee, H. Chou, J. F. Garza, and D. Woelk. Composite Objects Support in an Object-Oriented Database System. *ACM SIGPLAN Notices*, 22(12):118–125, Dezembro 1987.

[KBG89] W. Kim, E. Bertino, and J. F. Garza. Composite Objects Revisited. *ACM SIGMOD Record*, 18(2):337–347, Junho 1989.

[KC86] S. Khoshafian and G. Copeland. Object Identity. Em *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, páginas 406–416, 1986.

[KCGW88] W. Kim, N. B. Chou, J. F. Garza, and D. Woelk. Integrating an Object-Oriented Programming System with a Database System. Em *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, páginas 142–152, 1988.

[Kem87] A. Kemper. An Object-Oriented Database System for Engineering Applications. *ACM SIGMOD Record*, 16(3):299–310, Dezembro 1987.

[KGBW90] W. Kim, F.J. Garza, N. Ballou, and D. Woelk. Architecture of the ORION Next-Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, Março 1990.

[Kim89] W. Kim. A Model of Queries for Object-Oriented Databases. Em *Proceedings of the 15th. International Conference on Very Large Data Bases*, páginas 423–432, 1989.

[Kim90] W. Kim. Object-Oriented Databases: Definitions and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–342, Setembro 1990.

[Kim91] W. Kim. Object-oriented database systems: strengths and weaknesses. *Journal of Object-Oriented Programming*, 4(4):21–29, Julho 1991.

[Kim94] W. Kim. UniSQL/X Unified Relational and Object-Oriented Database System. *ACM SIGMOD Record*, 23(2):481, Junho 1994.

[KL89] W. Kim and F.H. Lochovsky, editores. *Object-Oriented Concepts, Databases and Applications*. ACM Press Frontier Series. Addison-Wesley Publishing Company, 1989.

[Kul94] K. G. Kulkarni. Object-Oriented Extensions in SQL3: A Status Report. *ACM SIGMOD Record*, 23(2):478, Junho 1994.

[LAC+93] M. E. S. Loomis, T. Atwood, R. Cattell, J. Duhl, G. Ferran, and D. Wade. The ODMG Object Model. *Journal of Object-Oriented Programming*, 6(3):64–69, Junho 1993.

[LKM90] P. C. Lockemann, A. Kemper, and G. Moerkotte. Future Database Technology: Driving Forces and Direction. *Lecture Notes On Computer Science*, 466:15–33, 1990.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore Database System. *Communications of the ACM*, 34(10):50–63, Outubro 1991.

[Loo94] M. E. S. Loomis. Comparing ODBMS features: A review of DBMS Needs Assesment for Objects. *Journal of Object-Oriented Programming*, 7(6):75–78, Outubro 1994.

[LRV88] C. Lécluse, P. Richard, and F. Velez. O2 an Object-Oriented Data Model. Em *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 424–433, Chicago, IL., Agosto 1988.

[Mai83] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Inc., 1983.

[MM95] J. Melton and N. M. Mattos. An Overview of the Emerging Third-Generation SQL Standard. *ACM SIGMOD Record*, 24(2):468, Junho 1995.

[MSOP86] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an Object-Oriented DBMS. Em *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, páginas 472–486, 1986.

[Pet87] R. W. Peterson. Object-Oriented Database Design. *AI Expert*, páginas 26–31, Março 1987.

[PM88] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, Setembro 1988.

[PS92] J. Palsberg and M. I. Schwartzbach. Three Discussions on Object-Oriented Typing. *ACM SIGPLAN Messenger*, 3(2):31–38, Abril 1992.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall International, Inc., 1991.

[RMS88] S. Riegel, R. Mellender, and A. Straw. Integration of Database Management with an Object-Oriented Programming Language. *Lecture Notes On Computer Science*, 334:317–322, 1988.

[Rum93] J. Rumbaugh. Desinherited! Examples of misuse of inheritance. *Journal of Object-Oriented Programming*, páginas 22–24, Fevereiro 1993.

[Sci89] E. Sciore. Object Specialization. *ACM Transactions on Office Information Systems*, 7(2):103–122, Abril 1989.

[SG92] S. S. Simmel and I. Godard. Objects of Substance. *BYTE*, páginas 167–172, Dezembro 1992.

[SLH90] M. Stonebraker, A. R. Lawrence, and M. Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, Março 1990.

[US90] R. Unland and G. Schlageter. Object-Oriented Database Systems: Concepts and Perspectives. *Lecture Notes On Computer Science*, 466:154–197, 1990.

[Vos91] G. Vossen. Bibliography on Object-Oriented Database Management. *ACM SIGMOD Record*, 20(1):24–46, Março 1991.

[Wan89] Y. Wand. *A Proposal for a Formal Model of Objects*, páginas 537–559. Em Kim and Lochovsky [KL89], 1989.

[WL89] S. P. Weiser and F. H. Lochovsky. *OZ+: An Object-Oriented Database System*, páginas 309–337. Em Kim and Lochovsky [KL89], 1989.

[ZM90a] S. B. Zdonik and D. Maier. *Fundamentals of Object-Oriented Databases*, páginas 1–32. Em [ZM90b], 1990.

[ZM90b] S. B. Zdonik and D. Maier, editores. *Readings in Object-Oriented Database Systems.* Morgan Kaufmann Publishers, 1990.

[ZW86] S. B. Zdonik and P. Wegner. Language and Methodology for Object-Oriented Database Environments. Em *Proceedings of the 19th Annual Hawaii International Conference on System Science*, páginas 378–387, Hononulu, 1986.

Apêndice A

Estudos de Caso

A.1 Introdução

Neste apêndice três estudos de caso são apresentados de forma a elucidar o modelo referência de objetos proposto no Capítulo 3 e formalizado no Capítulo 4. Pretende-se mostrar como os conceitos de classe, propriedades de classes (atributos e operações), relacionamento entre classes (associações quaisquer e do tipo agregação), hierarquia *is-a* de classes (com sobrecarga/sobreposição de operações) e encapsulamento são mapeados para o nível intencional do modelo referência proposto.

No primeiro estudo de caso, uma vez finalizada a modelagem no nível intencional pelo modelo referência será então apresentado o conjunto correspondente de extensões, contendo instâncias, no nível concreto de informações. As instâncias serão então analisadas para verificar se atendem a necessidades de compartilhamento de dados e de polimorfismo conforme especificado na apresentação inicial do estudo de caso.

Será utilizada a notação gráfica apresentada no Capítulo 3. Eventualmente alguns exemplos também serão apresentados na notação gráfica do OMT *(Object Modeling Technique)* [RBP+91], como um recurso gráfico extra para o entendimento do problema em questão.

A.2 Primeiro estudo de caso

O seguinte problema será utilizado como exemplo:

> *Deseja-se modelar um ambiente que manipula docentes e alunos em turmas para as quais deseja-se calcular a média aritmética salarial dos docentes e das*

notas dos alunos. Alunos sempre trabalham em grupo, compartilhando notas. Não existe o interesse em saber o relacionamento entre docentes e alunos e da mesma forma não interessa saber quais notas são compartilhadas por quais alunos.

Alunos podem ser de graduação ou de pós-graduação. A diferença entre alunos de graduação e pós-graduação está na nota e cálculo da média de suas notas. Alunos de graduação recebem notas que variam de A a E. Alunos de pós-graduação recebem notas que variam de A a C. Alunos de pós-graduação podem cursar disciplinas de graduação.

Docentes e alunos são identificados apenas pelos números de um dígito 1, 2, 3 e 4. Docentes além do número também têm salário, que variam de 1 a 4, e filhos, que também são identificados pelos números 1, 2, 3 e 4. Não interessa saber se dois docentes compartilham filhos. Alunos além do número têm até duas notas (que podem ter valores repetidos) e deseja-se saber a média das notas de cada aluno.

Esse exemplo foi inspirado em [Rum93] onde se discute erros de modelagem, como a utilização de herança para representar composição. No final do apêndice vê-se que o modelo referência proposto evita este tipo de uso errôneo de herança, pois oferece mecanismos que separam claramente a definição da composição da definição do comportamento e, portanto, fazem com que herança seja apenas utilizada como recurso para obter polimorfismo e/ou definição incremental.

A apresentação do exemplo dar-se-á por uma descrição informal e gráfica primeiramente da hierarquia *is-a* dos domínios básicos necessários no estudo de caso, seguida da hierarquia *is-a* dos domínios encapsulados e seus respectivos comportamentos e, finalmente, da hierarquia *is-a* de domínios compostos decorrente da hierarquia de domínios encapsulados. Uma vez apresentadas as hierarquias *is-a* de domínios e comportamentos no nível intencional, as instâncias correspondentes no nível concreto agrupadas em extensões serão apresentadas e analisadas.

A.2.1 Modelagem no nível intencional

Domínios básicos

Inicialmente serão definidos os domínios básicos da aplicação:

- $D_{b_2} = \{1, 2, 3, 4\}$, números de identificação de docentes, alunos e filhos de docentes.
- $D_{b_4} = \{A, B, C, D, E\}$, notas de graduação.
- $D_{b_5} = \{A, B, C\}$, notas de pós-graduação.
- $D_{b_6} = \{t, f\}$, valores *booleanos* resultados de testes de igualdades e de polimorfismo.
- $D_{b_{super}}$, D_{b_1} e D_{b_3}, super-domínios.

As Tabelas D.1 e D.2 no Apêndice D contêm a definição detalhada destes domínios básicos na sintaxe do formalismo do Capítulo 4.

A Figura A.1 mostra a organização destes domínios numa hierarquia *is-a*. Na parte (A) desta figura apresenta-se os relacionamentos de sub-tipicidade entre domínios básicos e a composição destes. A parte (B) desta figura mostra a hierarquia de domínios básicos na representação de diagramas Venn. O relacionamento de sub-tipicidade entre domínios equivale ao relacionamento de inclusão de conjunto entre os conjuntos de dados representados pelos domínios.

Esse conjunto simplista de domínios básicos forma a base de dados básicos que será utilizada para criar dados mais complexos neste primeiro estudo de caso. Propriedades de alunos e docentes como número pertencem ao domínio básico D_{b_2}, assim como a propriedade salário de docentes. A propriedade nota de alunos pertence ao domínio básico D_{b_4} ou D_{b_5}. O domínio D_{b_2} é uma especialização do domínio básico D_{b_1} que equivale ao conjunto de números inteiros. O domínio D_{b_4} é uma especialização do domínio básico D_{b_3} que equivale ao conjunto de caracteres. Ambos os domínios D_{b_1} e D_{b_3} são especializações do domínio básico $D_{b_{super}}$ que equivale ao conjunto de dados atômicos.

Domínios encapsulados

A aplicação define domínios encapsulados para representar:

- grupo de notas de alunos:

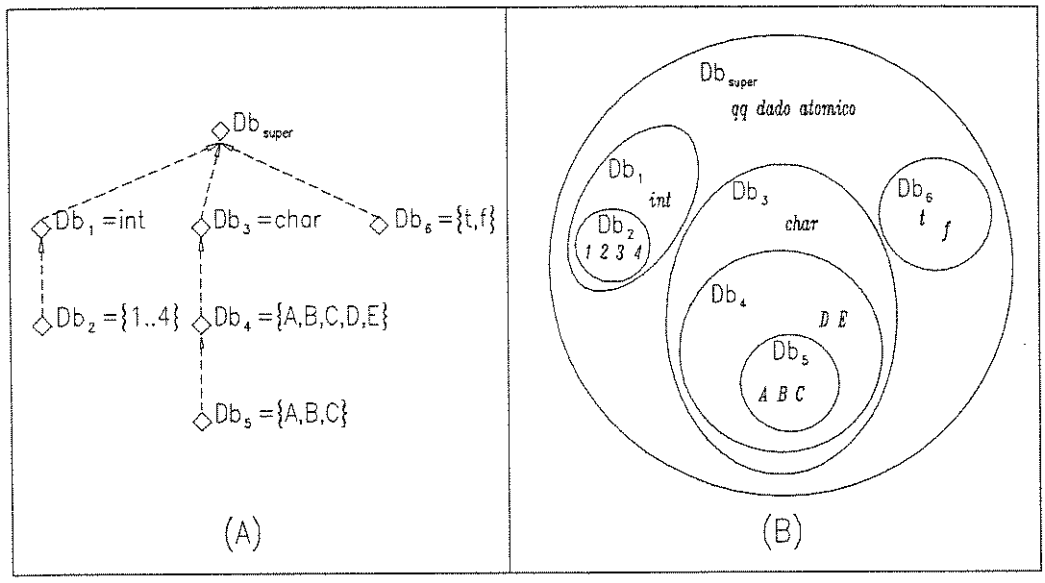

Figura A.1: Hierarquia de domínios básicos.

  - em listas com repetição de elementos (D_{e_1} e D_{e_2})

- docentes e alunos individualmente (D_{e_4}, D_{e_5} e D_{e_6}).

- turmas de docentes e alunos:

  - listas sem repetição de elementos (D_{e_8}, D_{e_9} e $D_{e_{10}}$)

Os domínios encapsulados abstratos D_{e_3}, D_{e_7} e $D_{e_{super}}$ ilustram a reutilização/compartilhamento de definições por subdomínios. A definição formal detalhada destes domínios está no Apêndice D nas Tabelas D.4, D.8, D.9, D.10, D.13 e D.14.

A Figura A.2 mostra a hierarquia de sub-tipicidade entre os domínios encapsulados definidos para a aplicação. Nesta figura para cada domínio lista-se o nome dos comportamentos que formam o anel de encapsulamento, *i.e.*, a interface do domínio encapsulado, e no centro do anel mostra-se o domínio composto utilizado na implementação do domínio encapsulado. No Apêndice D detalha-se a assinatura destes comportamentos.

A hierarquia de domínios encapsulados apresentada na Figura A.2 modela a funcionalidade (comportamento) da aplicação em questão, *i.e.*, a manipulação de grupos, indivíduos e turmas. Existem turmas de alunos de graduação e de pós-graduação, onde alunos de pós são também considerados alunos de graduação; portanto, podem ser incluídos em turmas

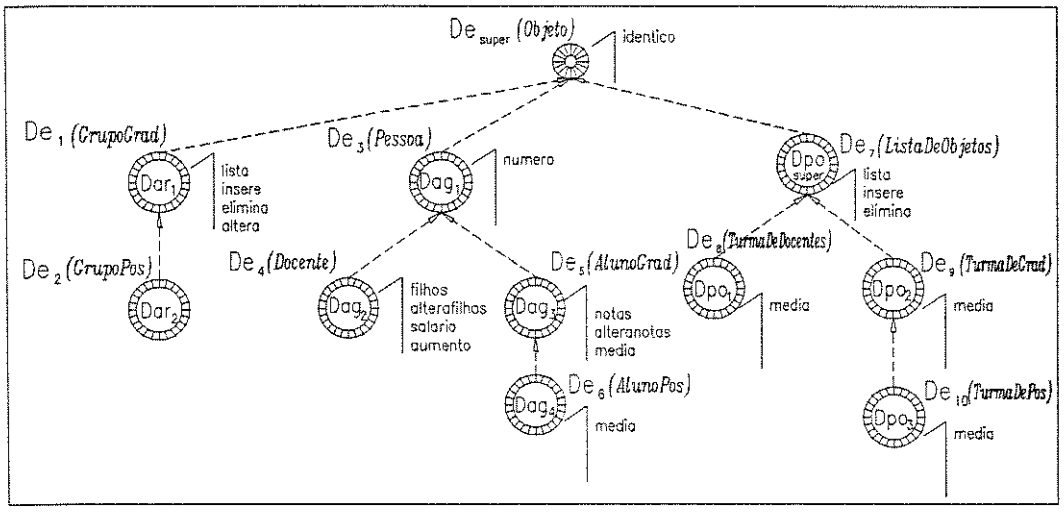

Figura A.2: Detalhamento da hierarquia de domínios encapsulados.

de alunos de graduação. Os grupos de notas também podem ser de graduação ou de pós-graduação, sendo feita a diferenciação nas possibilidades de notas. É possível calcular a média salarial de turmas de docentes e média de notas de turmas de alunos.

