Implementação de um Sistema Temporal em um Banco de Dados Orientado a Objetos

Ângelo Roncalli Alencar Brayner

Implementação de um Sistema Temporal em um Banco de Dados Orientado a Objetos

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Ângelo Roncalli Alencar Brayner 739 e aprovada pela Comissão Julgadora.

Campinas, 15 de abril de 1994.

Claudia Bauzer Medeiros
Orientadora

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Implementação de um Sistema Temporal em um Banco de Dados Orientado a Objetos[1]

Ângelo Roncalli Alencar Brayner[2]

Departamento de Ciência da Computação
IMECC – UNICAMP

Banca Examinadora:

- Ricardo de Oliveira Anido (Suplente)[3]
- Claudia Bauzer Medeiros(Orientadora)[3]
- Leo Pini Magalhães[4]
- Maria Beatriz Felgar de Toledo[3]

[1]Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.
[2]O autor é Especialista em Ciência da Computação pela Universidade Federal do Ceará.
[3]Professor do Departamento de Ciência da Computação – IMECC – UNICAMP.
[4]Professor do Departamento de Computação e Automação – FEE – UNICAMP.

À minha avó Maria Luiza Pinheiro de Freitas, pelo exemplo de luta, firmeza e determinação no qual me inspiro.

Agradecimentos

Concluída a dissertação, entendemos que quanto mais cooperativo mais produtivo é um trabalho científico. Professores, colegas de mestrado, amigos foram co-partícipes deste trabalho. Gostaria de agradecer a todos eles e, em especial, a aqueles que mais participaram deste trabalho de cooperação.

À minha orientadora Profa. Cláudia pela confiança, incentivo e ensinamentos propocionados nos encontros de orientação, nos quais aprendemos a ter uma visão mais objetiva e precisa sobre a área de Bancos de Dados.

Ao Prof. Rogério Drummond, pela solicitude e pelo incentivo inicial para ingressarmos no curso de Mestrado do DCC.

Aos funcionários do DCC, em especial o Luíz e a Solange, pela solicitude no encaminhamento das questões burocráticas.

Aos colegas do Clube do Joio Guto, Cláudio e Carrard, companheiros sempre presentes durante esta caminhada.

Ao Flores e à Milady, pelo apoio e pela amizade demonstrada.

Ao Guimarães, Geraldo Accioly, Welton, Franzé, Pedro Ivo e Cilinha pela prontidão e solicitude em nos ajudar.

Ao meu pai e minha mãe, presenças constantes nos momentos mais críticos de nossa vida, pelo apoio na superação de mais uma etapa.

À Inês, Andrezinho e Bárbara, pelos momentos felizes que nos propocionaram durante este período de superação de obstáculos e por entenderem nossa ausência em tantos momentos de suas vidas.

Ao CNPq, pelo suporte financeiro durante todo o trabalho de Mestrado.

Ao Banco do Nordeste do Brasil (BNB), pelo apoio financeiro necessário a concretização deste trabalho. Destacamos especialmente a pessoa de Antônio de Pádua Araújo, chefe do Departamento de Administração de Recursos Humanos do BNB, pelo entendimento da necessidade de formação de quadros para o Banco e para a região Nordeste.

Resumo

Vários modelos têm sido propostos para incorporar o conceito de Tempo em Banco de Dados. A maioria destes modelos limita-se a incorporar as facilidades temporais em Sistemas de Bancos de Dados Relacionais. Contudo, a maioria da aplicações que requerem um gerenciamento temporal de dados apresenta uma natureza orientada a objeto. As pesquisas sobre a incorporação de Tempo em Bancos de Dados Orientado a Objetos ainda estão em uma fase inicial. Este trabalho apresenta mais uma contribuição para o desenvolvimento das pesquisas nesta área. A contribuição consiste na implementação de um sistema de gerenciamento de tempo para um banco de dados orientados a objetos. Este sistema, a *Camada de Gerenciamento Temporal*, foi implementado sobre o sistema O_2 e permite a definição e gerenciamento de dados temporais orientado a objetos, bem como o processamento de consultas temporais.

Abstract

Many temporal data models have been suggested. A great number of these models is based on incorporating time only for relational databases systems. However, the applications that require temporal data management presents a object-oriented nature. Research on object-oriented database systems is still in its initial phase. This work presents a practical contribution to the research in this area. This contribution consists in the development of a temporal data management system for an object oriented database. This system – The Temporal Management Layer – was built on top of the O_2 database system and allows the definition and management of object oriented temporal data, as well as the processing of temporal queries.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação e Objetivos da Dissertação

Todos os fenômenos do mundo real estão inseridos no contexto temporal. Em outras palavras, os objetos existentes no mundo real e os relacionamentos entre estes objetos têm existência dependente de dimensões temporais. Os eventos ocorrem em pontos específicos do tempo.

Em tese, Sistemas de Bancos de Dados (SBD's) têm a funcionalidade de modelar e representar o mundo real. Contudo, os SBD's convencionais (por exemplo, os SBD's relacionais e orientados a objetos existentes no mercado) não apresentam a capacidade de capturar a semântica temporal existente no mundo real.

Desde o início da década de 70, a necessidade de incorporar o conceito de tempo em bancos de dados foi percebida. Mas, só na década de 80, as pesquisas com Bancos de Dados Temporais (BDT's) tiveram um impulso. Dois fatos foram determinantes para o crescimento das pesquisas na área de BDT's: *(i)* o surgimento de aplicações onde o gerenciamento de informações no contexto temporal é essencial (por exemplo, sistemas de suporte a decisões, as aplicações *Computer-aided* - as *CAx* e as aplicações que demandam o armazenamento e processamento de dados científicos e estatísticos); e *(ii)* a redução nos custos dos dispositivos de memória principal e de massa.

Durante os últimos anos foram propostos vários modelos de banco de dados que incorporam o conceito de tempo para habilitar os SBD's a capturarem a semântica temporal. A maioria destas propostas limitou-se apenas em estender o modelo relacional. Em contrapartida, grande parte das aplicações que requerem um gerenciamento de dados dentro do contexto temporal caracterizam-se por apresentar uma natureza intrinsicamente orientada a objeto. Consequentemente, torna-se imperativo a incorporação do conceito de tempo em bancos de dados orientados a objetos.

As pesquisas nesta direção ainda são incipientes [WD92, McL91, SC91]. O modelo proposto por [Lin93] pode ser considerado o mais completo tanto do ponto de vista da

utilização do paradigma de banco de dados orientado a objeto, quanto do ponto de vista da completeza temporal.

Para responder a essas necessidades, esta dissertação realizou a *implementação de um sistema de gerenciamento temporal de esquemas e de objetos em um Sistema de Banco de Dados Orientado a Objeto.* Esta implementação caracteriza-se por:

1. fundamentar-se no modelo temporal de dados proposto por [Lin93], o **TOODM** e na linguagem de consulta temporal proposta para este modelo, a linguagem **TOOL**;
2. utilizar o SGBDOO O_2 como suporte;
3. funcionar como uma camada sobre os módulos do processador de consultas e do processador de comandos DDL existentes no O_2;

O sistema O_2 não oferece um suporte temporal. Por este motivo, foi necessário especificar e implementar:

- estruturas que viabilizam a representação da evolução temporal de objetos;
- algoritmos que apresentam as seguintes funcionalidades:
  - Algoritmos para processar extensões aos comandos da *Linguagem de Descrição de Dados* (DDL) do O_2, incorporando, desta forma, o conceito de tempo no gerenciamento de dados realizado pelo O_2;
  - Algoritmos para processar consultas temporais construídas com a sintaxe e a semântica da linguagem **TOOL**. Na realidade, estes algoritmos mapeam consultas TOOL em consultas processadas pelo sistema O_2.

Além disso, a dissertação ainda apresenta as seguintes contribuições:

- Apresentar uma revisão bibliográfica das principais propostas de linguagens de consultas e álgebras temporais com quadros comparativos das principais propriedades das linguagens e álgebras estudadas;
- Apresentar extensões à linguagem de consulta TOOL;

1.2 A Incorporação do Contexto Temporal em Banco de Dados

Os modelos propostos para incorporarem o conceito de Tempo em Bancos de Dados fundamentaram-se basicamente em duas abordagens:

- incorporar extensões à semântica de um modelo de dados já existente (o modelo relacional ou o modelo orientado a objeto, por exemplo) com o objetivo de incluir nesta semântica o conceito de tempo; e

- incorporar a um modelo de dados convencional (que só representa o contexto temporal corrente do mundo real) atributos implícitos que representam as dimensões temporais.

Na primeira abordagem, torna-se necessária a formulação de uma lógica para formalizar a semântica temporal do BD. Esta abordagem foi utilizada por [CW83] que formulou a *Intensional Logic* (IL) para incorporar o conceito de tempo na semântica do modelo relacional.

Na segunda abordagem, considerando a incorporação do contexto temporal no modelo relacional, cada relação temporal é considerada como um conjunto de relações instantâneas que contêm atributos temporais implícitos[1] (por exemplo, os atributos *Tempo-de-Transação* e *Tempo-Válido* existentes no modelo proposto por [SA85a]). Nesta abordagem, a lógica suporte do modelo não necessita incorporar o conceito de tempo. Portanto, a linguagem de consulta tem como função básica mapear operações que envolvem o conceito de tempo em operações que manipulam os atributos temporais adicionados às relações instantâneas. Estas operações que manipulam os atributos temporais têm sua semântica expressa através do formalismo do cálculo relacional ou da álgebra relacional tradicional. Com esta abordagem, pode-se implementar um BDT sobre um SGBD relacional tradicional.

1.2.1 O Gerenciamento de Dados no Contexto Temporal

A evolução temporal de objetos e de seus relacionamentos pode ser representada em diferentes eixos de tempo. Em outras palavras, o contexto temporal pode ser representado por diferentes dimensões de tempo. A maioria do modelos de dados temporais propostos apresentam suporte a três dimensões temporais: **Tempo de Transação, Tempo Válido** e **Tempo Definido pelo Usuário**. Estas três dimensões temporais são ortogonais, pois cada uma destas dimensões temporais é representada por um eixo temporal específico.

A dimensão temporal **Tempo de Transação** representa o tempo de armazenamento físico de um dado no Banco de Dados. Em outras palavras, esta dimensão tem a função de representar a evolução temporal de um objeto no Banco de Dados. Portanto, sobre o eixo temporal *Tempo-de-Transação* estarão representados todos os estados de um objeto no Banco de Dados. Como pode ser observado, esta dimensão temporal apresenta semântica que não depende da aplicação cujos dados estão armazenados no BD.

[1] Atributos transparentes ao usuário.

A dimensão temporal **Tempo Válido** representa o tempo de existência de um dado no mundo real. Em outras palavras, representa a evolução temporal de um objeto no mundo real. Desta forma, o eixo *Tempo-Válido* representa o histórico de um objeto no mundo real. Um valor de *Tempo Válido* para um evento é o valor de tempo no qual o evento ocorreu no mundo real, independente do armazenamento deste evento em algum Banco de Dados.

Diferentemente da dimensão *Tempo de Transação*, a dimensão *Tempo Válido* apresenta semântica que pode depender da aplicação. Em outras palavras, o significado de um determinado valor de *Tempo Válido* depende da interpretação que o usuário faz deste valor. Por exemplo, considere o valor de *Tempo Válido* no qual o objeto Bárbara Vitorino começa a existir como pessoa. Em uma instituição bancária, este valor vai representar o momento em que a Bárbara abriu uma conta corrente. Para o ministério da fazenda, este valor vai representar o momento que a Bárbara se registrou no cadastro de pessoas físicas (CPF).

A dimensão **Tempo de Usuário** representa um tipo de dado que apresenta como domínio valores de tempo. Esta dimensão apresenta semântica definida pelo usuário. O gerenciamento de valores para esta dimensão é realizada pelo próprio usuário.

As dimensões temporais *Tempo de Transação* e *Tempo Válido* podem ser incorporadas tanto a nível de um objeto do mundo real, quanto a nível de atributos que compõem um objeto.

1.2.2 O Contexto Temporal em Bancos de Dados Orientados a Objetos

Em Bancos de Dados Orientados a Objetos, quatro tipos distintos de classes podem ser definidas em função das dimensões temporais incorporadas:

- **Classes Instantâneas**: Classes que não apresentam suporte a tempo;
- **Classes Tempo-de-Transação**: Classes que apresentam apenas a dimensão temporal *Tempo de Transação*. Representam, portanto, todos estados (passados e corrente) do seu conjunto de objetos;
- **Classes Tempo-Válido**: Classes que suportam apenas a dimensão temporal *Tempo Válido*. Conseguem representar os tempos de existência dos dados no mundo real. Estas classes representam um histórico do mundo real, enquanto as classes *Tempo-de-Transação* representam o histórico do banco de dados (conceito que se assemelha ao conceito de *Log* do BD);
- **Classes Bitemporais**: Apresentam suporte para as dimensões temporais *Tempo de Transação* e *Tempo Válido*. Representam os vários estados dos objetos e/ou

atributos com seus respectivos históricos do mundo real.

Muitas vezes, as hierarquias de classes (grafo de herança) e de composição (grafo de composição) em um SBDOO dão origem a conflitos entre as especificações de dados fornecidas pelo usuário. Isto torna necessário, portanto, a formulação de um conjunto de heurísticas, as *regras de resolução de conflitos*, para manter o BD em um estado consistente. A incorporação do contexto temporal a um SBDOO pode dar origem a mais uma categoria de conflitos (que não são decorrentes do ambiente orientado a objeto), os **conflitos temporais**. O modelo proposto por [Lin93] apresenta um conjunto de regras para a resolução deste tipo de conflito. Por exemplo, no caso de conflitos temporais decorrentes do grafo de herança, utiliza-se a seguinte regra: as subclasses devem apresentar no mínimo as dimensões temporais de sua superclasse. Se uma classe *C1* for definida como subclasse de uma classe bitemporal *C*, então *C1* obrigatoriamente deverá ser bitemporal.

1.2.3 Consultas Temporais

O processamento eficiente de consultas tem sido crítico no projeto de banco de dados que incorporam dimensões temporais. Independente do modelo de dados utilizado (relacional ou orientado a objeto), o processamento de consultas temporais[2] apresenta os seguintes problemas:

- O volume de dados armazenados em um BDT é de uma ordem de grandeza bastante superior ao volume de dados de um BD convencional. Isto implica que novos métodos de indexação (estruturas e algoritmos de busca) sejam desenvolvidos;
- Os métodos tradicionais de indexação só podem ser utilizados para valores sobre os quais pode ser executado algum tipo de ordenação. Valores do tipo intervalo[3] não suportam algum tipo de ordenação completa. Torna-se necessário o desenvolvimento de novas estruturas de acesso para intervalos (por exemplo, a estrutura de acesso *Índice Temporal* proposta por [DW92]);
- Manipulação de *informações incompletas*:
  - Para um dado objeto ou atributo, podem existir valores desconhecidos ou inexistentes. Em certos casos, pode, até mesmo, haver incertezas quanto à existência do objeto em certos pontos no tempo;
  - Podem existir eventos para os quais o tempo de ocorrência não é conhecido com exatidão, ou seja, existe uma *indeterminação temporal*.