Domínios compostos

A escolha de domínios compostos na implementação da hierarquia de domínios encapsulados dá origem implicitamente a uma hierarquia de domínios compostos (Figura A.3), *i.e.*:

- os domínios encapsulados D_{e_1} e D_{e_2} (grupos) utilizam em suas implementações os domínios compostos D_{ar_1} e D_{ar_2} que definem arranjos de um a dois elementos pertencentes aos domínios básicos D_{b_4} e D_{b_5}, respectivamente. A definição formal detalhada dos domínios compostos D_{ar_1} e D_{ar_2} está no Apêndice D na Tabela D.3.

- os domínios encapsulados $D_{e_3}, D_{e_4}, D_{e_5}$, e D_{e_6} (indivíduos) utilizam em suas implementações os domínios compostos $D_{ag_1}, D_{ag_2}, D_{ag_3}$, e D_{ag_4} que definem agregações de elementos pertencentes ao domínio básico e/ou encapsulados. Estes domínios compostos por agregações descrevem a composição de docentes e alunos, *i.e.*, o fato de:

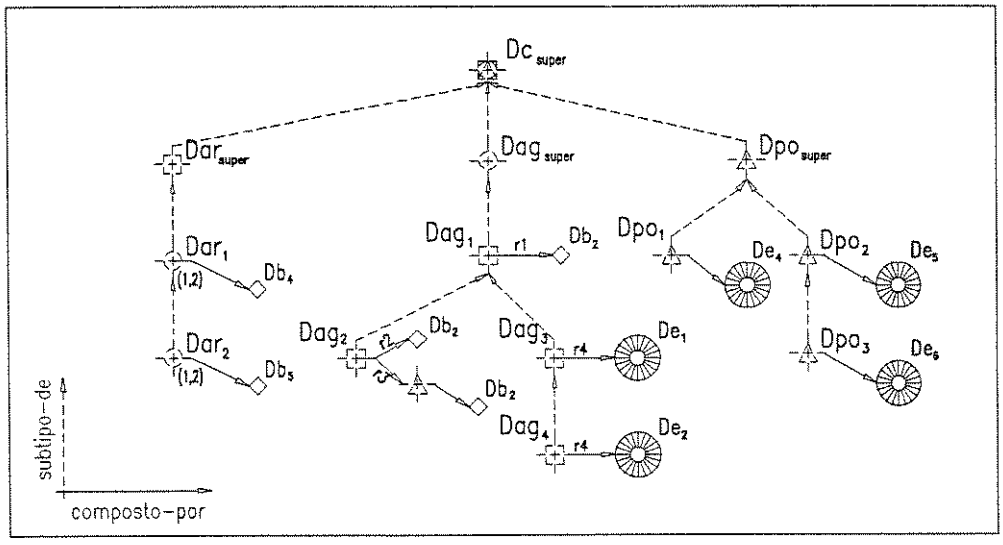

Figura A.3: Detalhamento da hierarquia de domínios compostos, incluindo o grafo de composição dos domínios

- tanto docentes como alunos serem identificados por números de 1 a 4, de docentes terem salário e um conjunto de filhos (ambos representados também por números de 1 a 4), desta forma definindo atributos mono e multi-valorados;
- alunos terem um conjuntos de notas compartilhado por outros, desta forma definindo uma associação 1:n entre aluno e notas.

A definição formal detalhada dos domínios D_{ag_1}, D_{ag_2}, D_{ag_3}, e D_{ag_4} está no Apêndice D nas Tabelas D.5, D.6 e D.7.

- os domínios encapsulados D_{e_7}, D_{e_8}, D_{e_9}, e $D_{e_{10}}$ (turmas) utilizam em suas implementações os domínios compostos $D_{po_{super}}$, D_{po_1}, D_{po_2}, e D_{po_3} que definem conjuntos de elementos pertencentes a domínios encapsulados. Estes domínios compostos por potenciação descrevem a composição de listas em forma de conjuntos de docentes e alunos, possibilitando associações 1:n entre lista de docentes ou alunos e docentes ou alunos, respectivamente. A definição formal detalhada dos domínios D_{po_1}, D_{po_2}, e D_{po_3} está no Apêndice D nas Tabelas D.11 e D.12.

As Figuras A.4 e A.5 reapresentam em diagramas Venn as hierarquias *is-a* de domínios compostos por arranjo, potenciação e agregação. Estas figuras servem para ilustrar as

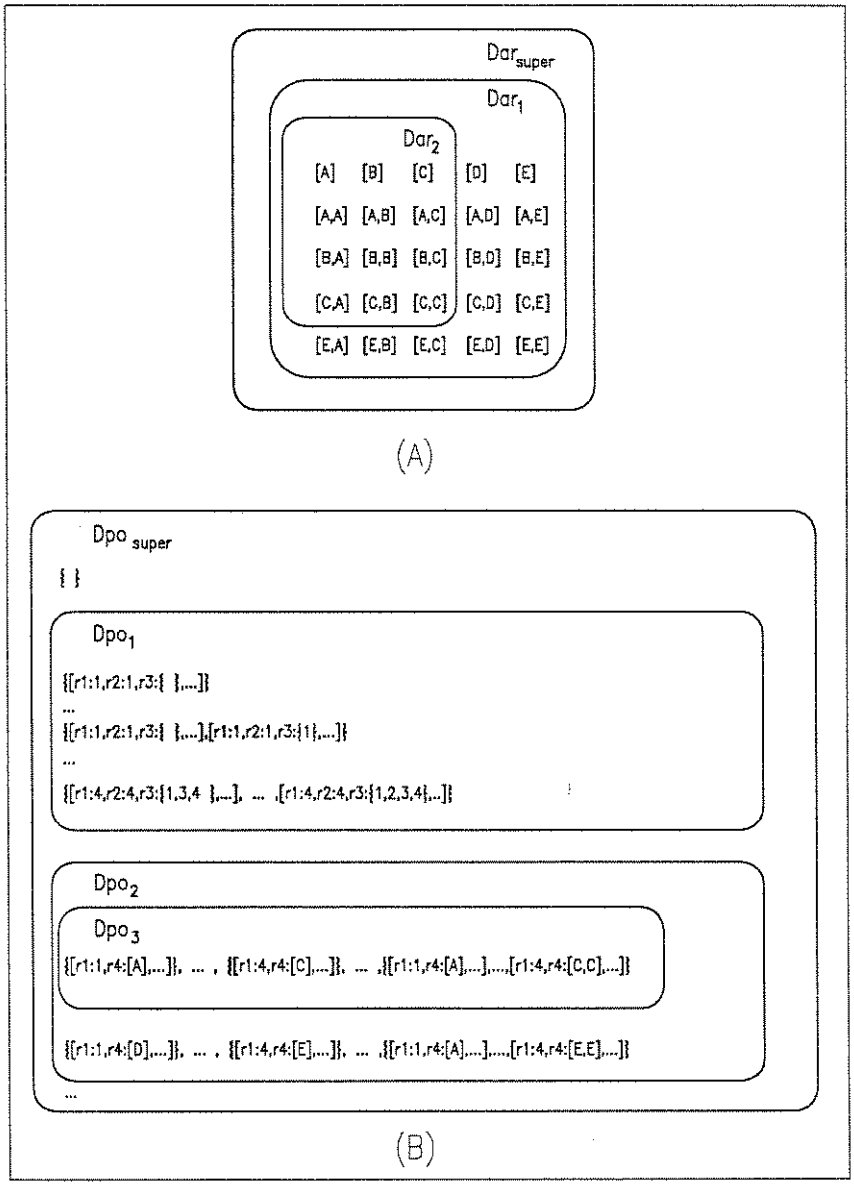

Figura A.4: (A) Diagrama Venn da hierarquia de domínios compostos por arranjo e (B) Diagrama Venn da hierarquia de domínios compostos por potenciação.

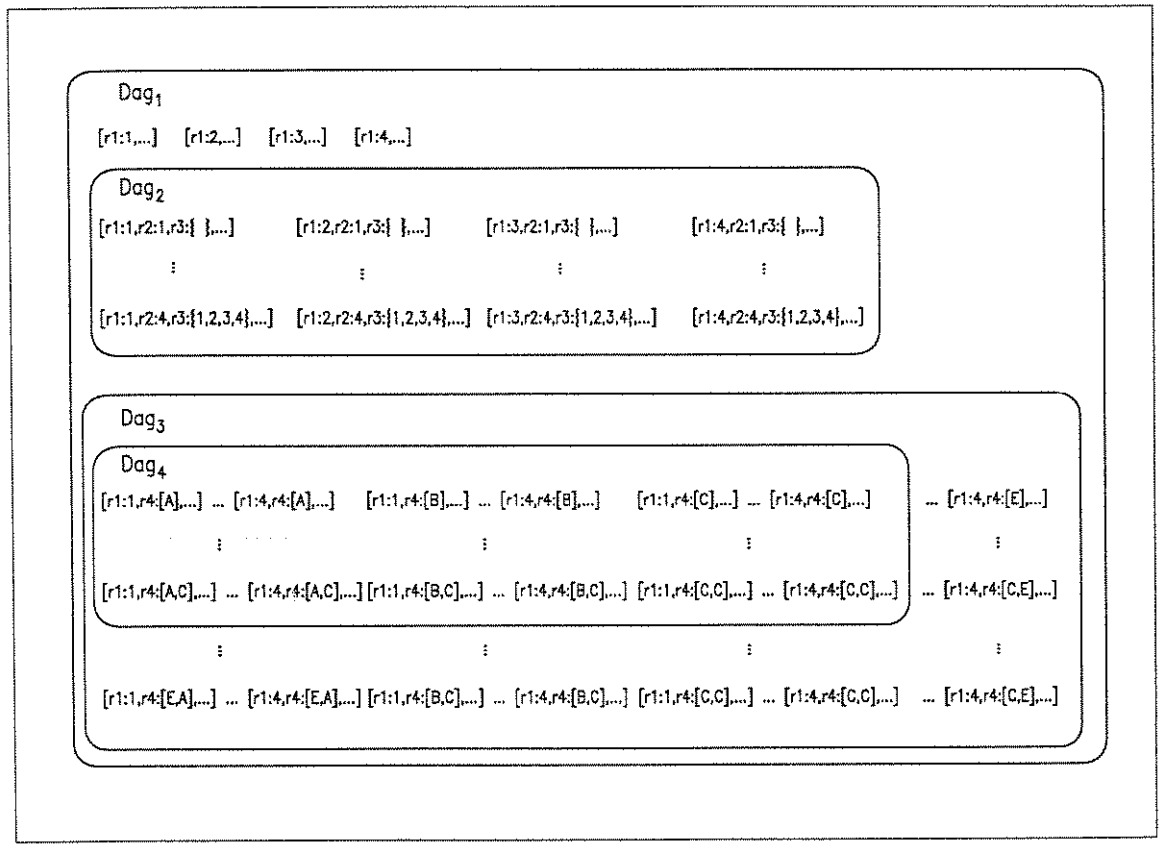

Figura A.5: Diagrama Venn da hierarquia de domínios compostos por agregação.

possíveis combinações dos dados básicos a partir das estruturações escolhidas, por exemplo:

- na Figura A.4 (A) vêm-se todos os possíveis arranjos de notas de graduação (D_{ar_1}) e pós-graduação (D_{ar_2}),
- na Figura A.4 (B) vêm-se todos os possíveis conjuntos de docentes (D_{po_1}), alunos de graduação (D_{po_2}) e alunos de pós-graduação (D_{po_3}) e
- na Figura A.5 vêm-se todas as possíveis agregações que representam pessoas (D_{ag_1}), docentes (D_{ag_2}), alunos de graduação (D_{ag_3}) e alunos de pós-graduação (D_{ag_4}).

Assim como na Figura A.1 (B) deve-se notar nas Figuras A.4 e A.5 a relação de inclusão de conjuntos entre os conjuntos de dados que representam os domínios compostos no nível intencional. A relação de inclusão de conjunto, que implica em polimorfismo, é fácil de ser reconhecida entre os conjuntos de dados dos sub e super-domínios compostos por arranjo e potenciação (ver Figura A.4); entretanto, não é tão fácil de ser reconhecida entre os conjuntos de dados de domínios compostos por agregação.

Para reconhecer a inclusão de conjuntos entre sub e super-domínios compostos por agregação, deve-se lembrar que a definição de um domínio composto por agregação representa qualquer produto cartesiano onde os domínios componentes participam identificados pelos seus respectivos rótulos. Portanto, no exemplo apresentado as agregações (ou n-uplas) que compõem D_{ag_1}, na Figura A.5, são definidas de forma aberta, *i.e.*, `[r1:1,...]` significa qualquer agregação que contenha o dado 1 rotulado por `r1`. Conseqüentemente uma agregação, por exemplo `[r1:1, r2:2, r3:{1},...]` $\in D_{ag_2}$, também pertence a D_{ag_1} pois contém `[r1:1,...]`, e como este fenômeno se repete para todas as agregações de D_{ag_2} com relação a D_{ag_1} diz-se que $D_{ag_2} \subset D_{ag_1}$.

Hierarquia global de domínios

A Figura A.6 apresenta de forma simplificada[1] a hierarquia *is-a* de domínios básicos, compostos e encapsulados da aplicação para este primeiro estudo de caso. Os domínios básicos definem a base sobre a qual novos dados serão compostos. Os domínios compostos formam a base estrutural dos novos dados, descrevendo relacionamentos de composição (atributos) e associações (1:1 e 1:n). Os domínios encapsulados representam a descrição

[1] Mostra-se apenas o relacionamento de sub-tipicidade entre domínios, não se detalhando o comportamento dos domínios encapsulados, nem a composição dos domínios compostos.

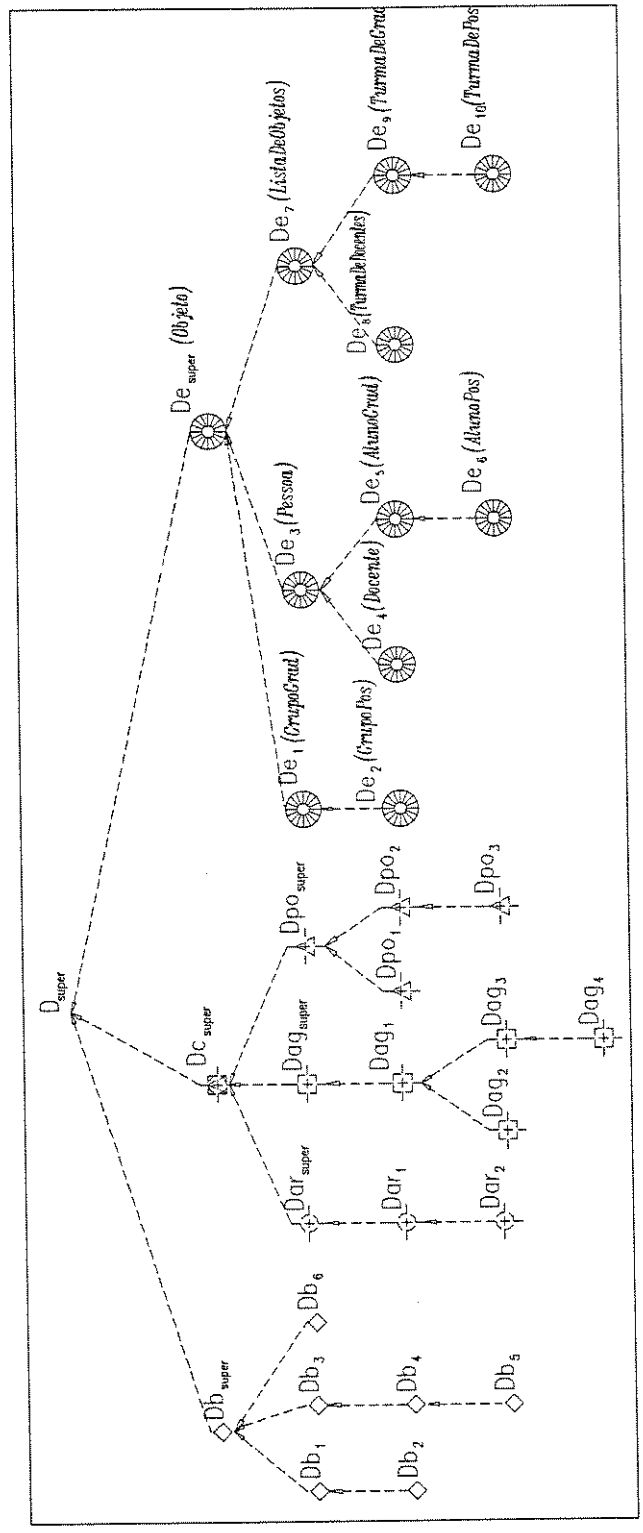

Figura A.6: A hierarquia de super e subdomínios do primeiro estudo de caso.