[2] consultas que envolvem o contexto temporal.

[3] A representação de valores através de intervalos é muito utilizada para especificar dimensões temporais, como, por exemplo, Tempo Válido.

O conjunto de problemas existentes no processamento de consultas temporais torna-se ainda mais complexo em um contexto orientado a objeto. Passam também a existir problemas específicos do processamento de consultas em BD's orientado a objeto, como, por exemplo, a inexistência de um formalismo comum para a fase de *Transformações Lógicas* (dentro do processo de otimização de consultas).

1.3 Organização da Dissertação

Com o intuito de atender aos objetivos definidos, esta dissertação foi estruturada da seguinte forma:

- o capítulo 2 apresenta uma revisão bibliográfica das principais propostas de linguagens de consultas e álgebras temporais, apresentando algumas características do respectivos modelos temporais;
- o capítulo 3 descreve o modelo **TOODM** e a linguagem **TOOL** de [Lin93];
- o capítulo 4 descreve as estruturas utilizadas para incorporar o contexto temporal no O_2. É descrito também o processo de implementação destas estruturas;
- o capítulo 5 descreve o funcionamento da *Camada de Gerenciamento Temporal*, desenvolvida sobre o sistema O_2 utilizando as estruturas propostas. O capítulo também descreve o mecanismo de criação de esquemas e classes temporais no O_2;
- o capítulo 6 descreve como foi implementado o processamento de consultas temporais utilizando a *Camada de Gerenciamento Temporal*;
- Finalmente, o capítulo 7 apresenta as conclusões e possíveis extensões a este trabalho.

Capítulo 2

Revisão Bibliográfica

2.1 Introdução

Em um SGBD, as consultas especificadas por usuários são geralmente expressas em linguagens de alto nível (o SQL, por exemplo) que são mapeadas para uma álgebra, denominada de *álgebra alvo* (no caso do SQL, a álgebra alvo é a álgebra relacional). Utilizando-se as propriedades da álgebra, viabiliza-se a otimização do processamento de consultas através da transformação da consulta requerida pelo usuário em expressões algébricas equivalentes à consulta inicial [Mat84], [Eug76], [CD92], [Mic92], [Jac92]. Este processo de transformação possibilita a construção de planos de acesso mais eficientes.

O princípio descrito acima também se aplica a consultas que envolvem o contexto temporal [AS89], [GS90], [LM90], [LM92], [DW92]. Portanto, para se obter um processamento eficiente de consultas temporais, não é suficiente apenas a especificação de uma linguagem de consulta. Torna-se fundamental a formalização de uma álgebra que capture o contexto temporal dos objetos. Desta forma, pode-se, a grosso modo, entender uma álgebra temporal como uma linguagem de consulta de mais baixo nível dentro do processamento de consultas. Sob esta ótica, serão descritas neste capítulo algumas linguagens de consultas e álgebras temporais.

Este capítulo está estruturado da seguinte forma: a seção 2.2 descreve as principais propriedades de algumas linguagens de consultas e álgebras propostas para operarem sobre Bancos de Dados Temporais; a seção 2.3 apresenta quadros comparativos que destacam as principais características das linguagens e álgebras descritas neste capítulo.

2.2 Linguagens de Consultas e Álgebras Temporais

Uma linguagem ou uma álgebra de consulta a bancos de dados podem ser definidas como um conjunto de objetos e um conjunto de operações sobre estes objetos. Por exemplo, no

caso da álgebra relacional, os objetos são do tipo *relação* e o conjunto de operações básicas[1] é formado por: *seleção, projeção, união, diferença de conjuntos* e *produto cartesiano.* As linguagens e álgebras a serem apresentadas neste capítulo diferem pelos tipos de objetos definidos e pelas operações possíveis.

As linguagens de consulta e álgebras também diferem pelos seguintes critérios utilizados no projeto de BDT's:

- Tipo de modelo de dados utilizado como suporte. Os mais utilizados são o relacional e o orientado a objeto;
- Tipos de dimensões temporais incorporadas. As mais comuns são *Tempo de Transação* e *Tempo Válido*, sendo estas duas dimensões ortogonais;
- Nível de representação das dimensões temporais, podendo ser: *(i)* a nível de um objeto do mundo real, ou; *(ii)* a nível de atributos de um objeto, ou; *(iii)* nos dois níveis simultaneamente.

As linguagens e álgebras apresentadas a seguir estão em ordem cronológica de sua proposição e serão identificadas pelo nome de um de seus propositores.

2.2.1 Jones

A linguagem de consulta LEGOL 2.0 [S. 79] utiliza o modelo relacional como modelo de dados suporte. Foi a primeira proposta de linguagem de consulta a incorporar o conceito de tempo.

Foi projetada para ser utilizada em BDT's que incorporam a dimensão temporal *Tempo Válido*, cuja representação é feita através de dois atributos implícitos: *start* e *stop.* Os valores destes atributos representam os extremos do intervalo de tempo em que o objeto ou relacionamento, representados através de uma tupla, são válidos (existem) no mundo real. Estes valores são especificados pelo usuário.

Além das operações básicas da álgebra relacional, a linguagem LEGOL 2.0 incorpora extensões temporais a estas operações para viabilizar a manipulação da dimensão temporal suportada pela linguagem. As operações temporais são: *intersecção temporal, união temporal, diferença temporal* e *pertinência temporal.* Não foi apresentado nenhum formalismo para demonstrar estas operações temporais. Contudo, Jones especificou que a operação de intersecção temporal utiliza a semântica da operação padrão de intersecção, ou seja, a intersecção temporal entre duas tuplas retornará o subintervalo que representa a sobreposição dos intervalos de Tempo Válido de cada tupla.

[1] As outras operações existentes na álgebra relacional podem ser definidas a partir do conjunto de operações básicas, como é o caso da operação de junção que pode ser definida como uma operação de *seleção* sobre uma operação de *produto cartesiano.*

2.2.2 Clifford

Em 1983, Clifford e Warren propuseram um modelo de dados [CW83] cuja semântica incorporava o conceito de tempo. Para isto, foi necessária a formalização da *Intensional Logic* (IL). Esta também foi a primeira proposta a incorporar o contexto temporal a nível de atributos.

O *Historical Relational Data Model* (HRDM) [CC87], um refinamento do primeiro modelo, apresenta dois tipos de objetos: *(i)* conjuntos de pontos no tempo que apresentam semântica semelhante à dimensão temporal Tempo Válido, e; *(ii)* relações históricas onde são associados intervalos de existência, denominado de *lifespan*, para cada atributo e tupla.

A álgebra proposta para o HRDM caracteriza-se por dois aspectos: *(i)* para viabilizar a representação da dimensão temporal a nível de atributos, o modelo não está na *1FN* (primeira forma normal), forçando a álgebra do modelo a também ser uma extensão da álgebra relacional convencional quanto à forma normal; *(ii)* para incorporar diretamente o conceito de tempo, são propostas extensões temporais à semântica dos seguintes operadores da álgebra relacional: intersecção, união, projeção, diferença e produto cartesiano. Por exemplo, o *lifespan* das tuplas resultado de um produto cartesiano é a união dos *lifespan's* das tuplas das relações utilizadas como operandos.

São propostas as seguintes operações temporais: *WHEN* (Ω) retorna o conjunto de intervalos (*lifespan*) para os quais uma condição sobre dados é satisfeita; *TIME-SLICE* (τ) representa uma projeção temporal que restringe as tuplas de uma relação a um subintervalo do *lifespan* original de cada tupla; *SELECT-IF* (σ_{IF}) representa uma seleção temporal que filtra tuplas válidas para uma dada condição em um determinado ponto no tempo; *SELECT-WHEN* (σ_{WHEN}) representa um outro tipo de seleção temporal que filtra tuplas para subintervalos do *lifespan* original de cada tupla nos quais a condição de seleção é satisfeita; *TIME-JOIN* restringe o *lifespan* de cada tupla pelo valor de um atributo temporal.

2.2.3 Tansel

A álgebra proposta por Tansel [Tan86] [TA86] suporta apenas um tipo de objeto, as *relações históricas*. O modelo de BDT, sobre o qual esta álgebra pode operar, é uma extensão ao modelo relacional convencional. Neste modelo de BDT, podem existir os seguintes tipos distintos de atributos: os atributos que incorporam o contexto temporal e os que não incorporam, os atributos atômicos e os não-atômicos.

Os operadores relacionais convencionais são estendidos para incorporar diretamente a semântica temporal. Foram propostos novos operadores cuja funcionalidade é manipular o contexto temporal: *PACK* agrupa tuplas que diferem pelo valor de um único atributo; *UNPACK* replica tuplas para cada valor de atributos não-atomicos; *T-DEC* decompõe um atributo atomico que incorpora o contexto temporal em três atributos, o valor do

atributo e os dois limites do intervalo temporal; *T-FORM* é a operação inversa a T-DEC; *DROP-TIME* remove a dimensão temporal de um atributo; *SLICE, USLICE* e *DSLICE* representam intersecção, união e diferença entre intervalos temporais de atributos, respectivamente.

Esta álgebra é utilizada como *álgebra alvo* para as linguagens de consulta temporais TBE (Time By Example) e HQuel (Historical Quel), também propostas por Tansel.

2.2.4 Snodgrass

Com base na abordagem que utiliza o modelo relacional convencional como suporte para um BDT, Snodgrass propôs um modelo de banco de dados que incorpora o contexto temporal através de atributos implícitos adicionados a relações instantâneas [SA85a], [SA85b]. Snodgrass incorporou ao seu modelo as dimensões temporais *Tempo de Transação* e *Tempo Válido* com semânticas gerenciadas diretamente pelo SGBD, e a dimensão temporal *Tempo de Usuário* com semântica definida e gerenciada pelo usuário.

Com base na existência destas dimensões temporais, uma relação no modelo de Snodgrass pode ser classificada[2] como: *Relação Instantânea* que não apresenta suporte ao conceito de tempo; *Relação Tempo-de-Transação* que apresenta suporte apenas para dimensão *Tempo de Transação*; *Relação Tempo-Válido* que apresenta apenas a dimensão temporal *Tempo Válido*; *Relação Bitemporal* que apresenta suporte para as duas dimensões temporais.

Snodgrass formulou para este modelo a linguagem de consulta *TQuel* (Temporal QUEry Language) [Sno87], um superconjunto da linguagem *Quel* (que foi formulada para funcionar como linguagem de consulta do INGRES, um SGBD relacional). A linguagem TQuel foi projetada para estender o mínimo possível, tanto sintaticamente quanto sematicamente, a linguagem Quel. Todo o conjunto de operadores e cláusulas da linguagem Quel são válidos para o TQuel.

A manipulação das dimensões temporais apresentadas pelo modelo é feita através de três novas cláusulas: *WHEN* seleciona tuplas em função de um predicado temporal, operando sobre a dimensão *Tempo Válido*; *VALID* especifica o tempo para o qual as tuplas resultado são válidas; *AS-OF* realiza uma operação de *Rollback*, cuja semântica é retornar o BD a um estado especificado como argumento, operando, portanto, sobre a dimensão *Tempo de Transação.*

São propostos também os *construtores temporais*[3] *BEGIN OF, END OF, OVERLAP* (intersecção temporal) *EXTEND* (união temporal) e os *operadores temporais*[4] *PRECEDE,*

[2]Taxonomia utilizada em [C.S92].

[3]Operadores unários e/ou binários que operam sobre uma ou mais dimensões temporais e resultam em uma dimensão temporal, em outras palavras, são operadores que operam sobre o tempo e retornam tempo.

[4]Operadores que retornam um valor booleano.

OVERLAP, EQUAL.

Snodgrass demonstra o formalismo semântico da linguagem TQuel através do cálculo relacional de tuplas, tendo como ponto de partida o formalismo semântico da linguagem Quel.

Em [LEMS87], McKenzie e Snodgrass formulam uma extensão à álgebra relacional convencional para habilitar esta álgebra a capturar o contexto temporal. Esta álgebra temporal incorpora o contexto temporal através da manipulação dos atributos implícitos que representam as dimensões temporais.

2.2.5 Ariav

Ariav propôs um modelo temporal de dados, o *Temporally Oriented Data Model* [Ari86], cuja característica principal é a represenatção abstrata dos dados inseridos no contexto temporal através de *cubos de dados*. Neste modelo que é uma extensão ao modelo relacional, uma relação é "vista" (entendida) como um cubo no qual duas de suas dimensões representam as dimensões de uma relação instantânea (linhas e colunas) e a terceira dimensão do cubo representa a dimensão temporal inserida na relação. Na realidade, o cubo de dados pode ser entendido como a sobreposição dos vários estados da relação que este cubo representa.

Para efetuar consultas sobre este modelo, Ariav formulou a linguagem de consulta TOSQL, uma extensão ao SQL. Uma consulta TOSQL apresenta a seguinte sintaxe (representada através da BNF):

<consulta> ::= <consulta-base> <espec-objetos> <espec-temporal> <qualif-temporal>
<consulta-base> ::= SELECT <lista-de-atrib> FROM <nome-cubo>
*<lista-de-atrib> ::= * | < $atrib_1$ >, ... , < $atrib_n$ >*
<nome-cubo> ::= um nome identificando um cubo de dados
<espec-objetos> ::= ALL-OBJECTS |
WHERE <predicado> <modo-generaliza-temporal>
<modo-generaliza-temporal> ::= EVERYWHEN | SOMEWHEN
<espec-temporal> ::= <período-tempo> <dimensão-temporal>
<período-tempo> ::= AT <ponto-tempo> |
WHILE <predicado> <limites-de-tempo> |
DURING (<t> - <t>) |
BEFORE <t> |
AFTER <t>
<dimensão-temporal> ::= ALONG RT | ALONG <tsa>
<ponto-tempo> ::= PRESENT | <t>
<limites-de-tempo> ::= DURING (- ∞ - + ∞) | DURING (<t> - <t>)
<t> ::= valor de tempo

<qualif-temporal> ::= AS-OF <ponto-tempo> <dimensão-temporal>

A linguagem TOSQL também apresenta funções agregadas inseridas no contexto temporal, propriedade não especificada para as outras linguagens de consulta temporal.

2.2.6 Navathe

Navathe e Ahmed formularam um modelo de BDT, o *Temporal Relational Model* [NA87], [NA89]. Este modelo caracteriza-se por incorporar a dimensão *Tempo Válido* no modelo relacional convencional. Apesar do modelo associar a dimensão temporal a atributos, fisicamente esta associação se dá a nível de tupla com o objetivo de se garantir a 1FN, o que simplifica a formulação de uma álgebra para o modelo. Esta incorporação do contexto temporal a nível de atributos, mas com representação física a nível de tuplas, é viabilizada através da decomposição de uma tupla de grau **n** em **n-1** tuplas de grau **4**, conservando-se as dependências funcionais entre os atributos não-chaves e os atributos chaves. Além do atributo chave e do atributo não-chave, cada tupla conterá dois atributos, o *Time-Start* e o *Time-End*, que representam os limites do intervalo de validade de um atributo.

A Álgebra que os autores propuseram para este modelo foi utilizada como linguagem alvo para a linguagem temporal TSQL. Os dois tipos de objetos manipulados por esta álgebra são: as *Relações Históricas* e as *Relações Instantâneas.*

A semântica dos operadores da álgebra relacional convencional é mantida, e novos operadores são introduzidos para manipular o contexto temporal. Os novos operadores são os seguintes: *TIME-SLICE, INNER TIME-VIEW* e o *OUTER TIME-VIEW* são tipos de seleção temporal; *TCJOIN* e *TCNJOIN* são tipos de junção temporal e que têm semânticas idênticas à da operação padrão de intersecção.

2.2.7 Sadeghi

A Álgebra proposta por Sadeghi [Sad87] foi formulada para funcionar como linguagem alvo da linguagem de consulta HQL, e é uma extensão à álgebra relacional convencional.

Os objetos desta álgebra são do tipo *Relações Históricas.* Esta álgebra, como extensão à álgebra relacional, opera sobre dois atributos implícitos, *Start* e *Stop*, para capturar a semântica temporal. Estes dois atributos representam os limites do intervalo temporal que representa a dimensão *Tempo Válido.* A dimensão temporal é incorporada a nível de tupla.

São propostas versões temporais para os operadores *união, diferença, produto cartesiano, seleção, projeção* e *junção.* Os operadores *produto cartesiano* e *junção* apresentam a semântica da operação padrão de intersecção. Para viabilizar seleções temporais, é introduzido o operador *WHEN.*

2.2.8 Gadia

Gadia formulou um modelo para incorporação do conceito de Tempo em [Gad88], o *Homogeneous Relational Model*, que se caracteriza por: *(i)* ser uma extensão ao modelo relacional; *(ii)* apresentar dois tipos de objetos, o *Elemento Temporal* e a *Relação Histórica*, e; *(iii)* apresentar a propriedade da *homogeneidade*.

Elemento Temporal é uma união finita de intervalos disjuntos. Os atributos no HRM representam funções que mapeiam elementos temporais aos conjuntos domínio dos atributos. *Homogeneidade* é uma propriedade existente em alguns modelos de BDT que determina o seguinte: *os valores dos atributos de uma dada tupla são funções do mesmo elemento temporal*, em outras palavras, todos os atributos de uma mesma tupla estão definidos e são válidos para o mesmo intervalo de tempo. Esta propriedade garante a simplificação na formulação de uma álgebra para o modelo, pois permite definir uma álgebra como extensão à álgebra relacional convencional, mesmo quando o modelo de BDT permite atributos não-atomicos.

Como linguagem de consulta para o HRM, Gadia e Vaishnov formularam a linguagem de consulta HTQuel, uma linguagem estilo Quel. Além das cláusulas Quel, a linguagem HTQuel apresenta a cláusula *DURING* para realizar seleções temporais, e a cláusula *TDOM* para mapear tuplas e/ou relações para seus domínios temporais (projeção temporal).

Para manipular as dimensões temporais, são introduzidos os seguintes operadores: *FIRSTINTERVAL, PREVIOUSINTERVAL, LASTINSTANT* e *NEXTINSTANT*.

2.2.9 Elmasri

Em [EW90], Elmasri e Wuu propõem um modelo temporal de dados, o *Temporal EER* (*TEER*), que se caracteriza em incorporar a dimensão temporal *tempo Válido* ao modelo *EER*[5] (*Enhanced Entity-Relationship*). A abordagem utilizada para formular este modelo foi a de adaptar a proposta de Gadia [Gad88] para o modelo EER.

A seguir é descrito como é realizada a incorporação da dimensão *Tempo Válido* pelo modelo proposto por Elmasri. Para cada valor **V** de um atributo pertencente ao conjunto de atributos de uma entidade é associado um intervalo de tempo **I**. O intervalo **I** representa o período de tempo no qual o valor **V** foi/é/será válido no mundo real.

O modelo apresenta ainda o conceito de *lifespan* de uma entidade. Cada entidade no Banco de Dados apresenta um identificador de entidade (*SURROGATE*), que é um atributo definido e gerenciado pelo sistema. Para cada *SURROGATE* é associado um intervalo de tempo. Este intervalo de tempo corresponde ao *lifespan* da entidade. Em outras palavras, este intervalo de tempo representa o período no qual a entidade foi/é/será

[5] O modelo EER é uma extensão ao modelo Entity-Relationship.

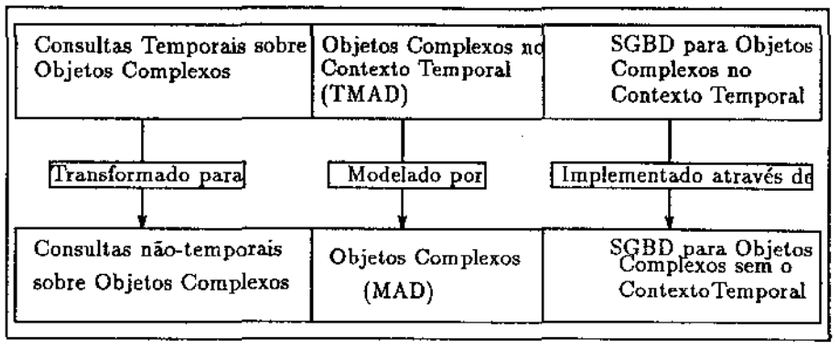

Figura 2.1: Processo de mapeamento do TMAD para o MAD.

válida no mundo real.

A linguagem de consulta temporal proposta para o *TEER* é uma extensão à linguagem *GORDAS* que se caracteriza por ser uma linguagem funcional. A linguagem de consulta temporal pode retornar dados ou valores de tempo.

Uma consulta temporal que retorna dados apresenta a seguinte sintaxe:

GET <projeção> OF <conjunto-de-entidades> : <projeção-temporal>
WHERE <predicado-não-temporal> — <predicado-temporal>

Para efetuar consultas temporais que retornam valores de tempo foram propostas as funções *TIME*, *FI* (First Instant) e *LI* (Last Instant). A função *TIME* retorna a união de todos os intervalos de tempo de um determinado tributo. Em outras palavras, retorna o histórico de um atributo. As funções *FI* e *LI* retornam respectivamente o primeiro e o último intervalo de um conjunto de intervalos.

2.2.10 Käfer

Käfer [Wol92] ressalta que a maioria das aplicações que requerem um gerenciamento de seus dados dentro de um contexto temporal também necessita de um suporte para o gerenciamento de objetos complexos, propriedade não apresentada pelos BDT's baseados no modelo relacional.

A partir desta constatação, Käfer formulou uma extensão temporal ao modelo MAD, um modelo de dados adequado para o gerenciamento de objetos complexos. Para isto, incorporou no modelo MAD temporal (TMAD) atributos implícitos para representar as dimensões temporais. A figura 2.1, apresentada originalmente em [Wol92], ilustra o processo de mapeamento do TMAD para o MAD.

O TMAD apresenta as dimensões temporais *Tempo de Transação* e *Tempo Válido.* Para consultar este modelo de BDT, Käfer formulou extensões à linguagem de consulta existente no MAD, a linguagem MQL. Estas extensões restringem-se à manipulação das dimensões temporais introduzidas no TMAD.

Uma consulta no TMAD apresenta a seguinte sintaxe:

T_SELECT <lista-de-projeção> [Projeção-Temporal]
T_FROM <definição-do-objeto>
T_WHERE <predicado> [Seleção-Temporal]
Seleção-Temporal ::= AT <ponto-tempo> |
{ SOMETIMES | ALWAYS }
DURING [<ponto-tempo>,<ponto-tempo>]
Projeção-Temporal ::= CORRESPONDING |
AT <ponto-tempo> |
DURING [<ponto-tempo>,<ponto-tempo>]

2.2.11 Wuu

Wuu e Dayal apresentaram em [WD92] uma proposta para introduzir o contexto temporal em banco de dados orientados a objetos, estendendo um SBDOO já existente, o OODAPLEX. O OODAPLEX, por sua vez, já é uma extensão ao DAPLEX (SGBD funcional) para introduzir neste último o ambiente orientado a objetos. A versão temporal do OODAPLEX caracteriza-se por representar objetos e relacionamentos entre objetos através de funções que retornam uma outra função[6], mapeando, desta forma, valores de tempo para valores instantâneos (os valores de atributos, por exemplo).

As dimensões temporais apresentadas pelo OODAPLEX temporal são *Tempo de Transação* e *Tempo Válido.* A dimensão *Tempo de Transação* também apresenta uma semântica de identificador de tupla (conceito distinto de identificador de objeto) dentro do modelo.

Para efetuar consultas, foram introduzidas na linguagem de consulta do OODAPLEX convencional as seguintes funções que operam com o conceito de tempo: *lifespan(o)* e *T_extent.* A função *lifespan* retorna o conjunto de pontos no tempo durante os quais um objeto ou um conjunto de objetos é válido para o mundo real. A função *T_extent(C)(t)* retorna o conjunto de objetos de C que são válidos para o tempo t.

[6]Esta função origina-se do modelo OODAPLEX instantâneo e é utilizada para mapear um atributo a um valor do conjunto domínio deste mesmo atributo.

2.2.12 Medina

Em [Lin93], Medina apresenta um modelo de BDT, onde é proposta a incorporação das dimensões temporais *Tempo de Transação* e *Tempo Válido* em bancos de dados orientados a objetos. Não foi apresentada nenhuma proposta de implementação para o modelo e o autor se detem a formular a incorporação do contexto temporal no modelo de dados orientado a objeto. Este modelo de BDT não é dependente de nenhum SGBD específico, o que não acontece com a proposta de Wuu que é dependente do OODAPLEX.

Para formular sua proposta de linguagem de consulta temporal, Medina utilizou como suporte a linguagem de consulta do O_2, a O_2Query. Esta abordagem de estender uma linguagem de consulta já existente viabiliza o aproveitamento do processador de consulta da linguagem suporte. Uma consulta temporal é mapeada em consultas básicas da linguagem estendida.

A linguagem Medina apresenta dois tipos de consultas temporais:

- consultas que retornam *valores temporais* (tempo), e;
- consultas que retornam *dados* dentro de um contexto temporal.

Consultas que retornam valores temporais são especificadas através das seguintes cláusulas temporais: **TWHEN** retorna um conjunto de tempos de transação nos quais um predicado é satisfeito para um tempo válido especificado pela cláusula **VALID**; **VWHEN** retorna um conjunto de tempos válidos nos quais um predicado é satisfeito em um estado do banco de dados especificado pela cláusula **INDB**.

Consultas que retornam valores temporais podem ser construídas a partir da seguinte sintaxe:

TWHEN | *VWHEN* <*predicado*>
FROM <*lista-de-classes*>
VALID | *INDB* <*expressão-temporal*>

Consultas que retornam dados para um determinado contexto temporal são especificadas pelas seguintes cláusulas de projeção temporal: **TSLICE** restringe os dados a um tempo de transação especificado; **VSLICE** restringe os dados a um tempo válido especificado. Este tipo de consulta apresenta a seguinte sintaxe:

SELECT <*lista-de-projeção-não-temporal*> *[Projeção-Temporal]*
FROM <*lista-de-classes*>
WHERE <*predicado*> *VALID* | *INDB*
Projeção-Temporal ::= *TSLICE* | *VSLICE* <*expressão-temporal*>

São propostos também os *operadores temporais* **BEFORE, AFTER, OVERLAPS, CONTAINS, IN, EQUAL, MEETS, FOLLOWS, ADJACENT** e os *construtores temporais* **FIRST_INSTANT, LAST_INSTANT, FIRST_INTERVAL, LAST_INTERVAL, T_UNION, T_MINUS, T_INTER, TEMPORAL_EL**. Estes operadores e construtores temporais podem ser combinados com as sintaxes básicas dos tipos de consultas existentes nesta linguagem.

2.3 Quadros Comparativos

São apresentados quadros comparativos, que sintetizam as principais características das linguagens e álgebras de consulta presentes nesta proposta. Algumas das características a serem apresentadas referem-se ao modelo de BDT, pois estas podem ser determinantes na formulação da linguagem e/ou da álgebra temporal.

A característica *Operações Padrão do Modelo Suporte* (2.2) refere-se à introdução ou não de extensões aos operadores existentes no modelo de dados utilizado como suporte. O critério *Indeterminação Temporal* (2.3) foi conceituado no capítulo 1. A propriedade *Implementabilidade Demonstrada* pode ser comprovada de duas formas: *(i)* formalmente através de semânticas fundamentadas em uma álgebra bem definida, e; *(ii)* empiricamente através de uma implementação eficiente. A dissertação também se propõe a ser uma demonstração, utilizando-se a abordagem empírica, da implementabilidade da linguagem proposta por Medina.