Tabela A.1: Tabulação dos domínios formalizados.

Tabela	Domínio	Descrição
D.1	D_{b_2}	$\{1, 2, 3, 4\}$
D.1	D_{b_4}	$\{A, B, C, D, E\}$
D.2	D_{b_5}	$\{A, B, C\}$
D.2	D_{b_6}	$\{t, f\}$
D.3	D_{ar_1}	arranjo de notas de graduação
D.3	D_{ar_2}	arranjo de notas de pós-graduação
D.4	D_{e_1}	grupos de notas de graduação
D.4	D_{e_2}	grupos de notas de pós-graduação
D.5	D_{ag_1}	agregação pessoa
D.5	D_{ag_2}	agregação docente
D.6	D_{ag_3}	agregação aluno de graduação
D.7	D_{ag_4}	agregação aluno de pós-graduação
D.8	D_{e_3}	pessoa
D.8	D_{e_4}	docente
D.9	D_{e_5}	aluno de graduação
D.10	D_{e_6}	aluno de pós-graduação
D.11	D_{po_1}	conjunto de docentes
D.11	D_{po_2}	conjunto de alunos de graduação
D.12	D_{po_3}	conjunto de alunos de pós-graduação
D.13	D_{e_7}	lista de objetos
D.13	D_{e_8}	turma de docentes
D.14	D_{e_9}	turma de alunos de graduação
D.14	$D_{e_{10}}$	turma de alunos de pós-graduação

abstrata da aplicação sendo modelada. A Tabela A.1 apresenta a seqüência da descrição formal dos domínios no Apêndice D.

Com o intuito de enriquecer a compreensão deste primeiro estudo de caso, a Figura A.7 reapresenta a hierarquia da Figura A.6 na notação do OMT. A Figura A.8 apresenta um resumo da notação gráfica do OMT. Deve-se notar que todos os domínios encapsulados correspondem a classes na notação do OMT e que neste exemplo existem apenas associações 1:n entre classes e relações de herança, não existindo associações do tipo agregação entre classes. Deve-se notar também que as associações entre classes definidas na notação do OMT estão representadas separadamente na hierarquia de domínios compostos implícita

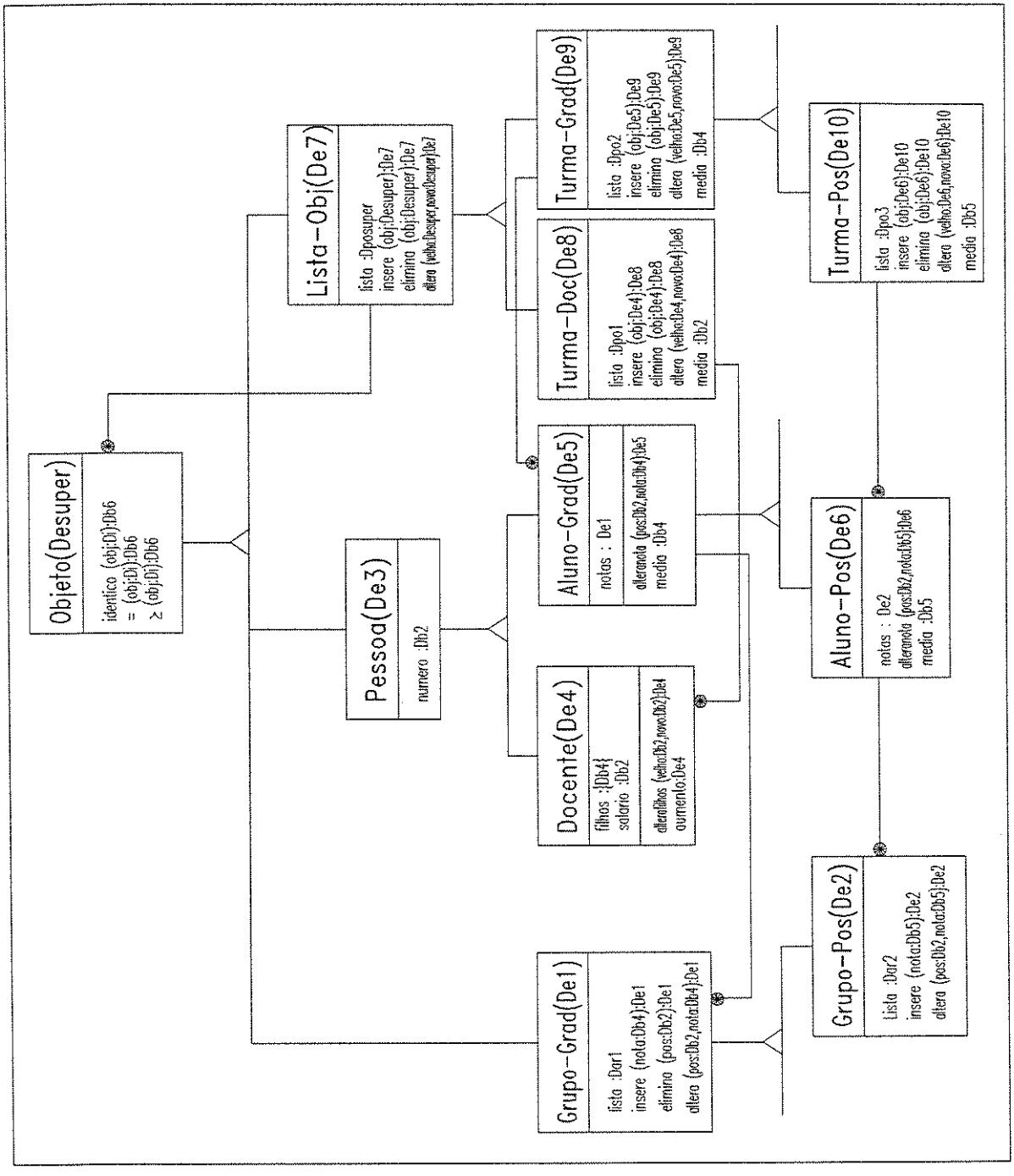

Figura A.7: O primeiro estudo de caso descrito na notação do OMT.

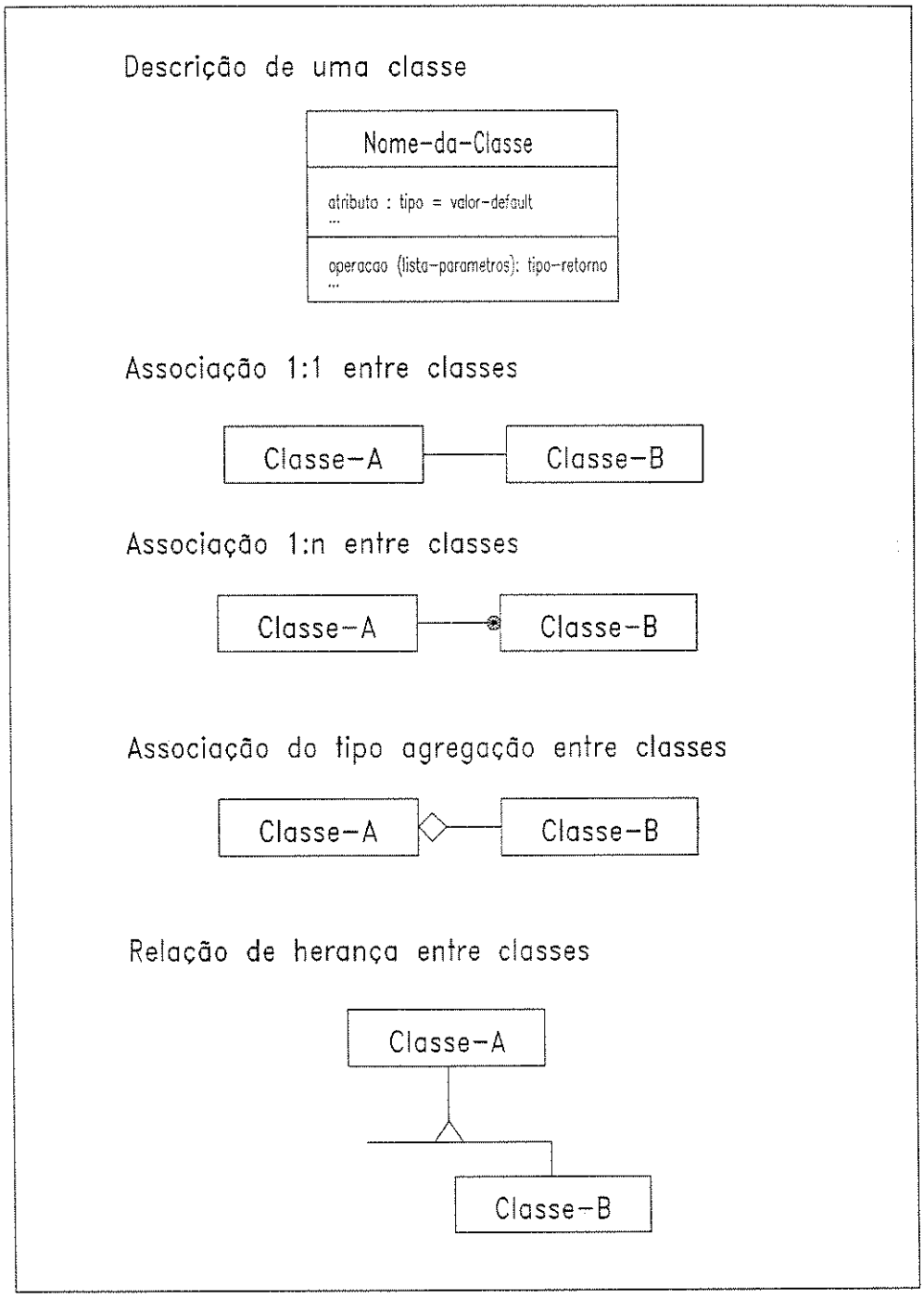

Figura A.8: Resumo da notação gráfica do OMT.

na implementação dos domínios encapsulados. Este fato é conseqüência de domínios encapsulados enfatizarem o comportamento, enquanto que relações de composição ou associações são descritas no modelo referência por domínios compostos; portanto, estas informações estão encapsuladas numa hierarquia de domínio encapsulados. Nas Figuras A.6 e A.2 a hierarquia de domínios encapsulados especifica apenas herança e comportamento.

A seguir uma amostra de como o comportamento de um domínio encapsulado, especificado em termos de assinaturas, foi traduzido para a notação do OMT. Tomar-se-á como exemplo o comportamento do domínio D_{e_4} (Docentes) (Apêndice D, Tabela D.8).

- o comportamento $filhos : D_{e_4} \rightarrow \{D_{b_2}\}$ que representa uma relação do domínio D_{e_4} (Docentes) no domínio $\{D_{b_4}\}$ (conjunto de números de 1 a 4) foi traduzido na notação do OMT para um atributo de nome `filhos` e do tipo $\{D_{b_4}\}$ (conjunto de números de 1 a 4).

- o comportamento $alterafilhos : D_{e_4} \times D_{b_2} \times D_{b_2} \rightarrow D_{e_4}$ que representa uma relação do domínio D_{e_4} (Docentes) com o produto $D_{b_2} \times D_{b_2}$ no próprio domínio D_{e_4} foi traduzido na notação do OMT em uma operação de nome `alterafilhos` com os parâmetros de entrada `velho` e `novo` ambos do domínio básico D_{b_2} com um valor de retorno do tipo D_{e_4}.

Deve-se notar que o modelo referência inclui uniformemente no comportamento do domínio tanto atributo/associações (relacionados a composição) como operações; representados por assinaturas de relações. Portanto, o modelo referência é mais genérico na descrição da interface do domínio, comparando-se com a descrição de interface de tipo do padrão ODMG-93.

A.2.2 Instâncias no nível concreto

Extensão dos domínios básicos

A Figura A.9(A) mostra a hierarquia de domínios básicos no nível intencional bem com suas respectivas instâncias no nível concreto. Deve-se notar que o nível concreto é composto por instâncias agrupadas em extensões (Figura A.9(B)). Instâncias são pares *(identificador, estado)* presentes na aplicação. Instâncias pertencentes a extensões associadas a domínios básicos são denominadas valores atômicos, pois o identificador é igual ao estado, e o estado

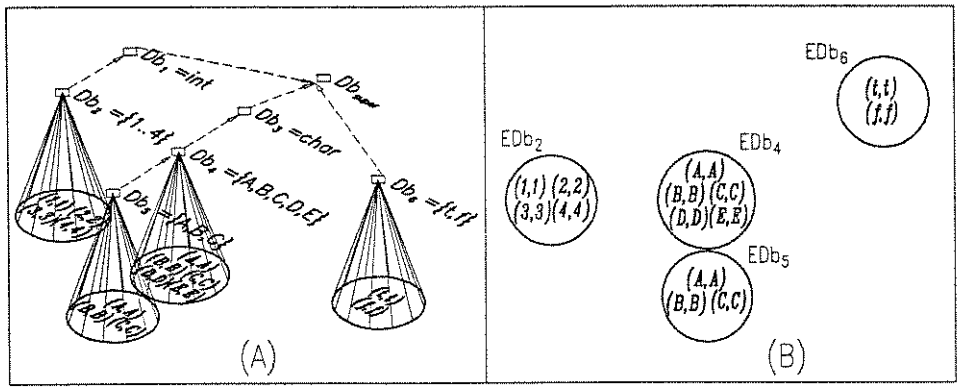

Figura A.9: (A) Níveis intencional e concreto da hierarquia de domínios básicos e (B) Instâncias no nível concreto dos domínios básicos.

é um dado indivisível. O conjunto de dados que representa um domínio no nível intencional apresenta todos os possíveis estados das instâncias nas extensões destes no nível concreto.

Extensões dos domínios compostos e encapsulados

As Figuras A.10 e A.11 mostram as hierarquias de domínios compostos e encapsulados no nível intencional bem como suas respectivas instâncias no nível concreto. A parte (B) de ambas figuras mostra apenas as instâncias nas extensões no nível concreto. Deve-se notar a existência de dois tipos de instâncias: valores e objetos. Instâncias do tipo valor são pares *(identificador, estado)* onde o identificador é igual ao estado. Instâncias do tipo objeto são pares *(identificador, estado)* onde o identificador é diferente do estado e identificador pertence ao conjunto IDO de identificadores. Deve-se notar ainda valores e objetos do tipo simples e complexos, por exemplo:

- valores simples:
  - `([A],[A])` $\in \{ED_{ar_1}, ED_{ar_2}\}$ — o valor pertence às extensões dos domínios compostos por arranjo D_{ar_1} e D_{ar_2} (Figura A.10). Deve-se notar que o identificador e estado são iguais. O estado por sua vez faz referência apenas a valores atômicos, *i.e.*, `A` é o identificador para o valor `(A,A)`. Conseqüentemente o valor `([A],[A])` é denominado valor simples. Este valor é utilizado para compor grupos de notas de graduação ou de pós-graduação.

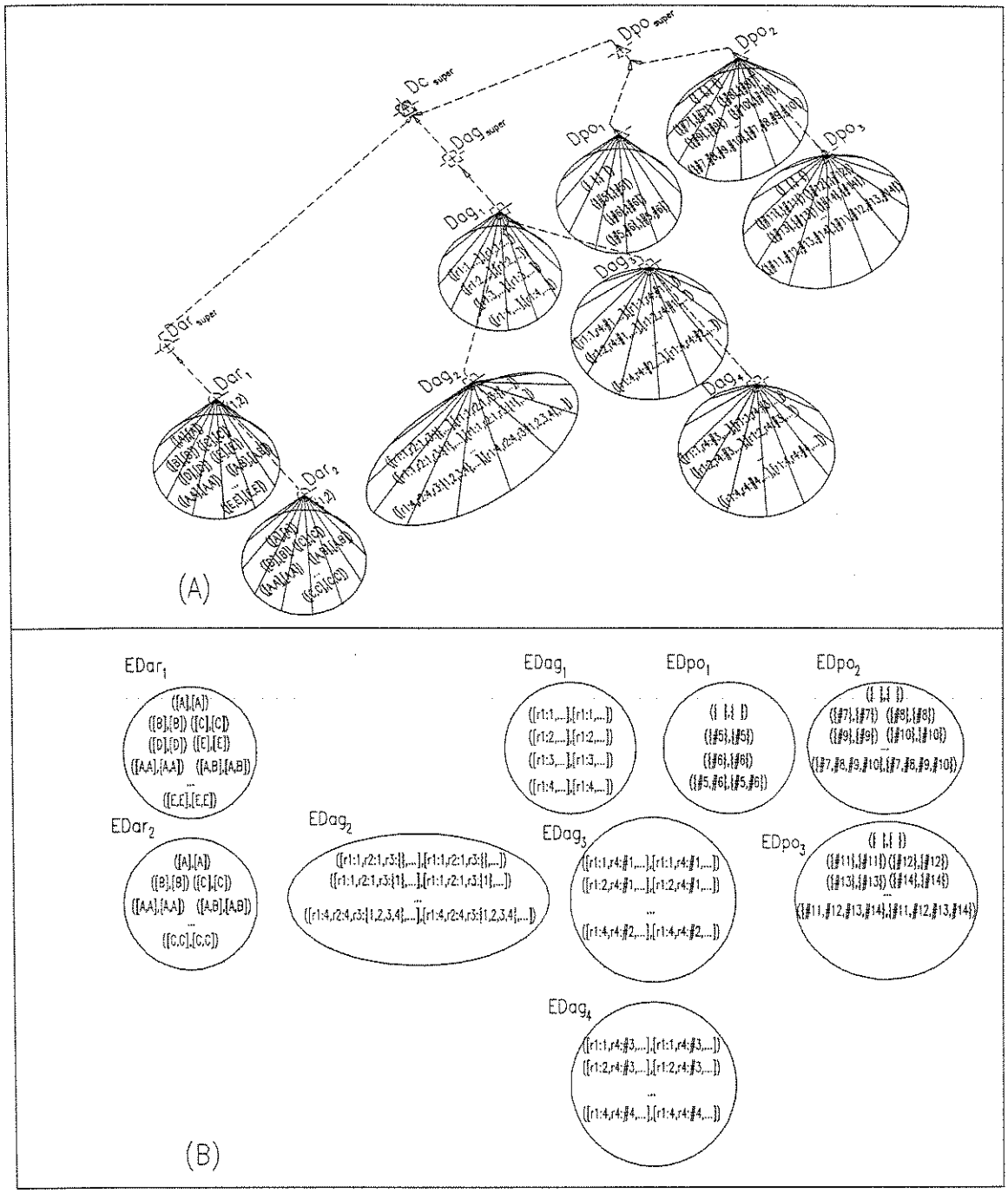

Figura A.10: (A) Níveis intencional e concreto da hierarquia de domínios compostos e (B) Instâncias no nível concreto dos domínios compostos.

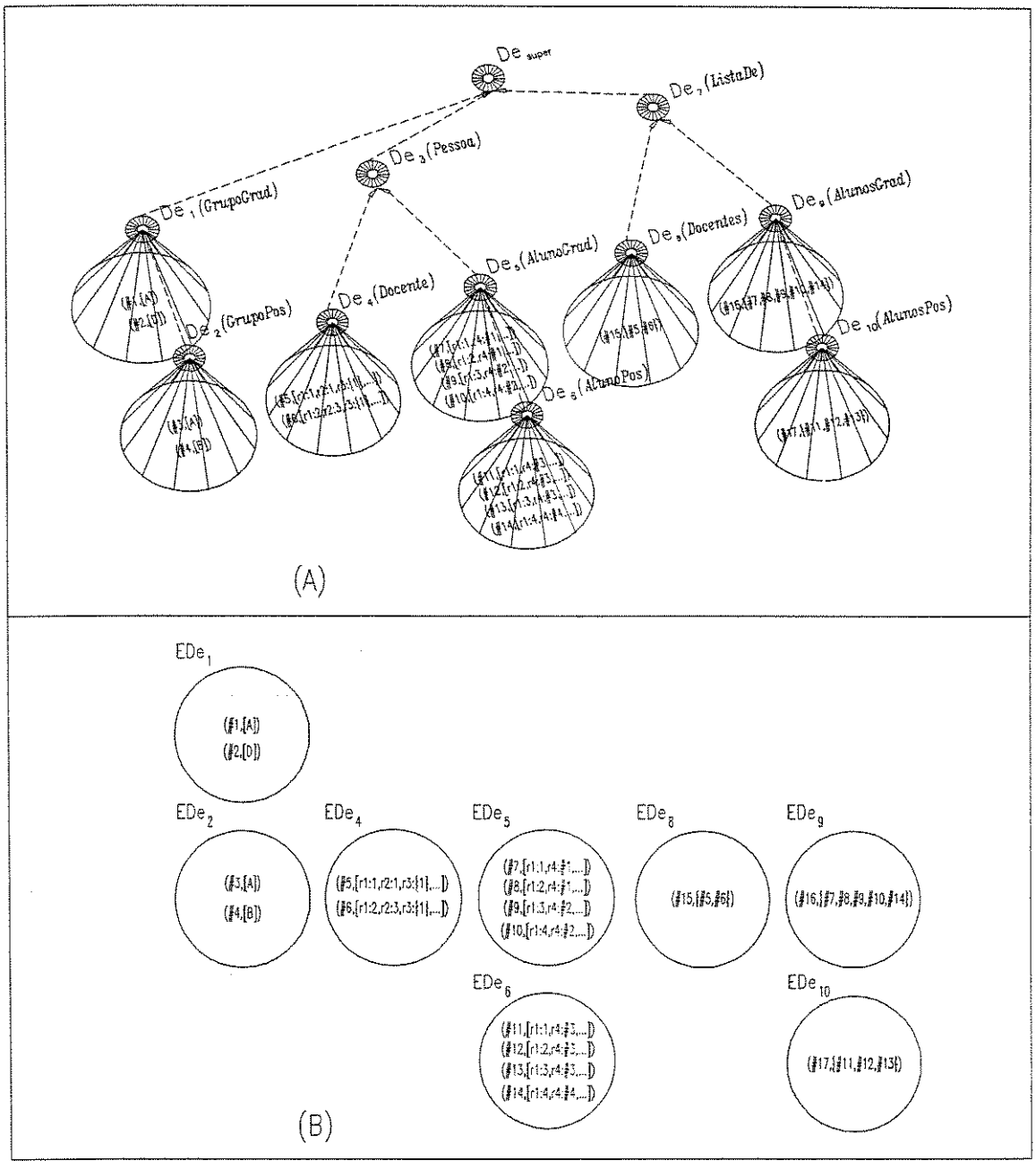

Figura A.11: (A) Níveis intencional e concreto da hierarquia de domínios encapsulados e (B) Instâncias no nível concreto dos domínios encapsulados.

- `([r1:1,r2:1,r3:{1},...],[r1:1,r2:1,r3:{1},...])` $\in ED_{ag_2}$ — o valor pertence à extensão do domínio composto por agregação D_{ag_2} (Figura A.10). Deve-se notar que o identificador e estado são iguais. Assim como no exemplo anterior, o estado é estruturado e faz referência apenas a valores atômicos, neste caso ao valor `(1,1)`. Conseqüentemente o valor `([r1:1,r2:1,r3:{1},...], [r1:1,r2:1,r3:{1},...])` é denominado valor simples. Este valor é utilizado para compor docentes.

- objetos simples :
  - `(#1,[A])` $\in ED_{e_1}$ — o objeto pertence à extensão do domínio encapsulado grupos de graduação D_{e_1} (Figura A.11). Deve-se notar que o identificador e estado são diferentes. O objeto é identificado por `#1` e seu estado é o identificador `[A]`. Portanto, como o objeto faz referência ao valor simples `([A],[A])`, o objeto também é denominado objeto simples. Este objeto poderá ser usado em alunos de graduação que participam de um grupo que tem apenas uma nota igual a `A`.
  - `(#6,[r1:1,r2:1,r3:{1}, ...])` $\in ED_{e_4}$ — o objeto pertence à extensão do domínio docentes D_{e_4} (Figura A.11). Deve-se notar que o identificador e estado são diferentes. O objeto é identificado por `#6` e seu estado é o identificador `[r1:1,r2:1, r3:{1},...]`. Portanto, como o objeto faz referência a um valor simples, *e.g* `([r1:1,r2:1,r3:{1},...], [r1:1,r2:1,r3:{1},...])`, o objeto também é denominado objeto simples.

- valor complexo:
  - `([r1:1,r4:#1,...],[r1:1,r4:#1,...])` $\in ED_{ag_3}$ — o valor pertence à extensão do domínio agregação alunos de graduação D_{ag_3} (Figura A.10). Deve-se notar que o identificador e estado são iguais. Entretanto, como o estado é composto entre outros também por identificadores de objetos, *e.g.*, `#1`, o valor é denominado valor complexo.

- objeto complexo:
  - `(#8,[r1:1,r4:#1,...])` $\in ED_{e_5}$ — o objeto pertence à extensão do domínio alunos de graduação D_{e_5} (Figura A.11). Deve-se notar que o identificador e estado são diferentes. O objeto é identificado por `#8` e o estado do objeto

é o identificador do valor `([r1:1,r4:#1,...],[r1:1,r4:#1,...])`. Como o valor sendo referenciado pelo objeto é um valor complexo o objeto também é denominado objeto complexo.

O que interessa no nível concreto para a aplicação sendo modelada são os objetos, os valores servem apenas para compor objetos. A diferenciação entre valor atômico ou simples e complexo serve para implementar ou não o compartilhamento de dados como será descrito a seguir:

- *deseja-se compartilhar notas entre alunos pois alunos trabalham em grupos*[2]*:*

  Para descrever um exemplo concreto da situação acima utilizaremos os objetos `(#7,[r1:1,r4:#1,...])` e `(#8,[r1:2,r4:#1,...])`, que representam alunos de graduação (Figura A.11). Deve-se notar que ambos fazem referência ao objeto `(#1,[A])` que representa um grupo de graduação que tem uma única nota igual a `A`. Neste caso, os alunos de graduação identificados por `#7` e `#8` trabalham no mesmo grupo e, portanto, compartilham notas. Se o estado de qualquer destes objetos for modificado com relação ao rótulo r_4, isto significa que o aluno mudou de grupo. Entretanto, se o estado de ambos não for alterado e o estado do objeto compartilhado através do rótulo r_4 for modificado, por exemplo, para `(#1,[A,E])`, isto significa que ambos continuam trabalhando em grupo e tiveram seu conjunto de notas alterado (por sinal com uma nota bem ruim!).

  Deve-se notar que o compartilhamento de dados foi obtido utilizando-se objetos complexos, *i.e.*, objetos cujos estados são valores complexos. Neste caso tem-se representado entre alunos de graduação e notas um relacionamento (associação) 1:n.

- *docentes têm filhos mas não existe o interesse em representar o compartilhamento de filhos entre docentes*[3]*:*

  Para descrever um exemplo concreto da situação acima utilizaremos os objetos `(#5,[r1:1,r2:2,r3:{1},...])` e `(#6,[r1:1,r2:2,r3:{1},...])` que representam docentes. Deve-se notar que no estado de ambos está presente o conjunto `{1}` associado ao rótulo r_3 que representa o conjunto de filhos. Neste caso temos que

[2] *O que quer dizer que alunos têm uma relação de associação com notas através de grupos. Portanto, associações permitem o compartilhamento de dados.*

[3] *O que quer dizer que filho é um atributo em docentes e não uma associação em docentes. Portanto, atributos implicam no não compartilhamento de dados.*

ambos docentes compartilham o mesmo conjunto de filhos. Da mesma forma como no exemplo anterior, se modificarmos o conjunto associado ao rótulo r_3, *i.e.*, o estado destes objetos com relação ao rótulo r_3, isto significa que se estaria trocando o conjunto de filhos dos docentes. Entretanto, uma modificação no estado do objeto `(#5,[r1:1,r2:2,r3:{1},...])` não se propaga para o objeto `(#6,[r1:1,r2:2, r3:{1},...])`, ou vice-versa, pois estes objetos não se referenciam. Além do mais, deve-se notar que não é possível modificar-se o conjunto `{1}` de tal forma que ambos objetos compartilhem esta modificação. Neste caso, o compartilhamento de dados não é explícito; conseqüentemente, é considerado inexistente.

Deve-se notar que o não compartilhamento de dados foi obtido utilizando-se objetos simples, *i.e.*, objetos cujos estados são valores simples. Neste caso, os rótulos r_1 e r_2 no estado destes objetos são equivalentes a atributos mono-valorados e o rótulo r_3 é equivalente a um atributo multi-valorado.

Com a discussão dos exemplos acima pode-se dizer que rótulos no estado de objetos equivalem a:

atributos mono-valorados: se estiver associado a um identificador pertencente a um domínio básico,

atributos multi-valorados: se estiver associado a um identificador pertencente a um domínio composto por arranjo ou potenciação cuja extensão possuir apenas valores simples,

atributos compostos (ou aninhados): se estiver associado a um identificador pertencente a um domínio composto por agregação cuja extensão possuir apenas valores simples,

associação 1:1: se estiver associado a um identificador de objeto e o estado do objeto sendo referenciado pertence a um domínio composto por agregação,

associação 1:n: se estiver associado a um identificador de objeto e o estado do objeto sendo referenciado pertencer a um domínio composto por arranjo ou potenciação.

Toda a discussão acima aborda a caracterização de composição e associação de objetos. Entretanto, o exemplo inicial apresentado nesse apêndice também requer dados com

polimorfismos de inclusão, *i.e.*, alunos de pós-graduação também podem cursar disciplinas de graduação; portanto, também podem fazer o papel de alunos de graduação. Deve-se notar que o objeto (#16,{#7,#8,#9,#10,#14}) $\in ED_{e_9}$ representa uma lista de alunos de graduação (Figura A.11). O estado desse objeto contém referências a objetos das extensões $\{ED_{e_5}, ED_{e_6}\}$; portanto, extensões do domínio de alunos de graduação D_{e_5} e do domínio de alunos de pós-graduação D_{e_6}. Neste caso temos incluído um aluno de pós numa turma de alunos de graduação, o que era desejado no exemplo.