	Modelo de Dados Suporte	Dimensões Temporais	Nível Incorporação das Dimensões Temporais
Jones	Relacional	Tempo Válido	Tupla
Clliford	Entidade Relacionamento	Tempo Válido	Tupla e/ou Atributo
Tansel	Relacional	Tempo Válido	Atributo
Snodgrass	Relacional	Tempo Válido e Tempo de Transação	Tupla
Ariav	Relacional	ilimitado	Tupla
Navathe	Relacional	Tempo Válido e Tempo de Transação	Atributo
Sadeghi	Relacional	Tempo Válido	Tupla
Gadia	Relacional	Ilimitado	Atributo
Elmasri	EER	Tempo Válido	Atributo
Käfer	Objeto Complexo	Tempo Válido Tempo de Transação	Objeto
Wuu	Orientado a Objeto (Funcional)	Tempo Válido e Tempo de Transação	Atributo
Medina	Orientado a Objeto	Tempo Válido e Tempo de Transação	Objeto atômico

Tabela 2.1:

	Operações Padrão do Modelo Suporte	Novas Operações
Jones	Mantidas	Intersecção Temporal,União Temporal, Diferença Temporal,Pertinência Temporal
Clliford	Estendidas	When,Select-If,Select-When, Time-Slice,Time-Join
Tansel	Estendidas	Pack,Unpack,T-Dec,T-Form,Drop-Time, Slice,U-Slice,D-Slice,Enumeration
Snodgrass	Estendidas	When,Valid,As-Of,Overlap, Precede,Begin-of,End-of,Extend
Ariav	Estendidas	EveryWhen,SomeWhen,While, During,Before,After,Along,As-Of
Navathe	Mantidas	Time-Slice,TCJoin,TCNJoin, Inner Time-View,Outer Time-View
Sadeghi	Estendidas	When,Time-Join
Gadia	Estendidas	TDOM,During,FirstInterval, PreviousInterval,LastInstant,NextInstant
Elmasri	Estendidas	TIME, FI, LI
Käfer	Estendidas	At,Sometimes,Always, During, Corresponding
Wuu	Estendidas	Lifespan,T _ extent
Medina	Mantidas	VWhen,TWhen,VSlice,TSlice,Before,After, Overlaps, Contains,In,Equal,Meets,Follows, Adjacent,First_Instant, Last_Instant, First_Interval, Last_Interval, T_Minus, T_Union, T_Inter, Temporal_El

Tabela 2.2:

	Linguagem de Consulta Temporal	Álgebra Temporal	Estratégias de Otimização	Manipulação de Indeterminação Temporal
Jones	LEGOL 2.0	√	∇	∇
Clliford	∇	√	NA	∇
Tansel	TBE/HQuel	√	∇	∇
Snodgrass	TQuel	√	P	P
Ariav	TOSQL	√	∇	∇
Navathe	TSQL	√	∇	P
Sadeghi	HQL	√	∇	∇
Gadia	HTQuel	√	∇	∇
Elmasri	GORDAS (Temporal)	∇	∇	P
Käfer	MQL (Temporal)	∇	∇	P
Wuu	OODAPLEX (Temporal)	√	√	∇
Medina	TOOL	∇	∇	∇

Tabela 2.3:

√ - Apresenta propriedade.
∇ - Não apresenta a propriedade.
P - Apresenta parcialmente a propriedade.
NA - Não se aplica.

	Consultas Histórica	Operação de *Rollback*	Linguagem de Consulta não-procedimental	Implementabilidade Demonstrada
Jones	√	∇	∇	√
Clliford	√	∇	√	∇
Tansel	√	∇	∇	√
Snodgrass	√	√	√	√
Ariav	√	√	√	∇
Navathe	√	∇	√	∇
Sadeghi	√	∇	√	∇
Gadia	√	∇	√	√
Elmasri	√	∇	√	∇
Käfer	√	√	√	√
Wuu	√	√	∇	√
Medina	√	√	√	∇

Tabela 2.4:

√ - Apresenta propriedade.
∇ - Não apresenta a propriedade.

Capítulo 3

O Modelo TOODM e a Linguagem de Consulta TOOL

3.1 Introdução

O objetivo básico da dissertação, como já foi mencionado no capítulo 1, é implementar um sistema de gerenciamento temporal de objetos e esquemas em um Sistema de Banco de Dados Orientado a Objetos. Este sistema utiliza como modelo temporal suporte o **TOODM** (*Temporal Object-Oriented Data Model*) proposto por Medina em [Lin93].

O **TOODM** é uma extensão ao modelo de dados orientado a objetos, introduzindo, neste modelo, as dimensões temporais *Tempo de Transação* e *Tempo Válido*. Isto implica que, em função do tipo e do número de dimensões temporais, podem existir quatro tipos de classes no **TOODM**:

- classe *Instantânea*;
- classe *Tempo-de-Transação*;
- classe *Tempo-Válido*;
- classe *Bitemporal*.

O modelo ainda apresenta regras para resolução de conflitos temporais (veja conceituação no capítulo 1).

A linguagem de consulta proposta por Medina para processar as operações temporais introduzidas pelo **TOODM** é a linguagem **TOOL** (*Temporal Object-Oriented Language*). Esta linguagem caracteriza-se por ser um superconjunto da linguagem O_2Query (linguagem de consulta apresentada pelo O_2). Em outras palavras, a linguagem **TOOL** apresenta suporte para processar consultas construídas com a sintaxe **TOOL** e consultas construídas com a sintaxe O_2Query.

Este capítulo apresenta uma descrição do modelo **TOODM** e da linguagem de consulta **TOOL**. Por este motivo, foi estruturado da seguinte forma: a seção 3.2 descreve as principais características do modelo TOODM, que utilizamos como base para nossa implementação. Nesta seção, é apresentada também uma introdução ao problema de modelagem e representação de Tempo. A seção 3.3 apresenta uma visão geral da linguagem de consulta TOOL, descrevendo seus operadores, construtores e cláusulas temporais.

3.2 O Modelo TOODM

O modelo TOODM representa o Tempo através da introdução de atributos especiais. Estes atributos têm a função de representar as dimensões temporais suportadas pelo modelo. Para o paradigma de orientação a objetos, um atributo é um objeto. Dentro deste enfoque, o modelo TOODM apresenta três tipos de objetos:

- objetos do tipo **TIME**: objetos utilizados para representar as dimensões temporais;
- objetos invariantes no tempo: objetos que não são compostos por nenhum objeto do tipo TIME;
- objetos variantes no tempo: objetos complexos compostos por pelo menos um objeto do tipo TIME.

Pode-se entender os objetos do tipo TIME como atributos que têm a função de representar as dimensões temporais. O TOODM pode representar a evolução temporal de objetos complexos e objetos simples. Segundo o paradigma de orientação a objetos, objetos complexos são compostos por objetos simples. Isto implica que o TOODM representa a evolução temporal de objetos e de seus componentes (conceito equivalente ao de atributos no modelo relacional).

3.2.1 Categorias Temporais

Segundo Medina, **categoria temporal** de uma classe é a classificação desta quanto ao número de dimensões temporais incorporadas a seus objetos.

Para o TOODM, podem existir quatro tipos de categoria temporal em um banco de dados: classes *Instantâneas*, classes *Tempo-de-Transação*, classes *Tempo-Válido* e classes *Bitemporais*. O capítulo 1 apresenta a conceituação de cada um destes tipos de classes.

3.2.2 Temporalização

Os relacionamentos entre objetos são representados no modelo orientado a objetos através dos grafos de herança e composição. As propriedades de herança e composição podem

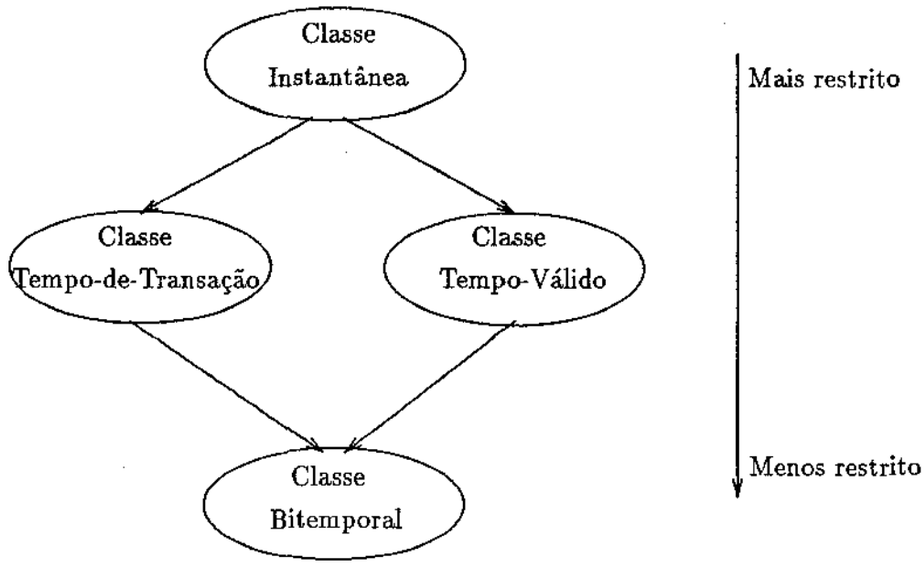

Figura 3.1: Ordem parcial das classes em função de sua categoria temporal.

gerar conflitos durante a evolução de esquemas. Ao se incorporar o contexto temporal em um banco de dados orientado a objetos, o número destes conflitos pode crescer como decorrência do que denominamos de *conflitos temporais* (veja conceituação no capítulo 1). O TOODM apresenta heurísticas que tentam resolver os conflitos temporais. O processo de execução destas heurísticas é denominado de **temporalização** [Lin93]. Este processo deve ser executado durante a evolução de esquemas (criação e alteração).

Segundo Medina, o processo de temporalização é composto de duas fases: resolução de conflitos temporais pelo grafo de herança e pelo grafo de composição.

Para definir o processo de temporalização foi proposta uma propriedade temporal para os grafos de herança e de composição. Denominamos esta propriedade como **invariante temporal** (o conceito de invariantes no modelo orientado a objetos é definido em [Jay87]) dos grafos de herança e de composição. Qualquer alteração na definição ou na estrutura de uma classe deve preservar esta invariante:

$\longrightarrow$ *Ordem parcial entre as categorias temporais*: uma classe deve manter no mínimo as dimensões temporais incorporadas por suas antecessoras.

O esquema apresentado na figura 3.1 ilustra a ordem parcial entre as classes em um BDOO modelado através do TOODM. A ordem representada na figura 3.1 indica, por exemplo, que uma classe *Bitemporal* pode apresentar com antecessora qualquer outro tipo de classe. Porém, uma classe *Tempo-de-Transação* só pode apresentar como antecessora uma classe *Instantânea*.

Temporalização por Herança

A temporalização por herança tem a função de compatibilar as propriedades temporais em uma hierarquia de classes. Em outras palavras, na temporalização por herança as subclasses são compatibilizadas com as superclasses.

O processo de temporalização por herança garante a invariante temporal da ordem parcial na definição de uma classe em função da categoria temporal de sua superclasse. A temporalização é efetuada através da criação de uma nova classe com as propriedades temporais que garanta a invariante da ordem parcial.

Por exemplo, considere a classe *Pessoa* definida como *Tempo-Válido*. Considere ainda que a classe *Funcionário* é definida como subclasse da classe *Pessoa* e do tipo *Tempo-de-Transação*. A definição da classe *Funcionário* contraria a invariante da ordem parcial. Por este motivo, o processo de temporalização gera uma nova definição da classe *Funcionário* como *Bitemporal*.

Medina não trata o processo de temporalização por herança para o caso de herança múltipla.

Temporalização por Composição

Segundo Medina, a temporalização por composição é necessária, pois para manter um objeto composto em um certo valor de tempo é preciso manter os valores de todos os seus componentes neste valor de tempo. No processo de temporalização por composição as classes componentes são compatibilizadas com as classes compostas.

Por exemplo, considere a classe *Departamento* definida como *Instantânea* e a classe *Funcionário* definida como *Bitemporal*. Considere ainda o atributo *lotação* pertencente à classe *Funcionário*. Foi definido que o tipo do atributo *lotação* é *Departamento* (*lotação:Departamento*).

Pelo processo de temporalização por composição, uma nova classe é inserida ao esquema. Esta nova classe é classe *Departamento* definida como *Bitemporal*. Portanto, existirão duas classes *Departamento* no esquema, diferindo apenas pela categoria temporal, uma classe seria *Instantânea* e a outra *Bitemporal*.

3.2.3 Evolução de Esquema

Em [Lin93], são apresentadas regras que garantem a consistência temporal de um banco de dados durante a evolução de esquemas. Medina classificou estas regras em cinco grupos:

1. Regras para criação/remoção de uma classe;
2. Regras para alteração no grafo de composição;
3. Regras para alteração no grafo de herança;

4. Regras para alteração da categoria temporal de uma classe;

5. Regras para alteração em métodos.

3.2.4 Tempo - Modelo e Representação

O modelo TOODM preocupa-se com a modelagem de dados temporais, sem determinar como o Tempo deve ser modelado e representado. O problema de representação do Tempo é abordado por vários pesquisadores, como por exemplo [Sno92] [DS92] e [DS93].

A definição de um modelo e de uma representação de Tempo envolve a seleção de opções para cada propriedade pertencente ao domínio do Tempo. Estas propriedades são ortogonais aos eixos temporais.

As propriedades mais conhecidas para caracterizar um modelo de Tempo são:

- Modelo de Tempo;
- Densidade do eixo temporal;
- Número de dimensões temporais;
- Tipos de representação de marcas de tempo (*time-stamps*);
- Granularidade;

A seguir, estas propriedades serão conceituados. Serão apresentadas as opções na definição da representação de Tempo para cada propriedade.

Modelo de Tempo

Os modelos estruturais de tempo mais conhecidos são:

- *Modelo Linear*: neste modelo, o Tempo avança inexoravelmente do **passado para o futuro** de forma linear e ordenada. Em outras palavras, as marcas de tempo neste modelo estão linearmente ordenadas;
- *Modelo Ramificado* (Modelo de Futuros Possíveis): neste modelo, o Tempo ocorre de forma linear e ordenada do **passado para o presente**. A partir dai, existe uma ramificação da linha do Tempo em várias linhas. Estas linhas representam sequências de eventos que potencialmente podem ocorrer no futuro. Portanto, no Modelo Ramificado, os valores de tempo estão parcialmente ordenados (apenas os valores de tempo que vão até o presente).

Densidade do Eixo Temporal

Existem três tipos distintos de densidade para representar um eixo temporal. Para cada tipo de densidade é associado um modelo de representação para os valores de tempo.

Os três tipos de modelo para representar marcas de tempo são:

- *Modelo Discreto*: os valores de tempo são representados através de números naturais. Em outras palavras, este modelo apresenta um isomorfismo com o conjunto dos números naturais. Desta forma, a densidade de um eixo temporal (representado através deste modelo) equivale ao número de valores pertencentes ao conjunto dos números naturais;

- *Modelo Denso*: os valores de tempo são representados ou por números reais ou por números racionais. Em outras palavras, neste modelo, podem existir valores de tempo representados por números reais e valores de tempo representados por números racionais. Portanto, neste modelo ainda podem existir *gaps* entre *chronons*[1];

- *Modelo Contínuo*: os valores de tempo são representados através de números reais, ou seja, este modelo é isomórfico ao conjunto de números reais.