A.3 Segundo e terceiro estudos de caso

O estudo de caso modelado na Seção A.2 exemplificou relações de herança e de associações 1:n entre classes, atributos e comportamento de classes. Esta seção apresentará dois estudos de casos mais simples: um que trata relações de associações 1:1 entre classes e outro que trata associações do tipo agregação entre classes. Estes estudos de caso serão inicialmente apresentados na notação do OMT e em seguida traduzidos para a notação proposta no modelo referência proposto, *i.e.*, em termos de hierarquias de domínios básicos, compostos e encapsulados e comportamentos correspondentes.

A.3.1 Associação 1:1

Um exemplo de associação 1:1 entre classes é a relação única existente entre país e cidade em termos de capital, *i.e.*, todo país tem apenas uma capital[4] e uma cidade pode ser capital de apenas um país. Este exemplo foi obtido de [RBP+91] e é apresentado na notação do OMT na Figura A.12(A) onde, por simplicidade, se especifica apenas a associação 1:1 entre as classes `País` e `Cidade`, sendo que a composição ou o comportamento da classe não são detalhados. O mesmo exemplo é traduzido na Figura A.12(B) para a notação do modelo referência proposto. Na parte (B) da figura optou-se por clareza incluir o comportamento dos domínios encapsulados equivalente as classes da parte (A) da figura, bem como possíveis propriedades de uma cidade como nome e número de habitantes. Deve-se notar que a associação entre país e cidade é especificada nos dois sentidos entre os domínios compostos utilizados na implementação dos domínios encapsulados, *i.e* `país-tem-capital-cidade` e `cidade-é-capital-de-país`. Isto implica na possibilidade de fazer consultas em ambos

[4]Neste exemplo não estão representados países com mais de uma capital.

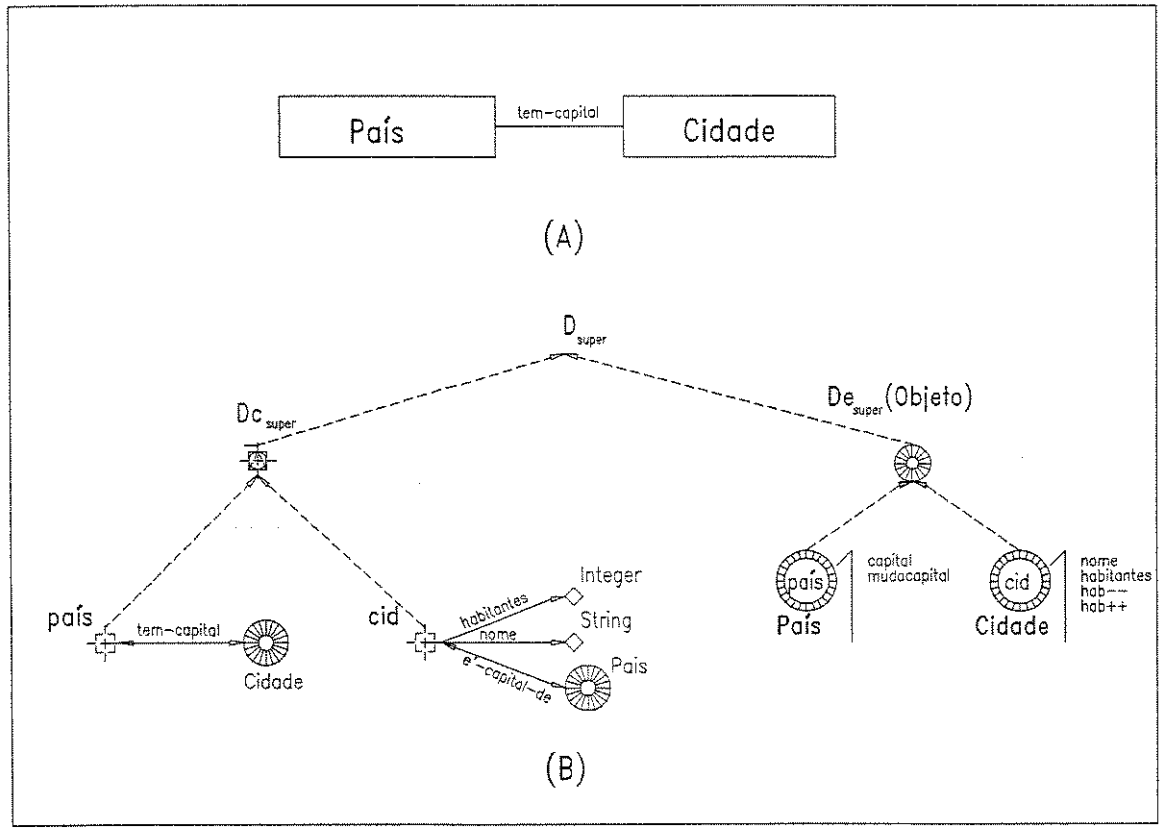

Figura A.12: (A) Associação 1:1 entre classes na notação do OMT e (B) O mesmo exemplo apresentado na notação do modelo referência.

os sentidos da relação. Poderia se ter implementado a relação apenas no sentido de país para cidade se não fosse necessário saber de qual país uma cidade é capital. Este tipo de decisão depende muito de opções de projeto. Portanto, deve-se notar que o modelo referência proposto é flexível o suficiente para adaptar-se a decisões diversas.

A.3.2 Agregação

Um exemplo de classe formada por agregação de outras classes é a possível relação entre documento, parágrafo e sentença. Uma visão simplificada de documento é de um documento ser composto por uma seqüência de parágrafos, um parágrafo ser composto por uma seqüência de sentenças e, por sua vez, uma sentença ser composta por uma seqüência de caracteres. Este exemplo também foi obtido de [RBP+91] e é descrito na notação do OMT na Figura A.13(A), onde apresentam-se as relações de agregação entre as classes `Documento`, `Parágrafo` e `Sentença` especificando-se também comportamento destas classes.

Deve-se notar um processo acumulativo de comportamento da classe componente para a classe composta conseqüência dos relacionamentos de agregação. A classe `Sentença` tem os comportamentos de cálculo do total de palavras na sentença e impressão da sentença. A classe `Parágrafo` também tem estes comportamentos redefinidos de tal forma que o comportamento de cálculo de palavras no parágrafo faz uso do comportamento do cálculo de palavras nas sentenças. Deve-se notar que este comportamento não é herdado mas sim composto pelo outro. O mesmo deve acontecer com o comportamento impressão de parágrafo com relação a impressão de sentença. A classe `Parágrafo` também acrescenta novos comportamentos como consultar e alterar o autor de um parágrafo. A classe `Documento` acumula os comportamentos de cálculo de total de palavras, consulta de autor e impressão que por sua vez fazem uso dos comportamentos correspondentes da classe componente `Parágrafo`. A classe `Documento` também tem comportamentos próprios como consulta e alteração da data do documento.

A Figura A.13(B) reapresenta o exemplo na notação do modelo referência proposto. Deve-se notar que a hierarquia de domínios encapsulados apresenta apenas o comportamento e relações de herança entre as classes apresentadas na parte (A) da figura, sendo que o relacionamento de agregação entre estas classes é especificado na hierarquia de composição dos domínios compostos utilizados na implementação dos domínios encapsulados.

Deve-se notar que no modelo referência proposto tanto uma associação do tipo 1:1/1:n

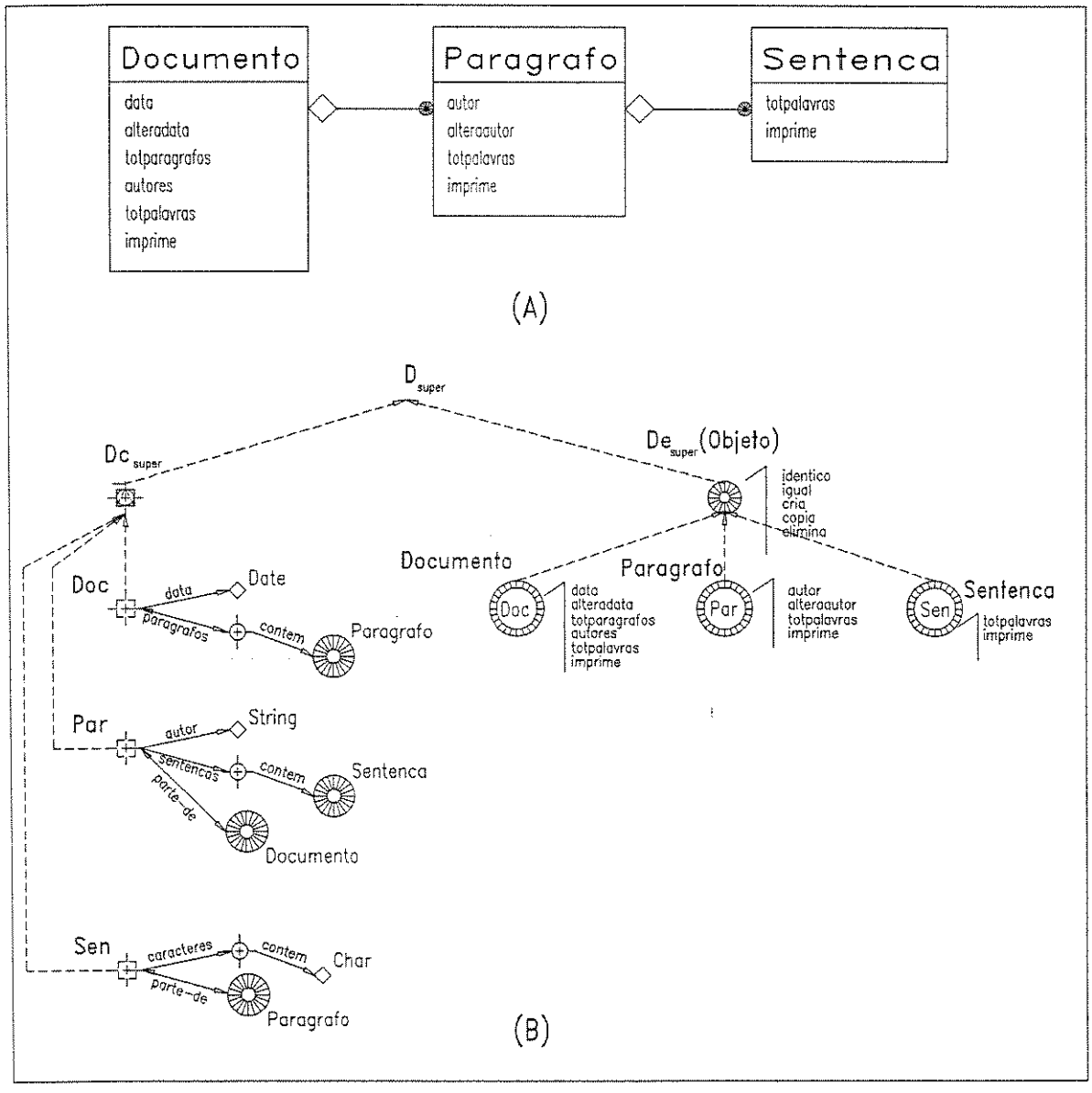

Figura A.13: (A) Seqüência de agregações de classes na notação do OMT e (B) O mesmo exemplo apresentado na notação do modelo referência.

ou do tipo agregação são representadas dentro de um domínio composto por agregação, resultado da uniformidade no formalismo matemático adotado nesse trabalho. Entretanto, existe uma diferenciação em termos de comportamento. No caso de associação do tipo agregação existe um acúmulo de comportamento que não acontece na associação 1:1 ou 1:n.

A.4 Análise dos estudos de caso

Neste apêndice procurou-se demonstrar em três estudos de casos toda a potencialidade e abrangência do modelo referência de objetos proposto e formalizado. Os estudos foram descritos informalmente e graficamente, tanto na notação gráfica proposta para o modelo referência quanto na notação gráfica do OMT. Somente o primeiro exemplo foi totalmente formalizado e esta formalização é apresentada nas tabelas do Apêndice D.

O primeiro estudo de caso mostrou como classes, propriedades de classes (como atributos e comportamentos) e relações entre classes (como associações 1:n e herança) são representadas no modelo referência. Mostrou-se que a hierarquia de domínios encapsulados equivale à hierarquia *is-a* de classes onde se modela apenas o comportamento da aplicação sendo que propriedades como atributos e associações estão implícitas ou encapsuladas. A implementação dos domínios encapsulados implica na definição correspondente de uma hierarquia de domínios compostos. Na hierarquia de domínios compostos especifica-se a composição dos domínios, conseqüentemente explicitam-se seus inter-relacionamentos.

Pode-se dizer então que a hierarquia de domínios encapsulados define sintaticamente as funcionalidades do sistema, pois comportamentos são definidos apenas por assinaturas. A hierarquia de domínios compostos define a parte estrutural do sistema. A implementação dos domínios encapsulados associa a hierarquia de funcionalidade e a hierarquia de estruturação.

O segundo e terceiro estudos de caso mostraram como relações do tipo associação 1:1/1:n e do tipo agregação são representadas uniformemente no modelo referência. Esta uniformidade está de acordo com a própria definição de agregação em [RBP+91] que classifica agregação como um tipo de associação; portanto, nada mais coerente do que representá-las estruturalmente num modelo referência da mesma forma. Foi apontado também que a diferença entre associação e agregação está na representação do comportamento associado às classes que estas representam, o que também foi possível de ser representado no exemplo pelos recursos do modelo referência.

Apêndice B

Nomenclatura e Simbologia Adotada no Formalismo

Este apêndice apresenta a nomenclatura (Tabelas B.1, B.2, B.3, B.4, B.5, B.6 e B.7) e simbologia (Tabela B.8) adotadas no formalismo matemático do Capítulo 4.

Tabela B.1: Nomenclatura adotada para universos.

Nome	Universo
$\mathcal{D}$	universo dos domínios de dados
$\mathcal{DAG}$	universo dos domínios compostos por agregação
$\mathcal{DAR}$	universo dos domínios compostos por arranjo
$\mathcal{DB}$	universo dos domínios básicos, *i.e.* atômicos
$\mathcal{DC}$	universo dos domínios de dados compostos (ou universo dos domínios compostos)
$\mathcal{DE}$	universo dos domínios de dados encapsulados
$\mathcal{DPO}$	universo dos domínios compostos por potenciação
$\mathcal{M}$	universo dos comportamentos dos domínios de dados
$\mathcal{MAG}$	universo dos comportamentos dos domínios compostos por agregação
$\mathcal{MAR}$	universo dos comportamentos dos domínios compostos por arranjo
$\mathcal{MB}$	universo dos comportamentos dos domínios básicos
$\mathcal{MC}$	universo dos comportamentos dos domínios compostos
$\mathcal{ME}$	universo dos comportamentos dos domínios encapsulados
$\mathcal{MPO}$	universo dos comportamentos dos domínios compostos por potenciação

Tabela B.2: Nomenclatura adotada para domínios.

Nome	Domínios
D_i	um domínio de dados $\in \mathcal{D}$
D_{ag_i}	um domínio de dados compostos por agregação $\in \mathcal{DAG}$
D_{ar_i}	um domínio de dados compostos por arranjo $\in \mathcal{DAR}$
D_{b_i}	um domínio de dados básicos $\in \mathcal{DB}$
D_{c_i}	união dos domínios de dados compostos $\in \mathcal{DC}$
D_{e_i}	um domínio de dados encapsulados $\in \mathcal{DE}$
D_{po_i}	um domínio de dados compostos por potenciação $\in \mathcal{DPO}$

Tabela B.3: Nomenclatura adotada para extensões de domínios.

Nome	Extensões
ED_{b_i}	extensão do domínio básico D_{b_i}
ED_{c_i}	extensão do domínio composto D_{c_i}
ED_{e_i}	extensão do domínio encapsulado D_{e_i}

Tabela B.4: Nomenclatura adotada para funções específicas.