Número de Dimensões Temporais

Esta propriedade define o número de eixos temporais, sobre os quais será gerenciada a evolução temporal dos dados.

A maioria dos modelos temporais utiliza dois eixos temporais, o eixo *Tempo-de-Transação* e o eixo *Tempo-Válido*, para representar a evolução temporal dos objetos do mundo real. Existem modelos que propõem um número infinito de eixos temporais [Ari86] e [Gad88].

Tipos de Representação de Marcas de Tempo

O conceito de marcas de tempo (*time-stamps*) possui semântica de registros de tempo. Existem três formatos básicos para se registrar tempo: **evento**, **intervalo** e **span**.

O formato **evento** representa marcas de tempo por um ponto no eixo temporal. Em outras palavras, a cada evento é associado uma valor de tempo sobre um determinado eixo.

Através do formato **intervalo**, o tempo é registrado como um lapso de tempo ocorrido entre dois eventos, onde um evento especifica o início da marcação do tempo (limite temporal inferior) e o outro evento especifica o término da marcação do tempo (limite temporal superior).

[1]Conceitua-se como *chronon* a menor unidade indivisível de tempo. Este conceito equivale ao conceito de ponto em termos de espaço.

O formato **span** registra tempo através de tempos relativos a um determinado ponto no eixo temporal. Na realidade, o formato *span* funciona como um contador de *chronons*. Como podem existir *chronons* de duração constante e *chronons* de duração variável (por exemplo, se utilizarmos como *chronon* a métrica mês, teremos um tipo de *chronon* de duração variável), pode-se ter, então, dois tipos de *span*:

- *span constante*: utiliza *chronons* de duração constante;
- *span variável*: utiliza *chronons* de duração variável.

Granularidade

Esta propriedade define a precisão com que um valor de tempo pode ser representado. A granularidade pode ser expressa através de um único componente. Esta granularidade é denominada de **granularidade simples**. Existe ainda a **granularidade composta** que é expressa através de vários componentes.

Um mesmo valor de tempo pode ser expresso através dos dois tipos de granularidade e apresentar a mesma precisão. Por exemplo, considere como origem de um determinado eixo temporal a seguinte marca de tempo: zero hora, zero minuto, zero segundo do dia 01 de janeiro de 1970. Considere ainda que foi definida como granularidade a métrica *segundo* para representar marcas de tempo. A marca de tempo referente à zero hora, três minutos, 45 segundos do dia 01 de janeiro de 1970 pode ser representada, segundo os dois tipos de granularidade, por:

- *225* – granularidade simples; ou
- *<1970,01,01,00,03,45>* – granularidade composta.

3.3 A Linguagem de Consulta TOOL

A sintaxe e a semântica da linguagem TOOL foi definida a partir da linguagem de consulta existente no sistema O_2, a O_2Query.

A linguagem O_2Query caracteriza-se por ser uma linguagem funcional. Isto significa que uma consulta é uma função cujos argumentos podem ser outras consultas ou outras funções (para maiores detalhes sobre o sistema O_2 e seu processamento de consultas veja [dddO91], [Tec92], [CD92] e [Chr88]).

3.3.1 Os Operadores e Construtores Temporais da Linguagem TOOL

Os operadores definidos para a linguagem TOOL, conforme sintaxe apresentada em [Lin93], são os seguintes:

- BEFORE (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando o último instante de *operando_1* é maior que o primeiro instante de *operando_2*.
- AFTER (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando o primeiro instante de *operando_1* é posterior ao último instante de *operando_2*.
- OVERLAPS (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando os dois operandos se sobrepõem em algum instante de tempo.
- CONTAINS (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando *operando_1* contem todos os valores de tempo pertencentes a *operando_2*.
- IN (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando *operando_2* contem todos os valores de tempo pertencentes a *operando_1*.
- EQUAL (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando *operando_1* apresenta uma marca de tempo equivalente a marca de tempo apresentada por *operando_2*.
- MEETS (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando *operando_2* inicia-se no instante seguinte (próximo *chronon*) ao instante final de *operando_1*.
- FOLLOWS (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando a marca de tempo representada por *operando_1* inicia-se no instante seguinte (próximo *chronon*) ao instante final de *operando_2*.
- ADJACENT (*operando_1,operando_2*) ⟶ retorna o valor *booleano* **True** quando *operando_1* inicia-se no instante seguinte (próximo *chronon*) ao instante final de *operando_2* ou o inverso.

Além destes operadores, estendemos a linguagem TOOL incluindo mais dois operadores temporais: **Before_Overlap** e **After_Overlap**. A sintaxe e a semântica destes operadores serão descritas no capítulo 6.

Os construtores temporais propostos por Medina para a linguagem TOOL são os seguintes:

- INTERVAL (*operando_1,operando_2*) ⟶ retorna um intervalo cujo limite inferior é o evento representado por *operando_1* e o limite superior é o evento representado por *operando_2*.
- T_UNION (*operando_1,operando_2*) ⟶ retorna o intervalo de tempo construído pela união de *operando_1* com *operando_2*.

- T_MINUS (*operando_1,operando_2*) ⟶ semântica equivalente a operação de diferença de conjuntos. Em outras palavras, retornará o subintervalo de tempo resultante da diferença entre *operando_1* e *operando_2*.
- T_INTER (*operando_1,operando_2*) ⟶ retorna o subintervalo de tempo resultante da operação de intersecção entre *operando_1* e *operando_2*.
- FIRST_INSTANT (*operando_1*) ⟶ retorna o primeiro instante de *operando_1*.
- LAST_INSTANT (*operando_1*) ⟶ retorna o último instante de *operando_1*.
- FIRST_INTERVAL (*operando_1*) ⟶ retorna o primeiro intervalo do conjunto de intervalos *operando_1*.
- LAST_INTERVAL (*operando_1*) ⟶ retorna o último intervalo do conjunto de intervalos *operando_1*.

Estendemos a linguagem TOOL para incorpora em sua semântica os seguintes construtores temporais: **Max_Interval, Min_Interval, All_Interval.** A sintaxe e a semântica destes construtores são apresentadas no capítulo 6.

3.3.2 Cláusulas Temporais

Consultas temporais construídas na linguagem TOOL podem retornar valores de tempo ou objetos.

As consultas que retornam valores de tempo são construídas através das cláusulas *TWhen* e *VWhen*. AS consultas temporais que retornam dados do BD são construídas utilizando-se as cláusulas *TSlice* e *VSlice*.

Além destas cláusulas temporais, introduzimos na linguagem TOOL as cláusulas temporais: **TWhen_View** e **VWhen_View.**

A semântica de todas as cláusulas temporais é descrita no capítulo 6, que trata da implementação de consultas temporais.

Capítulo 4

A Incorporação de Tempo no SGBDOO O_2

4.1 Introdução

Um *modelo de dados* consiste em um conjunto de objetos com alguma estrutura, um conjunto de regras genéricas de integridade para estes objetos e um conjunto de operações sobre estes mesmos objetos.

Portanto, para incorporar o conceito de Tempo em um modelo de dados já existente, torna-se necessário definir: *(i)* estruturas que consigam representar a evolução temporal de objetos; *(ii)* regras que garantam a consistência temporal dos objetos, referenciadas na dissertação como *restrições de integridade temporal*; *(iii)* um conjunto de operações sobre as dimensões temporais, as *operações temporais*.

Este capítulo descreve como foi efetuada a incorporação de Tempo aos objetos gerenciados pelo SGBD O_2 no plano temporal. A incorporação de dimensões temporais no O_2 foi realizada através do desenvolvimento de uma camada externa ao sistema, que realiza o gerenciamento temporal de dados. Esta camada é denominada de **Camada de Gerenciamento Temporal.**

Este capítulo está organizado da seguinte forma: inicialmente, na seção 4.2, são descritas as principais funções e características da *Camada de Gerenciamento Temporal.* A seção 4.3 descreve a representação de Tempo utilizada para incorporar o contexto temporal no O_2. A seção 4.4 descreve as estruturas definidas para representar a evolução temporal de objetos em um BDOO. Nesta seção são apresentadas também as estratégias utilizadas para viabilizar a implementação das dimensões temporais propostas no TOODM em objetos gerenciados pelo O_2. A seção 4.5 descreve a forma como estas estruturas foram implementadas utilizando-se elementos do modelo do O_2. A seção 4.6 aborda a introdução de Meta-dados, cuja função é servir de suporte para a *Camada de Gerenciamento Temporal*

no gerenciamento temporal de dados.

4.2 A Camada de Gerenciamento Temporal

A *Camada de Gerenciamento Temporal* se comporta como uma *interface* entre o usuário e o SBDOO O_2. Sua função básica é suprir o O_2 com uma plataforma temporal. Em outras palavras, a *Camada de Gerenciamento Temporal* serve de suporte ao gerenciamento de objetos no plano temporal. Para isto, utiliza os seguintes critérios como parâmetros:

1. as especificações fornecidas pelo usuário;
2. o modelo de dados temporal *TOODM*, e;
3. o modelo de dados do O_2.

O princípio básico de funcionamento da *Camada de Gerenciamento Temporal* é o de aliar as funções básicas de gerenciamento temporal às funções de evolução de esquema e de gerenciamento de objetos apresentadas pelo O_2.

Para representar a incorporação do contexto temporal em um SGBD, o modelo *TOODM* pressupõe a existência prévia de um modelo de dados e de uma linguagem de consulta neste SGBD. Com base nesta premissa, *o mapeamento de operações temporais solicitadas pelo usuário em comandos pertencentes à sintaxe O_2 é principal tarefa da Camada de Gerenciamento Temporal.*

Este mapeamento pode gerar comandos que podem ser agrupados em três conjuntos distintos de comandos O_2. Estes conjuntos são os seguintes:

Comandos de manipulação de esquema: têm a função de definir *esquemas, bases, classes, aplicações, programas, funções* e *métodos*;

Instruções imperativas: são utilizadas para codificar *programas, métodos, funções* e *trechos de códigos.* A principal função das instruções imperativas é alterar objetos e valores;

Comandos de consulta: têm a função de recuperar objetos e valores no BD;

É importante ressaltar que no O_2, diferentemente do que ocorre na maioria dos SGBD's, estes três conjuntos de comandos são, na realidade, subconjuntos da DDL do sistema [Tec92].

Através destes mapeamentos, a *Camada de Gerenciamento Temporal* interage indiretamente com dois componentes da arquitetura do O_2: *o gerenciador de esquema* e *o gerenciador de objetos.*

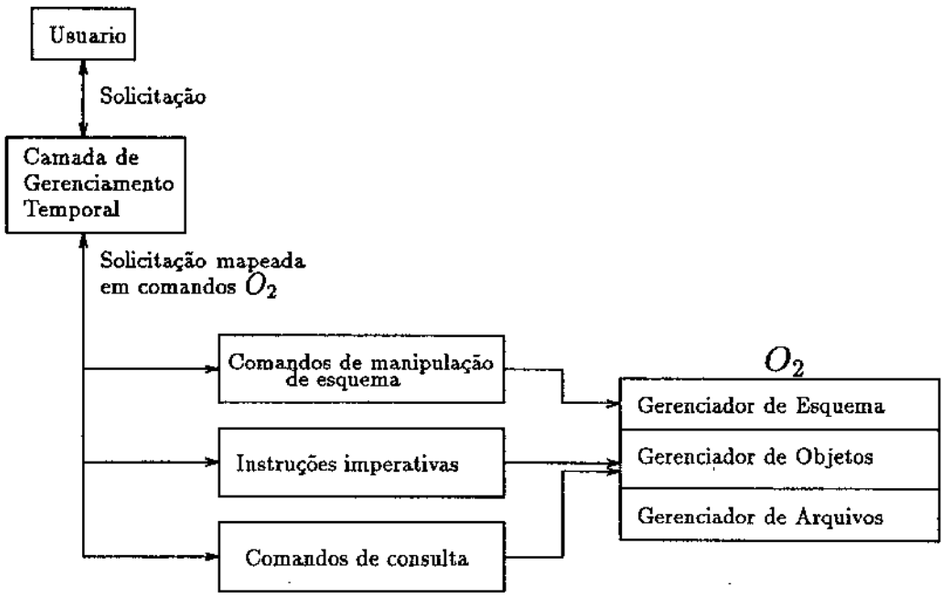

Figura 4.1: Processo de interação da *Camada de Gerenciamento Temporal* com o usuário e o O_2

A figura 4.1 ilustra o processo de interação da *Camada de Gerenciamento Temporal* com o usuário e com o O_2. A representação do O_2 foi feita através de sua arquitetura funcional (composta das três camadas especificadas na figura) [dddO91]. Vale destacar que a camada mais interna, o gerenciador de arquivos (gerenciador de dados armazenados em disco), é uma extensão do *Wisconsin Storage System* (*WiSS*).

Durante o processamento de mapeamento dos comandos do usuário em comandos O_2, a *Camada de Gerenciamento Temporal* precisa garantir:

1. a manutenção da semântica das especificações do usuário (por exemplo, garantir que, em uma consulta especificada pelo o usuário, os comandos O_2 gerados corespondam à execução da solicitação);

2. as restrições de integridade temporal do modelo TOODM (por exemplo, garantir que uma classe *Bitemporal* não possa ser declarada como superclasse de uma classe *Instantânea*);

3. a geração de comandos O_2 sintaticamente corretos.

4.3 Representação do Tempo

O capítulo 3 apresentou as características através das quais é possível representar-se o Tempo. A *Camada de Gerenciamento Temporal* utiliza um **modelo linear** e **discreto** de tempo, apresentando as duas dimensões temporais, *Tempo de Transação* e *Tempo Válido*, definidas para o TOODM.

Optou-se pelo modelo linear por questões pragmáticas, facilidade na representação e, consequentemente, na implementação. Esta opção implica, portanto, que valores de tempo ocorrerão de forma linear e ordenada, sempre no sentido *passado* → *futuro*. Este modelo de representação é semelhante ao de representação de espaço.

Apesar do conceito de tempo aproximar-se mais de uma representação contínua, optou-se pelo modelo discreto devido aos seguintes aspectos:

1. impossibilidade de medição de eventos instantâneos, pois a medição de eventos é feita com base em *chronons* que apresentam uma duração;

2. a necessidade de também se representar eventos não instantâneos, caso no qual o modelo discreto é mais adequado;

3. por último, qualquer implementação de um modelo temporal de dados sempre implicará na utilização de uma codificação discreta para tempo.

Para a definição da granularidade das marcas de tempo nos eixos temporais *Tempo-de-Transação* e *Tempo-Válido*, considerou-se a existência de dois níveis de abstração na representação de granularidade:

1. nível do usuário (nível de abstração mais alto), e;

2. nível interno da *Camada de Gerenciamento Temporal* (nível de abstração mais baixo).

Para o nível de abstração mais baixo (o nível interno da *Camada de Gerenciamento Temporal*), definiu-se que as marcas de tempo nos dois eixos temporais são representadas por uma granularidade do tipo **simples** e que utiliza a métrica **segundo**. A vantagem desta padronização de granularidade a nível interno da *Camada de Gerenciamento Temporal* é evitar a incompatibilidade de granularidades entre objetos do BD (esta incompatibilidade ocorre em operações sobre o eixo *Tempo-Válido*). Os valores de tempo neste nível de abstração serão definidos como sendo do tipo *real*, pois este tipo apresenta uma magnitude maior que o tipo *integer*.

Para representar a granularidade no nível mais alto de abstração (o nível do usuário) introduziu-se o conceito de **granularidade virtual**. Definiu-se como **granularidade virtual** a propriedade do usuário "enxergar" uma granularidade distinta da granularidade

interna da *Camada de Gerenciamento Temporal*. Em outras palavras, com base na propriedade da granularidade virtual, a *Camada de Gerenciamento Temporal* permite que o usuário manipule uma granularidade diferente da granularidade interna da *Camada de Gerenciamento Temporal*.

Para cada eixo temporal, foram definidas estratégias distintas para implementar o conceito de granularidade virtual.

O princípio básico da propriedade da granularidade virtual consiste na transformação do tipo de granularidade de uma marca de tempo. Por exemplo, considere que um eixo temporal qualquer apresenta como origem a seguinte marca de tempo: <1993,10,26,00,00,00>. Considere, ainda, a marca de tempo <3610>, que representa a quantidade de segundos após a origem do eixo. Pode-se transformar o tipo da granularidade desta marca de tempo para uma granularidade do tipo composta, formada pelos seguintes elementos: <1993,10,26,01,00,10>. Portanto, uma mesma marca de tempo é representada por diferentes tipos de granularidade.

Por definição, o usuário não exerce nenhum tipo de controle sobre a evolução temporal de objetos no eixo *Tempo-de-Transação*. Em outras palavras, ao usuário é permitido apenas consultar valores de tempo sobre este eixo temporal. Desta forma, padronizou-se que, neste eixo, a granularidade virtual (com a qual o usuário irá trabalhar) é do tipo **composta** e formada pelos seguintes elementos: **<ano, mês, dia, hora, minuto, segundo>**. Isto implica que as marcas de tempo sobre o eixo *Tempo-de-Transação* são apresentadas ao usuário através desta granularidade virtual.

Muitas vezes, o usuário deseja recuperar estados do BD especificando valores de tempo para a dimensão *Tempo de Transação* através de uma granularidade virtual que apresenta um refinamento menor. Por exemplo, o usuário pode desejar recuperar um estado do BD especificando uma marca de tempo que apresenta a seguinte granularidade virtual: **<ano,mês,dia>**. Para viabilizar este tipo de operação, foi introduzida um novo tipo de operação temporal que denominou-se de **Projeção Temporal sobre Granularidade** (PTG).

Pode-se definir uma operação de Projeção Temporal sobre Granularidade da seguinte forma:

Seja G uma granularidade do tipo composta formada pelos seguintes elementos: $< e_1, e_2, e_3, ...e_n >$ *onde* e_i *representa valores de tempo de métricas distintas (ano, mês, dia, hora, etc).*
A operação PTG $G[e_j, e_{n-5}]$, *com j<n-5, recupera os valores de tempo referentes aos elementos de posição* **j** *e* **n-5** *da granularidade G.*

Por exemplo, considere que um evento **e** é representado granularidade **<ano, mês, dia, hora, minuto, segundo>**. Considere ainda que **e** apresenta os seguintes valores

de tempo <1993,10,12,10,30,45>. A operação de PTG **G[dia, segundo]** para o evento **e** retorna os valores <12,45>. Como pode ser observado, uma operação de PTG retorna valores de tempo com uma granularidade virtual.

A implementação do conceito de granularidade virtual (no nível do usuário) para representar valores de tempo sobre o eixo *Tempo-Válido* baseou-se na seguinte estratégia: durante a especificação de atributos temporais, a *Camada de Gerenciamento Temporal* solicitará ao usuário que defina uma granularidade virtual para estes atributos. Ressalte-se que internamente a granularidade sobre a qual irá operar a *Camada de Gerenciamento Temporal*, sempre será do tipo simples, apresentado a métrica *segundo*. Por exemplo, caso o usuário defina uma granularidade como sendo o *segundo*, na realidade estará especificando uma granularidade do tipo simples. Neste caso, a granularidade virtual coincide com a granularidade real (interna). Caso defina uma granularidade que utilize a métrica *dia*, estará sendo definida, na verdade, uma granularidade do tipo composta. Neste caso a granularidade virtual é representada por uma lista composta pelos seguintes elementos: <*dia, hora, minuto, segundo*>, onde, para este exemplo, hora, minuto e segundo terão valores iguais a zero.

A introdução do conceito de granularidade virtual tem como objetivo tornar transparente para o usuário a granularidade interna da *Camada de Gerenciamento Temporal*. Por exemplo, quando o usuário define que deseja o gerenciamento temporal sobre o eixo *Tempo-Válido* de um determinado objeto através de uma granularidade virtual (diferente da granularidade interna) significa que, no mundo real, o gerenciamento da evolução temporal deste objeto é feito através desta granularidade especificada pelo usuário. Se o usuário especifica uma granularidade *dia*, então, o controle que exerce sobre a dimensão *Tempo Válido* de seus dados (no mundo real) é feito com a unidade *dia*. E esta será, na maioria da vezes, a forma como deseja que *Camada de Gerenciamento Temporal* processe suas consultas sobre dados temporais.

Marcas de tempo no eixo temporal *Tempo-de-Transação* são representadas como **eventos**. Em outras palavras, para a *Camada de Gerenciamento Temporal*, um evento no eixo *Tempo-de-Transação* representa a mudança de um estado **E1** do Banco de Dados para um estado **E2**. O Banco de Dados permanecerá neste estado **E2** até a ocorrência do próximo evento, que representará a passagem do Banco de Dados para o estado **E3**. Cada evento representa o valor do **Tempo de Máquina** correspondente ao tempo de *compromisso* (*commit*) de uma transação que ocasione a mudança de estado do BD.

Considerando que o Tempo de Máquina pode apresentar valores com uma granularidade mais refinada do que a granularidade definida para representar um evento no eixo *Tempo-de-Transação*, definiu-se que será feita a aproximação do valor de Tempo de Máquina para um valor de tempo expresso em segundos. Esta aproximação é efetuada através da eliminação dos valores que representam subunidades da métrica segundo. Assim a duração de cada *chronon* para a dimensão *Tempo de Transação* é de um segundo. Isto

significa que o lapso mínimo entre dois eventos é de um segundo.

A semântica para marcação de tempo no eixo *Tempo-de-Transação* passa a existir somente no momento em que é iniciado o gerenciamento temporal sobre este eixo (pois, a princípio, antes deste momento, o SGBD não armazenava os diversos estados do BD, armazenava apenas o último estado). Com base nesta premissa, definiu-se como origem para o eixo *Tempo-de-Transação* a marca de tempo que representa o instante da definição da *Camada de Gerenciamento Temporal* para um determinado *sistema*[1] do O_2. Portanto, para uma mesma instalação O_2, podem existir diversas origens distintas para o eixo *Tempo-de-Transação*, uma origem para cada sistema O_2.

Para representar marcas de tempo no eixo temporal *Tempo-Válido*, foi definido o formato de **intervalos**. A granularidade neste eixo utiliza a métrica **segundo**. Convencionou-se como origem deste eixo temporal a seguinte marca de tempo: **<1970,01,01,00,00,00>**. Alguns comentários devem ser feitos quanto à utilização desta marca de tempo como referencial: primeiramente, como as marcas de tempo neste eixo serão representadas como intervalos cujos limites (inferior e superior) representarão a quantidade de segundos contados a partir da origem do eixo, existirá um limite para a codificação destes valores, que corresponde à magnitude máxima apresentada pelo tipo primitivo *real* do O_2[2]; em segundo lugar, apesar de não existir o conceito físico de tempo negativo, optou-se por utilizar este artifício para viabilizar o gerenciamento temporal de objetos que apresentem valores para a dimensão *Tempo Válido* anteriores ao valor de tempo utilizado como origem para o eixo *Tempo-Válido*.

4.4 Estruturas para incorporar o Plano Temporal em um BDOO

Foram definidas estruturas de dados que viabilizassem a incorporação do contexto temporal em objetos com base em testes efetuados para verificar a completeza destas estruturas em relação aos operadores temporais propostos na linguagem de consulta do TOODM.

Como a implementação destas estruturas utiliza o sistema O_2 elas foram construídas a partir dos construtores de tipos apresentados pelo O_2.

Como estratégia de implementação, definiu-se que a inclusão das dimensões temporais em objetos deve ser feita com base em dois critérios: *tipo de construtor utilizado* e *nível de encapsulamento do objeto*.

- **Tipo de Construtor.** A associação de dimensões temporais *Tempo de Transação* e *Tempo Válido* a objetos depende do tipo de construtor utilizado na definição destes

[1] O conceito de sistema no O_2 assemelha-se ao conceito de um banco de dados, com seus esquemas e bases, e com as aplicações definidas para manipular os dados destas bases.

[2] Esta observação também vale para valores de tempo sobre o eixo *Tempo-de-Transação*.

objetos. As duas dimensões podem ser associadas a objetos definidos pelo construtor **tuple**. Para objetos definido pelo construtor **set** ou pelo construtor **list**, será associada somente a dimensão **Tempo de Transação**. A justificativa para a utilização desta estratégia é de ordem prática, pois não identificamos um único tipo de aplicação que necessitaria controlar a dimensão temporal *Tempo Válido* para conjuntos de objetos. Isto significa que o usuário no mundo real não faz este tipo de controle. Portanto, a inclusão desta dimensão para este tipo de objetos iria gerar um controle artificial e desnecessário;

- **Nível de Encapsulamento.** Para o nível mais externo de um objeto complexo, é incorporada somente uma dimensão temporal, dependendo do construtor utilizado. Por exemplo, se for utilizado o construtor **tuple** para definir o nível mais externo de um objeto, então será incorporada a dimensão *Tempo Válido* neste nível. Caso seja utilizado o construtor **set** ou **list**, a dimensão a ser incorporada será *Tempo de Transação*. Em nível de componentes de um objeto (ou seja, em nível de atributos[3] de um objeto) podem ser associadas as duas dimensões temporais, **desde que** estes componentes sejam definidos por um **tipo primitivo** do O_2 (*integer, real, char, string, boolean*), ou então, sejam definidos como um **objeto de uma outra classe** qualquer do esquema . A definição desta estratégia se deve aos seguintes fatores:

  - *(i)* Há uma sensível redução na replicação de dados, já que não se armazenará os estados de um objeto, mas sim, de seus atributos (componentes). Caso contrário, para cada alteração em um dos atributos de um determinado objeto, pois a solução canônica seria replicar todos os outros atributos com seus respectivos estados e históricos de mundo real;
  - *(ii)* A replicação de todos os outros atributos, com seus respectivos estados e históricos de mundo real, para cada alteração em um atributo de um determinado objeto, teria ainda uma outra implicação: esta replicação, na prática, representa a replicação do objeto dentro do BD. Portanto, haveria, dentro do mesmo BD, dois objetos com valores distintos, mas que deveriam possuir o mesmo *OID*, pois semanticamente representam o mesmo objeto do mundo real;
  - *(iii)* Verificou-se que era possível recuperar um estado qualquer de um objeto através de uma operação de união dos valores dos atributos deste objeto para o dado estado.

Para objetos do tipo conjunto ou lista e que apresentam recursão na sua definição (um conjunto cujos elementos são conjuntos, por exemplo), o gerenciamento temporal para a

[3]Entende-se atributo de um objeto como sendo um objeto que pertence ao grafo de composição do objeto que o encapsula.

dimensão *Tempo de Transação* só será efetuado sobre o conjunto mais externo, ou seja, o conjunto que encapsula os demais.

Evolução Temporal em um único Eixo. A representação da evolução temporal de um objeto ao longo de um único eixo, *Tempo-Válido* ou *Tempo-de-Transação*, utiliza a estrutura de **listas encadeadas**. Cada elemento da lista é uma tupla, do tipo: **<valor, valor_tempo>** onde *valor* é um ponteiro para um objeto (OID) ou um valor atômico, e *valor_tempo* é um valor de tempo no eixo temporal que está sendo representado. Este valor de tempo pode ser um número real (para representar valores no eixo *Tempo-de-Transação*) ou um intervalo (caso o eixo representado seja *Tempo-Válido*).

Evolução Temporal nos dois Eixos. Para incorporar as duas dimensões temporais simultaneamente (objetos bitemporais), efetua-se o aninhamento das duas listas encadeadas. Em nível de objeto a estrutura se mantem inalterada, pois neste nível há a incorporação de apenas uma dimensão temporal. Em nível de atributo há a seguinte alteração: para cada elemento da lista que representa a dimensão *Tempo de Transação*, é embutida a lista que representa a dimensão *Tempo Válido* no lugar atributo **valor** da tupla <valor, valor_tempo>. Com este aninhamento, a incorporação das duas dimensões temporais ocorre da seguinte forma: *cada estado de um atributo no BD é acompanhado do histórico deste atributo no mundo real.*

A seguir, é apresentada uma sequência de figuras que ilustra a incorporação do contexto temporal em um objeto através destas estruturas. Com o objetivo de facilitar o entendimento, os valores para a dimensão Tempo Válido representam a quantidade de segundos após a marca de tempo utilizada como origem para o eixo *Tempo-Válido* (<1970,01,01,00,00,00>). Os valores para a dimensão *Tempo-de-Transação* foram expressos em valores simbólicos para facilitar sua representação e para destacar a ortogonalidade dos valores de tempo nos dois eixos temporais.