Nome	Funções
`Comp`	composição de domínios ou dados
`CompID`	composição de identificadores
`Desref`	desreferencia o estado de uma instância
`estado`	estado da instância
`identificador`	identificador da instância
fid_{b_i}	função identidade entre conjunto de identificadores associados a domínios básicos
fid_{c_i}	função identidade entre conjunto de identificadores associados a domínios compostos
foa_{e_i}	função do conjunto de identificador associado a um domínio encapsulado no conjunto de identificador associado a um domínio básico
fo_{e_i}	função do conjunto de identificador associado a um domínio encapsulado no conjunto de identificador associado a um domínio composto

Tabela B.5: Nomenclatura adotada para conjunto de identificadores.

Nome	Conjunto de identificadores
ID_{b_i}	conjunto de identificadores associado ao domínio D_{b_i}
ID_{c_i}	conjunto de identificadores associado ao domínio D_{c_i}
ID_{e_i}	conjunto de identificadores associado ao domínio D_{e_i}
IDO	conjunto de identificadores associado a todos domínios encapsulados

Tabela B.6: Nomenclatura adotada para implementações.

Nome	Implementações
I_{M_i}	conjunto implementação das relações do conjunto de assinaturas M_i
I_{Mag_i}	idem para o conjunto de assinaturas M_{ag_i}
I_{Mar_i}	idem para o conjunto de assinaturas M_{ar_i}
I_{Mb_i}	idem para o conjunto de assinaturas M_{b_i}
I_{Me_i}	idem para o conjunto de assinaturas M_{e_i}
I_{Mpo_i}	idem para o conjunto de assinaturas M_{po_i}
$Imp(f)$	implementação da função f
$Imp(M_{ag_i})$	implementação do comportamento M_{ag_i} utilizando o domínio D_{ag_i} e o conjunto implementação de relações I_{Mag_i}
$Imp(M_{ar_i})$	implementação do comportamento M_{ar_i} utilizando o domínio D_{ar_i} e o conjunto implementação de relações I_{Mar_i}
$Imp(M_{b_i})$	implementação do comportamento M_{b_i} utilizando o domínio D_{b_i} e o conjunto implementação de relações I_{Mb_i}
$Imp(M_{po_i})$	implementação do comportamento M_{po_i} utilizando o domínio D_{po_i} e o conjunto implementação de relações I_{Mpo_i}

Tabela B.7: Nomenclatura adotada para conjuntos de assinaturas que definem o comportamento de domínios.

Nome	Comportamentos
M_i	conjunto de assinaturas de relações que descreve o comportamento do domínio $D_i \in \mathcal{D}$
M_{ag_i}	o comportamento do domínio $D_{ag_i} \in \mathcal{DAG}$
M_{ar_i}	idem para o comportamento do domínio $D_{ar_i} \in \mathcal{DAR}$
M_{b_i}	idem para o comportamento do domínio $D_{b_i} \in \mathcal{DB}$
M_{c_i}	idem para o comportamento de um domínio de dados compostos pertencente ao universo $\mathcal{DC}$
M_{e_i}	idem para o comportamento do domínio $D_{e_i} \in \mathcal{DE}$
M_{po_i}	idem para o comportamento do domínio $D_{po_i} \in \mathcal{DPO}$

Tabela B.8: Tabela de símbolos.

Símbolo	**Descrição**
$\exists$	existe
$\exists_1$	existe e é único
$\forall$	para todos ou qualquer
$\mid$	tal que
$\mapsto$	implica em ou mapeia em
$\in$	pertence a
$\subset$	está contido em
$\subseteq$	está contido em ou é igual a
$\supset$	contém
$\supseteq$	contém ou é igual a
$\cup$	união
$\cap$	interseção
$=$	igual ou idêntico
$\simeq$	igualdade rasa
$\sim$	igualdade total
$\preceq$	é comparável a ou é igual a
$\times$	produto cartesiano
$\times_k D_i$	produto cartesiano de k domínios
	$D_1 \times D_2 \times \ldots \times D_k$
$\times_k r_i : D_i$	produto cartesiano de k domínios rotulados
	$r_1 : D_1 \times r_2 : D_2 \times \ldots \times r_k : D_k$
r_i	um rótulo
	um símbolo que identifica um domínio ou dado
$r_i : D_i$	um domínio rotulado
	um domínio identificado pelo símbolo r_i

Apêndice C

Variações no Mecanismo de Herança no Formalismo

Neste apêndice apresentam-se variações do mecanismo de herança, segundo [PS92], adaptadas para a definição de subdomínios encapsulados; conseqüentemente, gerando variações da propriedade de polimorfismo de domínios encapsulados. Estas variações do mecanismo de herança foram apresentadas informalmente na Seção 3.3.4 e serão aqui descritas segundo o formalismo proposto no Capítulo 4.

Deve-se notar que as relações de comparabilidade entre comportamento (*i.e.*, conjunto de relações) utilizadas a cada variação do mecanismo de herança são cada vez mais rígidas e detalhadas na descrição do comportamento. Sendo que, quanto mais rígidas mais atendem ao princípio da substituição.

C.1 Derivação arbitrária

A definição de um subdomínio pela regra de derivação arbitrária para tipos funcionais permite que se acrescentem ou se eliminem relações no conjunto de relações do subdomínio e/ou que se redefina a implementação das relações herdadas. Assim temos a:

- *Propriedade de polimorfismo por derivação arbitrária de domínio encapsulado.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \cap M'_{e_i} \tag{C.1}$$

Sendo que o conjunto de relações M_{e_i} é caracterizado apenas pelos nomes das relações que descrevem o comportamento do domínio encapsulado D_{e_i} correspondente, *i.e.*:

$$M_{e_i} = \{R_j |\ R_j \text{ nome da j-ésima relação}\} \tag{C.2}$$

Neste caso não se pode garantir a sub-tipicidade entre os domínios encapsulados D_{e_i} e D'_{e_i} associados aos conjuntos de relações M_{e_i} e M'_{e_i} respectivamente. Esta regra apenas indica que D_{e_i} e D'_{e_i} possuem relações com nomes em comum, não garantindo que estas relações possuam a mesma assinatura e/ou a mesma implementação. Sendo assim o princípio da substituição não se aplica.

C.2 Derivação por compatibilidade de nomes

A definição de um subdomínio pela regra sub-tipicidade por compatibilidade de nomes para tipos funcionais requer apenas que o conjunto de relações do subdomínio possua a mesma quantidade de operações, com os mesmos nomes, que seu super-domínio. Portanto, segundo essa regra, para que o domínio encapsulado D'_{e_i} seja considerado subdomínio do domínio encapsulado D_{e_i} basta ambos possuírem conjuntos de relações iguais. Assim temos a:

- *Propriedade de polimorfismo por compatibiliade de nomes de domínio encapsulado*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M'_{e_i} = M_{e_i} \text{ onde } \{M'_{e_i}, M_{e_i}\} \text{ segundo a Equação C.2} \tag{C.3}$$

Neste caso a sub-tipicidade entre os domínios encapsulados D_{e_i} e D'_{e_i} é baseada no fato de que toda $R'_j \in M'_{e_i}$ também pertence a M_{e_i}, entretanto não garante que estas relações tenham a mesma assinatura e/ou a mesma implementação. Sendo assim, da mesma forma que em derivação arbitrária, o princípio da substituição não se aplica; pois pode levar a erros e inconsistências.

C.3 Derivação por compatibilidade de assinatura

A definição de um subdomínio pela regra de derivação por compatibilidade de assinatura para tipos funcionais requer que o conjunto de relações do subdomínio seja formado por relações com assinaturas compatíveis às assinaturas das relações do seu super-domínio. Portanto, temos a:

- *Propriedade de polimorfismo por compatibilidade de assinaturas de domínios encapsulados.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \preceq M'_{e_i} \text{ onde } \{M'_{e_i}, M_{e_i}\} \text{ segundo a Definição 4.34} \quad \text{(C.4)}$$

Portanto:

$$M_{e_i} = \{R_j | R_j : D_{e_i} \rightarrow D_j\} \text{ e}$$

$$M'_{e_i} = \{R'_j | R'_j : D'_{e_i} \rightarrow D'_j\}$$

Sendo que existem duas possibilidades de compatibilidade de assinaturas de relações:

- *Compatibilidade contra-variante.*

$$M_{e_i} \preceq M'_{e_i} \text{ se } D'_{e_i} \subseteq D_{e_i} \text{ e } D'_j \supseteq D_j \quad \text{(C.5)}$$

- *Compatibilidade co-variante.*

$$M_{e_i} \preceq M'_{e_i} \text{ se } D'_{e_i} \subseteq D_{e_i} \text{ e } D'_j \subseteq D_j \quad \text{(C.6)}$$

Nesse caso a sub-tipicidade entre domínio e subdomínios encapsulados é baseada no fato de que toda relação em M_{e_i} tem uma relação igual ou compatível em assinatura em M'_{e_i}, sem segurança de compatibilidade de implementação. O princípio da substituição é válido:

- estáticamente[1] quando a regra contra-variante de compatibilidade de assinatura é aplicada ou
- dinâmicamente[2] quando a regra co-variante de compatibilidade de assinatura é aplicada.

C.4 Derivação monotônica

A definição de um subdomínio pela regra de derivação monotônica para tipos funcionais permite que o conjunto de relações do subdomínio seja estendido com novas relações. O conjunto de relações é definido pela assinatura das relações. Não é feita nenhuma restrição quanto à possibilidade de redefinir a implementação das relações herdadas nos subdomínios. Portanto, temos a:

[1] Na compilação do programa.
[2] Durante a execução do programa.

- *Propriedade de polimorfismo por derivação monotônica de domínio encapsulado.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \subseteq M'_{e_i} \text{ onde } \{M'_{e_i}, M_{e_i}\} \text{ segundo a Definição 4.34} \quad \text{(C.7)}$$

Sendo que:

$$M_{e_i} \subseteq M'_{e_i}$$

$$\text{para } M_{e_i} = \{R_j | R_j : D_{e_i} \rightarrow D_j\} \text{ e}$$

$$M'_{e_i} = \{R'_k | R'_k : D'_{e_i} \rightarrow D'_k\}$$

$$\text{se } card(M_{e_i}) \leq card(M'_{e_i}) \text{ e } \forall R_j \exists_1 R'_k \text{ tal que } R_j = R'_k$$

Neste caso a sub-tipicidade entre os domínios encapsulados D_{e_i} e D'_{e_i} é baseada no fato de que toda relação pertencente a M_{e_i} também pertence a M'_{e_i}, garantindo a igualdade de interface sem verificar compatibilidade em implementação. Entretanto, o subdomínio D'_{e_i} pode possuir comportamentos extras, não definidos para o seu super-domínio D_{e_i}. Assim, o princípio da substituição apenas garante compatibilidade estática, podendo haver inconsistência de comportamento devido a flexibilidade de redefinição de implementação nas relações herdadas.

C.5 Derivação por compatibilidade comportamental

A definição de um subdomínio pela regra de derivação por compatibilidade comportamental para tipos funcionais considera a assinatura e o conjunto solução de cada relação. Portanto, temos a:

- *Propriedade de polimorfismo por compatibilidade comportamental de domínios encapsulados.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \subseteq M'_{e_i} \quad \text{(C.8)}$$

Onde

$$M_{e_i} \subseteq M_{e_i}$$

$$\text{para } M_{e_i} = \{R_j | R_j : D_{e_i} \rightarrow D_j, R^*_j\} \text{ e}$$

$$M'_{e_i} = \{R'_k | R'_k : D'_{e_i} \rightarrow D'_k, R'^*_k\}$$

$$\text{se } card(M_{e_i}) \leq card(M'_{e_i}), \forall R_j \exists_1 R'_k \text{ tal que } R_j = R'_k \text{ e } R^*_j = R'^*_k$$

Neste caso a sub-tipicidade entre os domínios encapsulados D'_{e_i} e D_{e_i} é baseada no fato de que toda relação em M_{e_i} possui uma relação correspondente em M'_{e_i}, compatível em interface e conjunto solução; portanto, é possível modificar a implementação de um relação herdada mantendo o conjunto solução da relação o mesmo. O conjunto de relações M'_{e_i} pode conter relações extras, que não fazem parte de M_{e_i}. Assim, o princípio da substituição é totalmente válido, *i.e.*, qualquer elemento do domínio D'_{e_i} pode substituir um elemento do seu super-domínio D_{e_i} e qualquer relação de M_{e_i} pode substituir uma relação correspondente de M'_{e_i}.

C.6 Derivação estritamente monotônica

Na derivação estritamente monotônica de subdomínios encapsulados a definição do conjunto de relações inclui a definição da interface e a implementação de cada relação no domínio, sendo que a única alteração possível no subdomínio é o acréscimo de comportamento; portanto, temos a:

- *Propriedade de polimorfismo estritamente monotônico de domínio encapsulado.*

$$D'_{e_i} \preceq D_{e_i} \text{ se } M_{e_i} \subseteq M'_{e_i} \tag{C.9}$$

Onde

$$M_{e_i} \subseteq M'_{e_i}$$

$$\text{para } M_{e_i} = \{R_j | R_j : D_{e_i} \rightarrow D_j, Imp(R_j)\}\} \text{ e}$$

$$\text{para } M'_{e_i} = \{R'_k | R'_k : D'_{e_i} \rightarrow D'_k, Imp(R'_k)\}$$

$$\text{se } card(M_{e_i}) \leq card(M'_{e_i}),\ \forall R_j \exists_1 R'_k \text{ tal que } R_j = R'_k \text{ e } Imp(R_j) = Imp(R'_k).$$

Neste caso a sub-tipicidade entre os domínios encapsulados D'_{e_i} e D_{e_i} é baseada no fato de que toda relação em M_{e_i} também pertence a M'_{e_i} não sendo possível modificações e que M'_{e_i} contém relações extras, que não fazem parte de M_{e_i}. Conseqüentemente o princípio da substituição é totalmente válido.

Apêndice D

Formalização do Primeiro Estudo de Caso

Os domínios apresentados no primeiro estudo de caso do Apêndice A são aqui detalhados em forma tabular onde cada tabela contém as seguintes informações:

Subdom. : Subdomínio segundo Equações 4.32, 4.33, 4.35, 4.36 ou 4.41.

Def. : Domínio segundo Definições 4.1, 4.33, 4.40, 4.41 ou 4.43.

Comp. : Composição do domínio segundo Definição 4.13.

Card. : Cardinalidade do domínio.

Comport. : Comportamento segundo Definições 4.7, 4.16, 4.22, 4.28 ou 4.34.

Imp. : Implementação do comportamento segundo Equações 4.3, 4.9, 4.14, 4.19 ou 4.25.

Ext. : Extensão do domínio (ou conjunto de instâncias) segundo Definições 4.48, 4.49, 4.50 ou 4.51

Est. : Estados das instâncias segundo Definição 4.54.

Ids : Conjunto de identificadores associado ao domínio segundo Definições 4.44, 4.45 ou 4.46.

Tabela D.1: Domínios básicos D_{b_2} e D_{b_4}.

DOMÍNIO D_{b_2}	
Nível Intencional	
Subdom.	$D_{b_2} \preceq D_{b_1} \preceq D_{b_{super}}$
Def.	$D_{b_2} = \{1, 2, 3, 4\}$
Card.	$Card(D_{b_2}) = 4$
Comport.	$+ : D_{b_2} \times D_{b_2} \rightarrow D_{b_1}$ $- : D_{b_2} \times D_{b_2} \rightarrow D_{b_1}$ $/ : D_{b_2} \times D_{b_2} \rightarrow D_{b_1}$ $=: D_{b_2} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{b_2} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{b_2}) = (D_{b_2}, I_{M_{b_2}})$
Nível Concreto	
Ext.	$ED_{b_2} = \{(1,1), (2,2), (3,3), (4,4)\}$
Est.=Ids	$ID_{b_2} = D_{b_2}$
DOMÍNIO D_{b_4}	
Nível Intencional	
Subdom.	$D_{b_4} \preceq D_{b_3} \preceq D_{b_{super}}$
Def.	$D_{b_4} = \{A, B, C, D, E\}$
Card.	$Card(D_{b_4}) = 5$
Comport.	$=: D_{b_4} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{b_4} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{b_4}) = (D_{b_4}, I_{M_{b_4}})$
Nível Concreto	
Ext.	$ED_{b_4} = \{(A,A), (B,B), (C,C), (D,D), (E,E)\}$
Est.=Ids	$ID_{b_4} = D_{b_4}$

Tabela D.2: Domínios básicos D_{b_5} e D_{b_6}.