Considere que os seguintes eventos ocorreram na sequência cronológica (no eixo *Tempo-de-Transação* em que são apresentados:

1. em um instante **TT1**, um objeto, o *Objeto_1*, é incluído na classe Bitemporal **C**. Este objeto começa a existir no mundo real um segundo após a marca de tempo definida como origem para o eixo *Tempo-Válido* (em outras palavras, o *Objeto_1* torna-se válido para o mundo real na seguinte marca de tempo <1970,01,01,00,00,01>). O objeto *Objeto_1* é composto pelos atributos temporais *atributo-1* (atômico) e *atributo-2* (não atômico, ou seja, é um objeto pertencente a uma outra classe qualquer), e pelo atributo implícito *histórico* (lista encadeada representando a evolução temporal do objeto no eixo *Tempo-Válido*). Neste mesmo instante **TT1** são atribuídos valores aos atributos *atributo-1* e *atributo-2*. O valor atribuído a *atributo-1* passa a ser válido no momento em que se inicia a validade do objeto *Objeto_1* no mundo

real (<1970,01,01,00,00,01>). O valor atribuído para *atributo-2* só começa a ser válido dois segundos após a origem do eixo *Tempo-Válido* (<1970,01,01,00,00,02>). O símbolo OID no histórico do *atributo-2* significa a presença de um ponteiro que aponta para o objeto definido como valor para este atributo. A representação deste contexto pode ser vista na figura 4.2;

2. em um instante **TT2**, o valor do *atributo-1* é alterado de *valor-1* para *valor-2*, cujo período de validade no mundo real inicia-se dois segundos após a origem do eixo *Tempo-Válido*. Com isso, é inserido um novo estado na lista de estados de *atributo-1*, o estado TT2. Na lista de histórico do mundo real deste atributo é inserido também um novo elemento que representa o período de validade (o intervalo [2,∞)) deste novo valor. A representação deste contexto pode ser vista na figura 4.2;

3. em um instante **TT3**, foi armazenada no BD a informação que, em um tempo válido igual a 5, Objeto_1 deixa de ser válido. Para tanto, é criado um novo estado para cada atributo temporal do objeto com o objetivo de fechar o intervalo de validade do último valor de cada atributo (por exemplo, observe que na figura **3** um novo estado é criado (TT3) para *atributo-2*, no qual o intervalo de validade do último valor deste atributo é fechado para o tempo válido igual a 5). O atributo *histórico* é atualizado para representar o final do perído de validade do objeto. Em outras palavras, o intervalo deste atributo é fechado no seu limite superior para um valor de tempo válido igual a 5. A representação deste contexto pode ser vista na figura 4.3;

4. em um instante **TT4**, é informado ao BD que Objeto_1 torna a ser válido em um tempo válido igual a 8. Com isso, é criado um novo estado para cada atributo. Na lista de histórico do mundo real de cada atributo, para este novo estado (TT4), é inserido um novo elemento. Este novo elemento é composto do último valor do atributo e pelo novo intervalo de validade do objeto, ou seja, o intervalo [8,∞). O atributo *histórico* também é atualizado para representar o novo intervalo de validade do objeto no mundo real. A representação deste contexto pode ser vista na figura 4.3;

5. em um instante **TT5**, o objeto *Objeto_1* é excluído do BD. A remoção física deste objeto implicaria na perda de toda sua evolução temporal. Portanto, para representar que *Objeto_1* foi excluído do BD, são criados dois novos estados para cada atributo temporal do objeto. É criado o estado TT5, no qual a lista de histórico do mundo real é vazia. O estado TT5 representa o estado em que o objeto foi removido do BD, portanto não necessita armazenar valores deste objeto. É criado também o estado **0** (zero), cuja função semântica é representar o fim da lista dos estados do atributo. Isto implica que este objeto não pode ser mais reinserido no BD. No

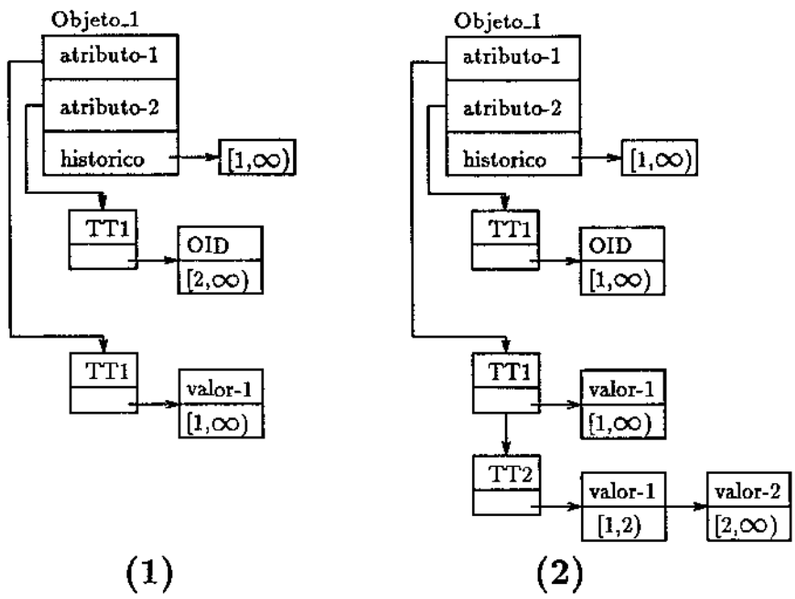

Figura 4.2: Os estados de Objeto_1 nos instantes **TT1** e **TT2** respectivamente.

entanto, como este objeto pode ser consultado (consultas feitas até o estado TT5), os seus valores podem ser reinseridos no BD na condição de um novo objeto (um novo OID). É incluído o intervalo [0,0] na lista de intervalos do atributo *histórico*. Este novo intervalo tem a função de sinalizar o fim da lista de historico do mundo real do objeto. A representação deste contexto pode ser vista na figura 4.4.

4.5 O Armazenamento do Contexto Temporal no O_2

As estruturas para armazenar a evolução temporal de objetos nos eixos *Tempo-de-Transação* e *Tempo-Válido* foram implementadas a partir dos construtores de tipos **list** e **tuple** existentes no O_2. Estes construtores foram utilizados por serem a forma mais eficiente para reduzir espaço de armazenamento e tempo de consulta, pois são otimizados internamente pelo O_2. A seguir, são apresentadas estas implementações para cada tipo de classe temporal.

Classe Bitemporal. Iniciaremos a apresentação de implementação das estruturas pelo tipo de classe Bitemporal, por ser este o mais genérico no plano temporal, já que apresenta as duas dimensões temporais possíveis no TOODM.

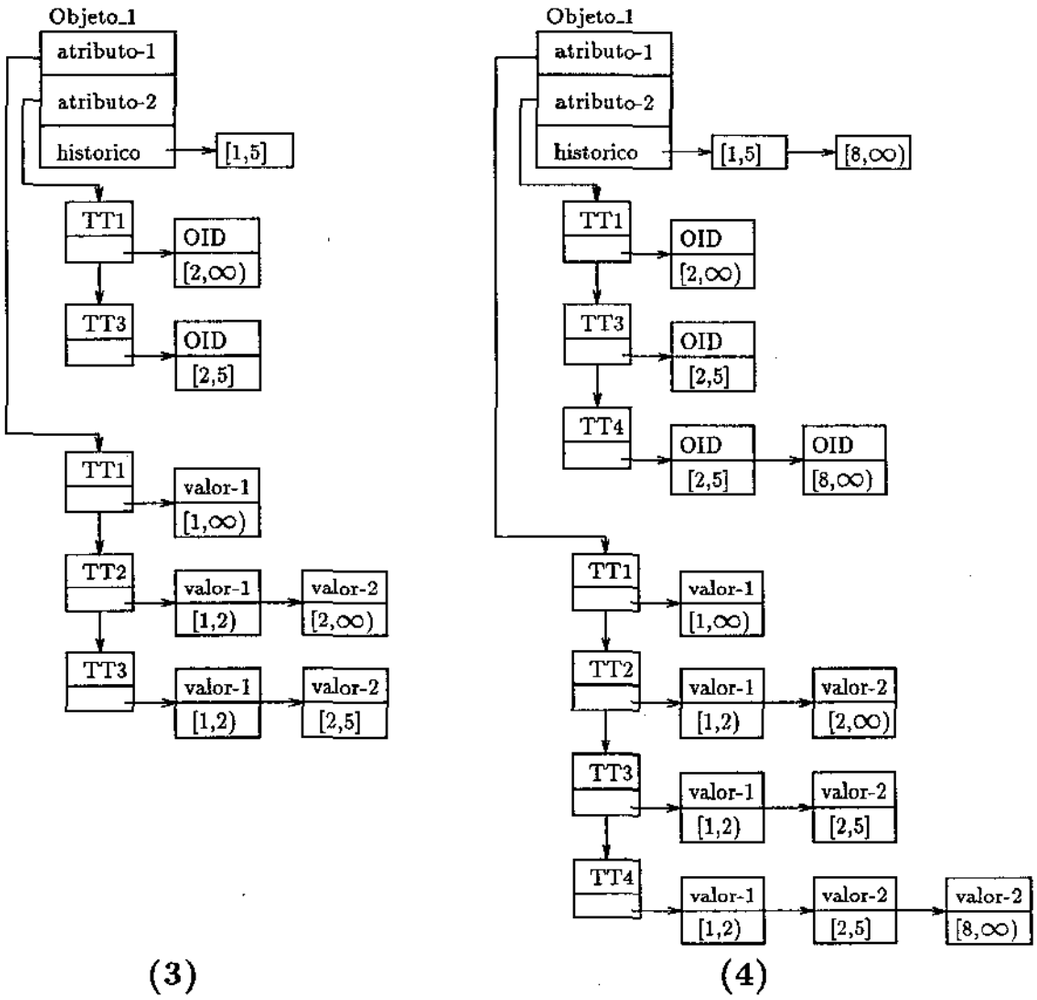

Figura 4.3: Os estados de Objeto_1 nos instantes **TT3** e **TT4** respectivamente.

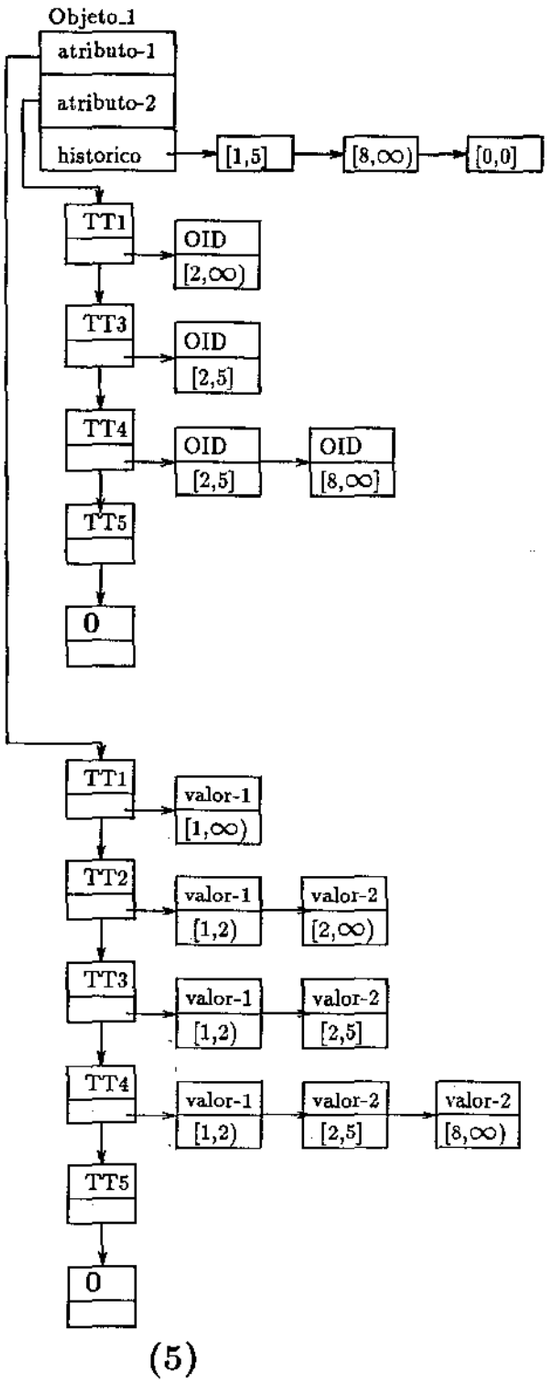

Figura 4.4: O estado de Objeto_1 no instante **TT5**.

Para implementar a estrutura que armazena o histórico de objetos do tipo **tuple**, a *Camada de Gerenciamento Temporal* especifica automaticamente o atributo <**historico_objeto** >, cuja definição é a seguinte:

historico_objeto: list(tuple(inicio_tv: real,
fim_tv : real))

Como pode ser observado, através do atributo *historico_objeto* está implementada uma lista de intervalos, onde cada intervalo (implementado como uma tupla composta de dois atributos) tem como limite inferior o atributo *inicio_tv* e como limite superior o atributo *fim_tv* . Foram definidas funções internas à *Camada de Gerenciamento Temporal* que viabilizam a transformação dos valores destes atributos (representados através da métrica segundo) em valores representados pela granularidade virtual definida pelo usuário. Caso o usuário especifique uma granularidade <*ano, mês, dia*> (granularidade do tipo composta), então estas funções podem transformar os valores de tempo expressos em segundo em valores expressos através desta granularidade virtual. O suporte para o gerenciamento da granularidade virtual é realizado através de Meta-classes, explicadas na próxima seção.

A implementação da estrutura que armazena os estados de objetos do tipo **set, unique set, list** obedece a seguinte regra: *a Camada de Gerenciamento Temporal transforma todo objeto de um dos tipos que definem conjuntos ou listas em objetos do tipo tupla.* A *Camada de Gerenciamento Temporal* gera neste caso a seguinte definição:

list(tuple(valor: <tipo-especificado-usuário>,
tt_set_list: real))

Dependendo do tipo de um atributo temporal, foram definidas diferentes estratégias para implementar a estrutura que armazenará sua evolução temporal, nos dois eixos temporais. Existem estratégias distintas para quando um atributo for definido como:

- *(i)* um tipo primitivo do O_2 (*integer, real, string, char, boolean*);
- *(ii)* um tipo complexo (atributos que correspondem a objetos complexos).