DOMÍNIO D_{b_5}	
Nível Intencional	
Subdom.	$D_{b_5} \preceq D_{b_4}$
Def.	$D_{b_5} = \{A, B, C\}$
Card.	$Card(D_{b_5}) = 3$
Comport.	$=: D_{b_5} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{b_5} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{b_5}) = (D_{b_5}, I_{M_{b_5}})$
Nível Concreto	
Ext.	$ED_{b_5} = \{(A, A), (B, B), (C, C)\}$
Est.=Ids	$ID_{b_5} = D_{b_5}$

DOMÍNIO D_{b_6}	
Nível Intencional	
Subdom.	$D_{b_6} \preceq D_{super}$
Def.	$D_{b_6} = \{t, f\}$
Card.	$Card(D_{b_6}) = 2$
Comport.	$and : D_{b_6} \times D_{b_6} \rightarrow D_{b_6}$ $or : D_{b_6} \times D_{b_6} \rightarrow D_{b_6}$ $=: D_{b_6} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{b_6} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{b_6}) = (D_{b_6}, I_{M_{b_6}})$
Nível Concreto	
Ext.	$ED_{b_6} = \{(t, t), (f, f)\}$
Est.=Ids	$ID_{b_6} = D_{b_6}$

Tabela D.3: Domínios compostos D_{ar_1} e D_{ar_2}.

DOMÍNIO D_{ar_1}	
Nível Intencional	
Subdom.	$D_{ar_1} \preceq D_{ar_{super}}$
Def.	$D_{ar_1} = \bigcup_{j=1}^{2} \times_j D_{b_4}$
Comp.	$Comp(D_{ar_1}) = D_{b_4}$
Card.	$Card(D_{ar_1}) = Card(D_{b_4})^1 + Card(D_{b_4})^2 = 30$
Comport.	$primeiro : D_{ar_1} \rightarrow D_{b_4}$ $segundo : D_{ar_1} \rightarrow D_{b_4}$ $=: D_{ar_1} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ar_1} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ar_1}) = (D_{b_4}, I_{M_{ar_1}})$
Nível Concreto	
Ext.	$ED_{ar_1} = \{([A],[A]),([B],[B]),\ldots([E,E],[E,E])\}$
Est.=Ids	$ID_{ar_1} = D_{ar_1}$
DOMÍNIO D_{ar_2}	
Nível Intencional	
Subdom.	$D_{ar_2} \preceq D_{ar_1}$
Def.	$D_{ar_2} = \bigcup_{j=1}^{2} \times_j D_{b_5}$
Comp.	$Comp(D_{ar_2}) = D_{b_5}$
Card.	$Card(D_{ar_2}) = Card(D_{b_5})^1 + Card(D_{b_5})^2 = 12$
Comport.	$primeiro : D_{ar_2} \rightarrow D_{b_5}$ $segundo : D_{ar_2} \rightarrow D_{b_5}$ $=: D_{ar_2} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ar_2} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ar_2}) = (D_{ar_2}, I_{M_{ar_2}})$
Nível Concreto	
Ext.	$ED_{ar_2} = \{([A],[A]),([B],[B]),\ldots,([B,C],[B,C]),([C,C],[C,C])\}$
Est.=Ids	$ID_{ar_2} = D_{ar_2}$

Tabela D.4: Domínios encapsulados D_{e_1} e D_{e_2}.

DOMÍNIO D_{e_1}	
Nível Intencional	
Subdom.	$D_{e_1} \preceq D_{e_{super}}$
Def.	$D_{e_1} = \{D_{ar_1}\}$
Comp.	$Comp(D_{e_1}) = \{Comp(D_{ar_1})\}$
Card.	$Card(D_{e_1}) = Card(D_{ar_1})$
Comport. herdado	$=: D_{e_1} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_1} \times D_i \rightarrow D_{b_6}$
Comport. acrescentado	$lista : D_{e_1} \rightarrow D_{ar_1}$ $insere : D_{e_1} \times D_{b_4} \rightarrow D_{e_1}$ $elimina : D_{e_1} \times D_{b_4} \rightarrow D_{e_1}$ $altera : D_{e_1} \times D_{b_4} \times D_{b_4} \rightarrow D_{e_1}$
Imp.	$Imp(M_{e_1}) = \{Imp(M_{e_1})_1\} = \{(D_{ar_1}, I_{M_{e_{1,1}}})\}$
Nível Concreto	
Ext.	$ED_{e_{1,t1}} = \{(\#1, [A]), (\#2, [D])\}$
Est.	$estado(ED_{e_{1,t1}}) = \{[A], [D]\} \subset ID_{ar_1}$
Ids	$ID_{e_1} = \{\#1, \#2\}$
DOMÍNIO D_{e_2}	
Nível Intencional	
Subdom.	$D_{e_2} \preceq D_{e_1}$
Def.	$D_{e_2} = \{D_{ar_2}\}$
Comp.	$Comp(D_{e_2}) = \{Comp(D_{ar_2})\}$
Card.	$Card(D_{e_2}) = Card(D_{ar_2})$
Comport. herdado	$=: D_{e_2} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_2} \times D_i \rightarrow D_{b_6}$ $lista : D_{e_2} \rightarrow D_{ar_2}$ $insere : D_{e_2} \times D_{b_5} \rightarrow D_{e_2}$ $elimina : D_{e_2} \times D_{b_5} \rightarrow D_{e_2}$ $altera : D_{e_2} \times D_{b_5} \times D_{b_5} \rightarrow D_{e_2}$
Imp.	$Imp(M_{e_2}) = \{Imp(M_{e_2})_1\} = \{(D_{ar_2}, I_{M_{e_{2,1}}})\}$
Nível Concreto	
Ext.	$ED_{e_{2,t1}} = \{(\#3, [A]), (\#4, [B])\}$
Est.	$estado(ED_{e_{2,t1}}) = \{[A], [B]\} \subset ID_{ar_2}$
Ids	$ID_{e_2} = \{\#3, \#4\}$

Tabela D.5: Domínios compostos D_{ag_1} e D_{ag_2}.

DOMÍNIO D_{ag_1}	
Nível Intencional	
Subdom.	$D_{ag_1} \preceq D_{ag_{super}}$
Def.	$D_{ag_1} = r_1 : D_{b_2} \times \ldots$
Comp.	$Comp(D_{ag_1}) = \{D_{b_2}\}$
Card.	$Card(D_{ag_1}) = Card(D_{b_2}) = 4$
Comport.	$r_1 : D_{ag_1} \rightarrow D_{b_2}$ $=: D_{ag_1} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ag_1} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ag_1}) = (D_{ag_1}, I_{M_{ag_1}})$
Nível Concreto	
Ext.	$ED_{ag_1} = \{([r_1 : 1, \ldots], [r_1 : 1, \ldots]), \ldots, ([r_1 : 4, \ldots], [r_1 : 4, \ldots])\}$
Est. = Ids	$ID_{ag_1} = D_{ag_1}$
DOMÍNIO D_{ag_2}	
Nível Intencional	
Subdom.	$D_{ag_2} \preceq D_{ag_1}$
Def.	$D_{ag_2} = r_1 : D_{b_2} \times r_2 : D_{b_2} \times r_3 : 2^{D_{b_2}}$
Comp.	$Comp(D_{ag_2}) = \{D_{b_2}\}$
Card.	$Card(D_{ag_1}) = Card(D_{b_2}) * Card(D_{b_2}) * 2^{Card(D_{b_2})} = 256$
Comport.	$r_1 : D_{ag_2} \rightarrow D_{b_2}$ $r_2 : D_{ag_2} \rightarrow D_{b_2}$ $r_3 : D_{ag_2} \rightarrow 2^{D_{b_2}}$ $=: D_{ag_2} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ag_2} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ag_2}) = (D_{ag_2}, I_{M_{ag_2}})$
Nível Concreto	
Ext.	$ED_{ag_2} = \{([r_1 : 1, r_2 : 1, r_3 : \{\}, \ldots], [r_1 : 1, r_2 : 1, r_3 : \{\}, \ldots]), \ldots,$ $([r_1 : 4, r_2 : 4, r_3 : \{1, 2, 3, 4\}, \ldots], [r_1 : 4, r_2 : 4, r_3 : \{1, 2, 3, 4\}, \ldots])\}$
Est. = Ids	$ID_{ag_2} = D_{ag_2}$

Tabela D.6: Domínio composto D_{ag_3}.

DOMÍNIO D_{ag_3}	
Nível Intencional	
Subdom.	$D_{ag_3} \preceq D_{ag_1}$
Def.	$D_{ag_3} = r_1 : D_{b_2} \times r_4 : D_{e_1} \times \ldots$
Comp.	$Comp(D_{ag_3}) = \{D_{b_2}, D_{e_1}\}$
Card.	$Card(D_{ag_3}) = Card(D_{b_2}) * Card(D_{e_1}) =$ $Card(D_{b_2}) * Card(D_{ar_1}) = 4 * 30 = 120$
Comport.	$r_1 : D_{ag_3} \rightarrow D_{b_2}$ $r_4 : D_{ag_3} \rightarrow D_{e_1}$ $=: D_{ag_3} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ag_3} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ag_3}) = (D_{ag_3}, I_{M_{ag_3}})$
Nível Concreto	
Ext.	$ED_{ag_{3,t1}} = \{([r_1:1, r_4:\#1, \ldots], [r_1:1, r_4:\#1, \ldots]),$ $([r_1:2, r_4:\#1, \ldots], [r_1:2, r_4:\#1, \ldots]),$ $([r_1:3, r_4:\#1, \ldots], [r_1:3, r_4:\#1, \ldots]),$ $([r_1:4, r_4:\#1, \ldots], [r_1:4, r_4:\#1, \ldots]),$ $([r_1:1, r_4:\#2, \ldots], [r_1:1, r_4:\#2, \ldots]),$ $([r_1:2, r_4:\#2, \ldots], [r_1:2, r_4:\#2, \ldots]),$ $([r_1:3, r_4:\#2, \ldots], [r_1:3, r_4:\#2, \ldots]),$ $([r_1:4, r_4:\#2, \ldots], [r_1:4, r_4:\#2, \ldots])\}$
Card.	$Card(ED_{ag_{3,t1}}) = Card(ID_{b_2}) * Card(ID_{e_{1,t1}}) = 4 * 2 = 8$
Est. = Ids	$ID_{ag_{3,t1}} = \{[r_1:1, r_4:\#1, \ldots], [r_1:2, r_4:\#1, \ldots],$ $[r_1:3, r_4:\#1, \ldots], [r_1:4, r_4:\#1, \ldots],$ $[r_1:1, r_4:\#2, \ldots], [r_1:2, r_4:\#2, \ldots],$ $[r_1:3, r_4:\#2, \ldots], [r_1:4, r_4:\#2, \ldots]\}$

Tabela D.7: Domínio composto D_{ag_4}.

DOMÍNIO D_{ag_4}	
Nível Intencional	
Subdom.	$D_{ag_4} \preceq D_{ag_3}$
Def.	$D_{ag_4} = r_1 : D_{b_2} \times r_4 : D_{e_2} \times \ldots$
Comp.	$Comp(D_{ag_4}) = \{D_{b_2}, D_{e_2}\}$
Card.	$Card(D_{ag_4}) = Card(D_{b_2}) * Card(D_{e_2}) =$ $Card(D_{b_2}) * Card(D_{ar_2}) = 4 * 12 = 48$
Comport.	$r_1 : D_{ag_4} \rightarrow D_{b_2}$ $r_4 : D_{ag_4} \rightarrow D_{e_2}$ $=: D_{ag_4} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{ag_4} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{ag_4}) = (D_{ag_4}, I_{M_{ag_4}})$
Nível Concreto	
Ext.	$ED_{ag_{4,t1}} = \{([r_1 : 1, r_4 : \#3, \ldots], [r_1 : 1, r_4 : \#3, \ldots]),$ $([r_1 : 2, r_4 : \#3, \ldots], [r_1 : 2, r_4 : \#3, \ldots]),$ $([r_1 : 3, r_4 : \#3, \ldots], [r_1 : 3, r_4 : \#3, \ldots]),$ $([r_1 : 4, r_4 : \#3, \ldots], [r_1 : 4, r_4 : \#3, \ldots]),$ $([r_1 : 1, r_4 : \#4, \ldots], [r_1 : 1, r_4 : \#4, \ldots]),$ $([r_1 : 2, r_4 : \#4, \ldots], [r_1 : 2, r_4 : \#4, \ldots]),$ $([r_1 : 3, r_4 : \#4, \ldots], [r_1 : 3, r_4 : \#4, \ldots]),$ $([r_1 : 4, r_4 : \#4, \ldots], [r_1 : 4, r_4 : \#4, \ldots])\}$
Card.	$Card(ED_{ag_{4,t1}}) = Card(ID_{b_2}) * Card(ID_{e_{2,t1}}) = 4 * 2 = 8$
Est. = Ids	$ID_{ag_{4,t1}} = \{[r_1 : 1, r_4 : \#3, \ldots], [r_1 : 2, r_4 : \#3, \ldots],$ $[r_1 : 3, r_4 : \#3, \ldots], [r_1 : 4, r_4 : \#3, \ldots],$ $[r_1 : 1, r_4 : \#4, \ldots], [r_1 : 2, r_4 : \#4, \ldots],$ $[r_1 : 3, r_4 : \#4, \ldots], [r_1 : 4, r_4 : \#4, \ldots]\}$

Tabela D.8: Domínios encapsulados D_{e_3} e D_{e_4}.

DOMÍNIO D_{e_3}	
Nível Intencional	
Subdom.	$D_{e_3} \preceq D_{e_{super}}$
Def.	$D_{e_3} = \{D_{ag_1}\}$
Comp.	$Comp(D_{e_3}) = \{Comp(D_{ag_1})\}$
Comport.	$=: D_{e_3} \times D_i \rightarrow D_{b_6}$
herdado	$\preceq: D_{e_3} \times D_i \rightarrow D_{b_6}$
acrescentado	$numero : D_{e_3} \rightarrow D_{b_2}$
Imp.	$Imp(M_{e_3}) = \{Imp(M_{e_3})_1\} = \{(D_{ag_1}, I_{M_{e_3,1}})\}$
DOMÍNIO D_{e_4}	
Nível Intencional	
Subdom.	$D_{e_4} \preceq D_{e_3}$
Def.	$D_{e_4} = \{D_{ag_2}\}$
Comp.	$Comp(D_{e_4}) = \{Comp(D_{ag_2})\}$
Card.	$Card(D_{e_4}) = Card(D_{ag_2})$
Comport.	$numero : D_{e_4} \rightarrow D_{b_2}$
herdado	$=: D_{e_4} \times D_i \rightarrow D_{b_6}$
	$\preceq: D_{e_4} \times D_i \rightarrow D_{b_6}$
acrescentado	$filhos : D_{e_4} \rightarrow \{D_{b_2}\}$
	$alterafilhos : D_{e_4} \times D_{b_2} \times D_{b_2} \rightarrow D_{e_4}$
	$salario : D_{e_4} \rightarrow D_{b_2}$
	$aumento : D_{e_4} \rightarrow D_{e_4}$
Imp.	$Imp(M_{e_4}) = \{Imp(M_{e_4})_1\} = \{(D_{ag_2}, I_{M_{e_4,1}})\}$
Nível Concreto	
Ext.	$ED_{e_{4,t1}} = \{(\#5, [r_1 : 1, r_2 : 1, r_3 : \{1\}, \ldots]), (\#6, [r_1 : 2, r_2 : 3, r_3 : \{1\}, \ldots])\}$
Est.	$estado(ED_{e_{4,t1}}) = \{[r_1 : 1, r_2 : 1, r_3 : \{1\}, \ldots], [r_1 : 2, r_2 : 3, r_3 : \{1\}, \ldots]$
Ids	$ID_{e_{4,t1}} = \{\#5, \#6\}$

Tabela D.9: Domínio encapsulado D_{e_5}.

DOMÍNIO D_{e_5}	
Nível Intencional	
Subdom.	$D_{e_5} \preceq D_{e_3}$
Def.	$D_{e_5} = \{D_{ag_3}\}$
Comp.	$Comp(D_{e_5}) = \{Comp(D_{ag_3})\}$
Card.	$Card(D_{e_5}) = Card(D_{ag_3})$
Comport. herdado	$numero : D_{e_5} \rightarrow D_{b_2}$ $=: D_{e_5} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_5} \times D_i \rightarrow D_{b_6}$
acrescentado	$notas : D_{e_5} \rightarrow D_{e_1}$ $alteranota : D_{e_5} \times D_{b_2} \times D_{b_4} \rightarrow D_{e_5}$ $media : D_{e_5} \rightarrow D_{b_4}$
Imp.	$Imp(M_{e_5}) = \{Imp(M_{e_5})_1, Imp(M_{e_5})_2\} =$ $Imp(M_{e_5})_1 = (D_{ag_3}, I_{M_{e_5,1}})$ $Imp(M_{e_5})_2 = (D_{ag_3}, I_{M_{e_5,2}})$
Nível Concreto	
Ext.	$ED_{e_5,t1} = \{(\#7, [r_1 : 1, r_4 : \#1, \ldots]),$ $(\#8, [r_1 : 2, r_4 : \#1, \ldots]),$ $(\#9, [r_1 : 3, r_4 : \#2, \ldots]),$ $(\#10, [r_1 : 4, r_4 : \#2, \ldots]), \}$
Est.	$estado(ED_{e_5,t1}) = \{[r_1 : 1, r_4 : \#1, \ldots],$ $[r_1 : 2, r_4 : \#1, \ldots], [r_1 : 3, r_4 : \#2, \ldots], [r_1 : 4, r_4 : \#2, \ldots]\}$
Ids	$ID_{e_5,t1} = \{\#7, \#8, \#9, \#10\}$

Tabela D.10: Domínio encapsulado D_{e_6}.

DOMÍNIO D_{e_6}	
Nível Intencional	
Subdom.	$D_{e_6} \preceq D_{e_5}$
Def.	$D_{e_6} = \{D_{ag_4}\}$
Comp.	$Comp(D_{e_6}) = \{Comp(D_{ag_4})\}$
Card.	$Card(D_{e_6}) = Card(D_{ag_4})$
Comport. herdado	$numero : D_{e_6} \rightarrow D_{b_2}$ $=: D_{e_6} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_6} \times D_i \rightarrow D_{b_6}$ $notas : D_{e_6} \rightarrow D_{e_1}$ $alteranota : D_{e_6} \times D_{b_2} \times D_{b_5} \rightarrow D_{e_6}$ $media : D_{e_6} \rightarrow D_{b_5}$
Imp.	$Imp(M_{e_6}) = \{Imp(M_{e_6})_1\} = \{(D_{ag_4}, I_{M_{e_6,1}})\}$
Nível Concreto	
Ext.	$ED_{e_{6,t1}} = \{(\#11, [r_1 : 1, r_4 : \#3, \ldots]),$ $(\#12, [r_1 : 2, r_4 : \#3, \ldots]),$ $(\#13, [r_1 : 3, r_4 : \#3, \ldots]),$ $(\#14, [r_1 : 4, r_4 : \#4, \ldots]), \}$
Est.	$estado(ED_{e_{6,t1}}) = \{[r_1 : 1, r_4 : \#3, \ldots],$ $[r_1 : 2, r_4 : \#3, \ldots], [r_1 : 3, r_4 : \#3, \ldots], [r_1 : 4, r_4 : \#4, \ldots]\}$
Ids	$ID_{e_{6_{t1}}} = \{\#11, \#12, \#13, \#14\}$

Tabela D.11: Domínios compostos D_{po_1} e D_{po_2}.

DOMÍNIO D_{po_1}	
Nível Intencional	
Subdom.	$D_{po_1} \preceq D_{po_{super}}$
Def.	$D_{po_1} = 2^{D_{e_4}}$
Comp.	$Comp(D_{po_1}) = \{D_{e_4}\}$
Card.	$Card(D_{po_1}) = 2^{(Card(D_{e_4}))} = 2^{(Card(D_{ag_2}))} = 2^{256}$
Comport.	$\in$: $D_{po_1} \times D_{e_4} \rightarrow D_{b_6}$ $=$: $D_{po_1} \times D_i \rightarrow D_{b_6}$ $\preceq$: $D_{po_1} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{po_1}) = (D_{po_1}, I_{M_{po_{1,1}}})$
Nível Concreto	
Ext.	$ED_{po_{1,t1}} = \{(\{\}, \{\}), (\{\{\#5\}, \{\#5\}\}(\{\#6\}, \{\#6\}\}(\{\#5, \#6\}, \{\#5, \#6\}\}\}$
Card.	$Card(ED_{po_{1,t1}}) = 2^{(Card(ID_{e_{4,t1}}))} = 2^2 = 4$
Est. = Ids	$ID_{po_{1,t1}} = \{\{\}, \{\#5\}, \{\#6\}, \{\#5, \#6\}\}$

DOMÍNIO D_{po_2}	
Nível Intencional	
Subdom.	$D_{po_2} \preceq D_{po_{super}}$
Def.	$D_{po_2} = 2^{D_{e_5}}$
Comp.	$Comp(D_{po_2}) = D_{e_5}$
Card.	$Card(D_{po_2}) = 2^{(Card(D_{e_5}))} = 2^{(Card(D_{ag_3}))} = 2^{120}$
Comport.	$\in$: $D_{po_2} \times D_{e_5} \rightarrow D_{b_6}$ $=$: $D_{po_2} \times D_i \rightarrow D_{b_6}$ $\preceq$: $D_{po_2} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{po_2}) = (D_{po_2}, I_{M_{po_{2,1}}})$
Nível Concreto	
Ext.	$ED_{po_{2,t1}} = (\{\}, \{\}), (\{\#7\}, \{\#7\}), (\{\#8\}, \{\#8\}), (\{\#9\}, \{\#9\}), (\{\#10\}, \{\#10\}),$ $(\{\#7, \#8\}, \{\#7, \#8\}), (\{\#7, \#9\}, \{\#7, \#9\}), (\{\#7, \#10\}, \{\#7, \#10\}),$ $(\{\#8, \#9\}, \{\#8, \#9\}), (\{\#8, \#10\}, \{\#8, \#10\}), (\{\#9, \#10\}, \{\#9, \#10\}),$ $(\{\#7, \#8, \#9\}, \{\#7, \#8, \#9\}), (\{\#7, \#8, \#10\}, \{\#7, \#8, \#10\}),$ $(\{\#7, \#9, \#10\}, \{\#7, \#9, \#10\}), (\{\#8, \#9, \#10\}, \{\#8, \#9, \#10\}),$ $(\{\#7, \#8, \#9, \#10\}, \{\#7, \#8, \#9, \#10\})\}$
Card.	$Card(ED_{po_{2,t1}}) = 2^{(Card(ID_{e_{5,t1}}))} = 2^4 = 16$
Est. = Ids	$ID_{po_{2,t1}} = \{\{\}, \{\#7\}, \{\#8\}, \ldots, \{\#7, \#8, \#9, \#10\}\}$

Tabela D.12: Domínio composto D_{po_3}.

DOMÍNIO D_{po_3}	
Nível Intencional	
Subdom.	$D_{po_3} \preceq D_{po_2}$
Def.	$D_{po_3} = 2^{D_{e_6}}$
Comp.	$Comp(D_{po_3}) = D_{e_6}$
Card.	$Card(D_{po_3}) = 2^{(Card(D_{e_6}))} = 2^{(Card(D_{ag_4}))} = 2^{48}$
Comport.	$\in$: $D_{po_3} \times D_{e_6} \rightarrow D_{b_6}$ $=$: $D_{po_3} \times D_i \rightarrow D_{b_6}$ $\preceq$: $D_{po_3} \times D_i \rightarrow D_{b_6}$
Imp.	$Imp(M_{po_3}) = (D_{po_3}, I_{M_{po_{3,1}}})$
Nível Concreto	
Ext.	$ED_{po_{3,t1}} = (\{\}, \{\}), (\{\#11\}, \{\#11\}), (\{\#12\}, \{\#12\}), (\{\#13\}, \{\#13\}), (\{\#14\}, \{\#14\}),$ $(\{\#11, \#12\}, \{\#11, \#12\}), (\{\#11, \#13\}, \{\#11, \#13\}), (\{\#11, \#14\}, \{\#11, \#14\}),$ $(\{\#12, \#13\}, \{\#12, \#13\}), (\{\#12, \#14\}, \{\#12, \#14\}), (\{\#13, \#14\}, \{\#13, \#14\}),$ $(\{\#11, \#12, \#13\}, \{\#11, \#12, \#13\}), (\{\#11, \#12, \#14\}, \{\#11, \#12, \#14\}),$ $(\{\#11, \#13, \#14\}, \{\#11, \#13, \#14\}), (\{\#12, \#13, \#14\}, \{\#12, \#13, \#14\}),$ $(\{\#11, \#12, \#13, \#14\}, \{\#11, \#12, \#13, \#14\})\}$
Card.	$Card(ED_{po_{3,t1}}) = 2^{(Card(ID_{e_{6,t1}})} = 2^4 = 16$
Est. = Ids	$ID_{po_{3,t1}} = \{\{\}, \{\#11\}, \{\#12\}, \ldots, \{\#11, \#12, \#13, \#14\}\}$

Tabela D.13: Domínios encapsulados D_{e_7} e D_{e_8}.

DOMÍNIO D_{e_7}	
Nível Intencional	
Subdom.	$D_{e_7} \preceq D_{e_{super}}$
Def.	$D_{e_7} = \{D_{po_{super}}\}$
Comp.	$Comp(D_{e_7}) = \{Comp(D_{po_{super}})\}$
Comport.	$=: D_{e_7} \times D_i \rightarrow D_{b_6}$
herdado	$\preceq: D_{e_7} \times D_i \rightarrow D_{b_6}$
acrescentado	$lista : D_{e_7} \rightarrow D_{po_{super}}$
	$insere : D_{e_7} \times D_{e_{super}} \rightarrow D_{e_7}$
	$elimina : D_{e_7} \times D_{e_{super}} \rightarrow D_{e_7}$
	$altera : D_{e_7} \times D_{e_{super}} \times D_{e_{super}} \rightarrow D_{e_7}$
Imp.	$Imp(M_{e_7}) = \{Imp(M_{e_7})_1\} = \{(D_{po_{super}}, I_{M_{e_{7_1}}})\}$
DOMÍNIO D_{e_8}	
Nível Intencional	
Subdom.	$D_{e_8} \preceq D_{e_{super}}$
Def.	$D_{e_8} = \{D_{po_1}\}$
Comp.	$Comp(D_{e_8}) = \{Comp(D_{po_1})\}$
Card.	$Card(D_{e_8}) = Card(D_{po_1})$
Comport.	$=: D_{e_8} \times D_i \rightarrow D_{b_6}$
herdado	$\preceq: D_{e_8} \times D_i \rightarrow D_{b_6}$
	$lista : D_{e_8} \rightarrow D_{po_1}$
	$insere : D_{e_8} \times D_{e_4} \rightarrow D_{e_8}$
	$elimina : D_{e_8} \times D_{e_4} \rightarrow D_{e_8}$
	$altera : D_{e_8} \times D_{e_4} \times D_{e_4} \rightarrow D_{e_8}$
acrescentado	$media : D_{e_8} \rightarrow D_{b_2}$
Imp.	$Imp(M_{e_8}) = \{Imp(M_{e_8})_1\} = \{(D_{po_1}, I_{M_{e_{8_1}}})\}$
Nível Concreto	
Ext.	$ED_{e_{8,t1-2}} = \{\}$
	$ED_{e_{8,t1-1}} = \{(\#15, \{\#5\})\}$
	$ED_{e_{8,t1}} = \{(\#15, \{\#5, \#6\})\}$
Est.	$estado(ED_{e_{8,t1}}) = \{\{\#5, \#6\}\}$
Ids	$ID_{e_{8,t1}} = \{\#15\}$

Tabela D.14: Domínios encapsulados D_{e_9} e $D_{e_{10}}$.

DOMÍNIO D_{e_9}	
Nível Intencional	
Subdom.	$D_{e_9} \preceq D_{e_7}$
Def.	$D_{e_9} = \{D_{po_2}\}$
Comp.	$Comp(D_{e_9}) = \{Comp(D_{po_2})\}$
Card.	$Card(D_{e_9}) = Card(D_{po_2})$
Comport. herdado	$=: D_{e_9} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_9} \times D_i \rightarrow D_{b_6}$ $lista : D_{e_9} \rightarrow D_{po_2}$ $insere : D_{e_9} \times D_{e_5} \rightarrow D_{e_9}$ $elimina : D_{e_9} \times D_{e_5} \rightarrow D_{e_9}$ $altera : D_{e_9} \times D_{e_5} \times D_{e_5} \rightarrow D_{e_9}$
acrescentado	$media : D_{e_9} \rightarrow D_{b_4}$
Imp.	$Imp(M_{e_9}) = \{Imp(M_{e_9})_1\} = \{(D_{po_2}, I_{M_{e_9,1}})\}$
Nível Concreto	
Ext.	$ED_{e_9,t1} = \{(\#16, \{\#7, \#8, \#9, \#10, \#14\})\}$
Est.	$estado(ED_{e_9,t1}) = \{\{\#7, \#8, \#9, \#10, \#14\}\}$
Ids	$ID_{e_9,t1} = \{\#16\}$

DOMÍNIO $D_{e_{10}}$	
Nível Intencional	
Subdom.	$D_{e_{10}} \preceq D_{e_9}$
Def.	$D_{e_{10}} = \{D_{po_3}\}$
Comp.	$Comp(D_{e_{10}}) = \{Comp(D_{po_3})\}$
Card.	$Card(D_{e_{10}}) = Card(D_{po_3})$
Comport. herdado	$=: D_{e_{10}} \times D_i \rightarrow D_{b_6}$ $\preceq: D_{e_{10}} \times D_i \rightarrow D_{b_6}$ $lista : D_{e_{10}} \rightarrow D_{po_3}$ $insere : D_{e_{10}} \times D_{e_6} \rightarrow D_{e_{10}}$ $elimina : D_{e_{10}} \times D_{e_6} \rightarrow D_{e_{10}}$ $altera : D_{e_{10}} \times D_{e_6} \times D_{e_6} \rightarrow D_{e_{10}}$
	$media : D_{e_{10}} \rightarrow D_{b_5}$
Imp.	$Imp(M_{e_{10}}) = \{Imp(M_{e_{10}})_1\} = \{(D_{po_3}, I_{M_{e_{10},1}})\}$
Nível Concreto	
Ext.	$ED_{e_{10},t1} = \{(\#17, \{\#11, \#12, \#13\})\}$
Est.	$estado(ED_{e_{10},t1}) = \{\{\#11, \#12, \#13\}\}$
Ids	$ID_{e_{10},t1} = \{\#17\}$