Atributos definidos pelo usuário como um dos tipos primitivo do O_2 são implementados com a seguinte definição:

nome_atributo: list(tuple(tt_nome_atributo: real,
historico_nome_atributo:
list(tuple(valor_nome_atributo: <tipo-especificado-usuário>,
inicio_tv_nome_atributo: real,
fim_tv_nome_atributo : real))))

Esta definição é gerada automaticamente pela *Camada de Gerenciamento Temporal* e, é consequentemente, transparente ao usuário. Ela permite armazenar os diversos estados (representados através do atributo *tt_nome_atributo*) de um atributo no BD, e, para cada estado, o histórico do mundo real para este mesmo atributo. Por exemplo, durante a criação de uma classe *Bitemporal*, o usuário definiu o atributo *endereço* como *temporal* (ou seja, o BD deve armazenar sua evolução nos dois eixos temporais) e do tipo *string*. Com base nestas especificações fornecidas, a *Camada de Gerenciamento Temporal* gera a seguinte definição:

```
endereco: list(tuple( tt_endereco: real,
                      historico_endereco: list(tuple(valor_endereco: string,
                                                     inicio_tv_endereco: real,
                                                     fim_tv_endereco: real))))
```

A representação da evolução temporal de atributos que apresentam como tipo uma classe do esquema é idêntica à representação da evolução temporal de atributos atômicos. A diferença está na representação do atributo *valor_nome_atributo*, o qual irá conter um OID. Este OID irá apontar para um objeto que, por sua vez, pode apresentar uma evolução temporal distinta. Por exemplo, durante a criação de uma classe *Bitemporal*, o usuário definiu o atributo *lotação* como *temporal* (ou seja, o BD deve armazenar sua evolução nos dois eixos temporais) e do tipo *Departamento* (um objeto da classe Departamento). Com base nestas especificações fornecidas, a *Camada de Gerenciamento Temporal* gera a seguinte definição:

```
lotacao: list(tuple( tt_lotacao: real,
                     historico_lotacao: list(tuple(valor_lotacao: Departamento,
                                                   inicio_tv_lotacao: real,
                                                   fim_tv_lotacao: real))))
```

Para implementar o contexto temporal em atributos complexos definidos através de construtores de tipos (*tuple, set, list*), segue-se o mesmo princípio utilizado na implementação do contexto temporal no nível (de encapsulamento) mais externo de objetos.

Classe Tempo-Válido. Este tipo temporal de classe só apresenta a dimensão temporal *Tempo-Válido*. Sua implementação é semelhante à das classes Bitemporais, com a remoção da estrutura que armazena a dimensão *Tempo de Transação*. Desta forma, para cada atributo temporal deste tipo de classe, a *Camada de Gerenciamento Temporal* gera a seguinte definição:

```
nome_atributo: list(tuple (valor_nome_atributo: <tipo-especificado-usuário>,
                           inicio_tv_nome_atributo: real,
                           fim_tv_nome_atributo : real))
```

Classe Tempo-de-Transação. O mesmo princípio utilizado na implementação do contexto temporal em uma classe *Tempo-Válido* pode ser utilizado para a classe *Tempo-de-Transação*. Portanto, é necessária apenas a remoção das listas que armazenam a dimensão *Tempo Válido*. Um atributo definido como temporal em uma classe deste tipo apresenta a seguinte especificação:

nome_atributo: list (tuple (tt_nome_atributo: real,
valor_nome_atributo: <tipo-especificado-usuário>))

A *Camada de Gerenciamento Temporal* viabiliza a incorporação das duas dimensões temporais em qualquer definição de tipo permitido pelo modelo do O_2. Denominamos esta propriedade de **ortogonalidade temporal de tipos.**

Exemplos. A seguir serão apresentados alguns exemplos que ilustram as implementações no O_2 das estruturas para representar a evolução temporal de objetos em um BDOO. Com o intuito de tornar os exemplos mais genéricos, considere que as classes apresentadas serão do tipo temporal *Bitemporal*. Para visualizar a implementação destas classes como sendo do tipo *Tempo-de-Transação* ou *Tempo-Válido*, bastaria apenas remover a estrutura correspondente à dimensão temporal não presente para o tipo temporal de classe, como já descrito. Em cada exemplo, será apresentada uma classe Instantânea e sua contrapartida Bitemporal.

Exemplo 1:
→Definição atemporal

```
class Pessoa
    type tuple ( nome: string,
                 endereco: string,
                 data-nascimento: Date,
                 cpf: integer)
```

→Definição temporal [4]

```
class Pessoa
type tuple ( nome: list(tuple( tt_nome: real,
                               historico_nome: list(tuple( valor_nome: string,
                                                           inicio_tv_nome: real,
                                                           fim_tv_nome: real)))),
             endereco: list(tuple( tt_endereco: real,
```

[4]Considere que foram definidos como temporais os atributos <*nome*> e <*endereco*>.

```
                                historico_endereco: list(tuple( valor_endereco: string,
                                                                inicio_tv_endereco: real,
                                                                fim_tv_endereco: real)))),
        data-nascimento: Date,
        cpf: integer,
        historico_objeto: list(tuple( inicio_tv: real,
                                      fim_tv: real)))
```

Exemplo 2:

→ Definição atemporal

```
class Funcionario inherit Pessoa
    type tuple ( salario: tuple( salario-base: integer,
                                 comissao: integer),
                 lotacao: Departamentos-empresa,
                 dependentes: set(Pessoa),
                 data-admissao: Date)
```

→ Definição temporal [5]

```
class Funcionario inherit Pessoa
type tuple ( salario: tuple( salario-base: list(tuple( tt_salario-base: real,
                                                       historico_salario-base:
                                                       list(tuple( valor_salario-base: integer,
                                                                   inicio_tv_salario-base: real,
                                                                   fim_tv_salario-base: real)),
                             comissao: list(tuple( tt_comissao: real,
                                                   historico_comissao:
                                                   list(tuple( valor_comissao: integer,
                                                               inicio_tv_comissao: real,
                                                               fim_tv_comissao: real)))))),
                             historico_objeto: list(tuple( inicio_tv: real,
                                                           fim_tv: real))),
        lotacao:list(tuple(tt_lotacao:real,
                           historico_lotacao: list(tuple( valor_lotacao:
                                                          Departamentos-empresa,
                                                          inicio_tv_lotacao:real,
```

[5] Considere que foram definidos como temporais os atributos <*salario*>, <*lotacao*> e <*dependentes*>.

```
                                                    fim_tv_lotacao:real)))) ,
    dependentes:list(tuple(valor_dependentes:set(list(tuple( tt_Pessoa:real,
                        historico_Pessoa:
                        list(tuple( valor_Pessoa:Pessoa,
                                    inicio_tv_Pessoa:real,
                                    fim_tv_Pessoa:real))))) ,
                        tt_set_list:real)),
    dependentes: list(tuple( valor_dependentes: set(Pessoa),
                             tt_set_list: real)),
    Data-admissao: Date,
    historico_objeto: list(tuple( inicio_tv: real,
                                  fim_tv: real)))
```

Exemplo 3:

→Definição atemporal

```
class Departamentos-empresa
    type set(tuple ( nome: string,
                     localizacao: tuple( endereco: string,
                                         cidade: string),
                     administrador: Funcionario,
                     lotacao: set(Funcionario)))
```

→ Definição temporal [6]

```
class Departamentos-empresa
    type tuple( valor: set(tuple( nome: list(tuple( tt_nome: real,
                                                    historico_nome:
                                                    list(tuple( valor_nome: string,
                                                                inicio_tv_nome: real,
                                                                fim_tv_nome: real)))),
                                  localizacao: tuple( endereco: string,
                                                      cidade: string),
                                  administrador:list(tuple( tt_administrador:real,
                                                            historico_administrador:
                                                            list( tuple
                                                                (valor_administrador:
                                                                Funcionario,
```

[6] Considere que foram definidos como temporais os atributos <*nome*>, <*administrador*> e <*lotacao*>.

```
                                                    inicio_tv_administrador:
                                                    real,
                                                    fim_tv_administrador:
                                                    real))))) ,
                                lotacao:list(tuple( valor_lotacao:
                                                set(list(tuple
                                                        ( tt_Funcionario:real,
                                                        historico_Funcionario:
                                                        list(tuple (valor_Funcionario:
                                                                Funcionario,
                                                                inicio_tv_Funcionario:
                                                                real,
                                                                fim_tv_Funcionario:
                                                                real))))) ,
                                tt_set_list:real)),
                                historico_objeto: list(tuple( inicio_tv: real,
                                fim_tv: real)))
                        tt_set_list: real)
```

No exemplo 2 é apresentada uma situação que merece destaque. Este exemplo refere-se à representação da evolução temporal de objetos em grafos de composição. Este fato está caracterizado na definição do atributo *lotacao*. Este atributo foi definido como temporal e do tipo *Departamentos-empresa*. Isto significa que será atribuído como valor ao atributo *lotacao* um objeto pertencente à classe *Departamentos-empresa*. Como pode ser observado na definição temporal deste atributo, foram incorporadas as duas dimensões temporais previstas no modelo. Isto se justifica pelo fato de que o atributo *lotacao* apresentar uma evolução temporal independente da evolução temporal do objeto que o compõe. Por exemplo, um funcionário pode ser lotado em distintos departamentos ao longo de sua permanência na empresa. Em outras palavras, existe uma evolução temporal deste atributo ao longo permanência do funcionário na empresa independente da evolução temporal do objeto departamento.

No último exemplo é apresentada uma outra situação que também merece destaque. O atributo *localização* é definido como *tuple* (um tipo estruturado do O_2), portanto é composto por dois outros atributos (veja definição no exemplo). Observe que a *Camada de Gerenciamento Temporal* proposta garante a manutenção da consistência temporal, pois como o atributo <*localizacao*> foi definido como *instantâneo*. Neste caso os atributos <*localizacao.endereco*> e <*localizacao.cidade*> não poderiam ser temporais. Este tipo de controle é gerenciado pela *Camada de Gerenciamento Temporal*.

4.6 A Introdução de Meta-Dados no Gerenciamento Temporal de Dados

Para maior eficiência no gerenciamento de dados no plano temporal introduziu-se o conceito **Meta-classe**, cuja função é servir de *suporte lógico* para este gerenciamento.

No momento da definição da *Camada de Gerenciamento Temporal* para um *sistema* O_2, são criadas duas Meta-classes[7] com seus respectivos métodos, **Meta_classe_esquemas** e **Meta_classe_classes**. Os objetos destas Meta-classes são visíveis somente para a *Camada de Gerenciamento Temporal*, garantindo-se, assim, a segurança dos meta-dados.

Os objetos pertencentes à extensão da Meta-classe *Meta_classe_esquemas* contêm os meta-dados relativos aos esquemas definidos para um determinado *sistema* O_2. A principal função desta Meta-classe é gerenciar a evolução de esquemas. Funcionalidades específicas a esta Meta-classe são:

1. Garantir restrições de integridade do modelo de dados do O_2 durante a evolução de esquemas (criação e manipulação de esquemas, bases e classes). Por exemplo, garantir a unicidade de uma classe em um esquema;

2. Garantir a integridade da lista de superclasses de uma classe durante a manipulação desta classe. Por exemplo, garantir que não sejam incluídas classes, que não estão definidas no esquema, na lista de superclasses de uma determinada classe.

A seguir, são apresentadas as especificações da Meta-classe *Meta_classe_esquemas* e de seus respectivos métodos e repositório.

class Meta_classe_esquemas public type tuple (nome_esquema:string,
bases_esquema:list(string),
classes_esquema:list(Meta_classe_classes),
sequencial[8] *:integer)*

method public altera_esquema(nome_esquema:string, sequencial:integer),
public retorna_sequencial : integer,
public delete_esquema,
public inclui_base (nome_base:string),
public altera_nome_base (nome_base:string, novo_nome:string),
public remove_base (nome_base:string),
public retorna_base (nome_base:string) : string,
public inclui_classe (classe:Meta_classe_classes),

[7] O processo de implementação destas Meta-classes é descrito no capítulo que trata da implementação da *Camada de Gerenciamento temporal.*

public retorna_classe (nome_classe:string) : Meta_classe_classes,
public verifica_existencia_classe (nome_classe:string) : boolean,
public remove_classe (nome_classe:string)
end;
Repositorio_Esquemas: set(Meta_classe_esquemas);

Os objetos pertencentes à extensão da Meta-classe *Meta_classe_classes* contêm os metadados relativos às classes e seus respectivos métodos definidas para um determinado esquema. Esta Meta-classe apresenta as seguintes funcionalidades específicas:

- Garantir a consistência temporal nos grafos de herança existentes em um esquema, ou seja, garantir a consistência temporal entre uma classe e suas superclasses. Por exemplo, se uma classe possui pelo menos uma superclasse *Bitemporal*, então esta classe não pode ser definida como *Instantânea*;
- Gerenciar a granularidade dos *time-stamps* de cada atributo dos objetos de uma classe;
- Servir de suporte para as operações temporais sobre objetos (manipulação e consulta);

A estrutura da Meta-classe *Meta_classe_classes* e os métodos que podem ser aplicados aos seus objetos apresentam a seguinte definição:

```
class Meta_classe_classes public type tuple (nome_classe: string,
                                              tipo_temporal: string,
                                              lista_atributos:list(tuple(nome_atributo:string,
                                                                         tipo: string,
                                                                         temporal: boolean,
                                                                         granularidade:
                                                                         list(string))),
                                              lista_metodos:
                                              list(tuple(nome_metodo: string,
                                                         parametros:
                                                         list(tuple(nome_parametro: string,
                                                                    tipo_parametro: string)))),
                                              lista_superclasses: unique set (string))
```

[8] A função do atributo <*sequencial*> é abordado no capítulo seguinte.

method public retorna_nome_classe : string,
public altera_nome_classe (nome_classe:string)
end;
name Repositorio_Classes: set(Meta_classe_classes);

Um outra função que estas *Meta-classes* também poderiam possuir era a de armazenar os diversos estados de um esquema durante sua evolução[9]. Para isto, seria necessário apenas inserir o eixo *Tempo-de-Transação* nas duas Meta-classes.

[9]Pode-se definir o conceito de evolução de esquema como a mudança do esquema de um estado X para um novo estado Y.

Capítulo 5

DDL Temporal Virtual para o O_2

5.1 Introdução

Até agora foram descritos a modelagem, a representação do conceito de Tempo e *o que* a *Camada de Gerenciamento Temporal* proposta deve implementar para incorporar este conceito em um BDOO, em particular no O_2. Nesta capítulo é descrito *como* implementar este conceito de Tempo, ou seja, será apresentado *como* a *Camada de Gerenciamento Temporal* cria objetos temporais a partir da DDL do O_2, garantindo-se a consistência temporal destes objetos com o TOODM e a consistência com o modelo de dados do O_2.

A função básica da *Camada de Gerenciamento Temporal* é realizar o mapeamento dos comandos especificados pelo usuário em comandos básicos da DDL do O_2, garantindo-se a manutenção da semântica das especificações fornecidas e as restrições de integridade temporal do modelo.

Esta camada é a única interface permitida para a interação entre o usuário e o BDT. Estas premissa é de fundamental importância para garantir a consistência e integridade dos dados e meta-dados administrados pela camada. Esta permite ao usuário *consultar, criar, alterar* ou *remover* qualquer componente de um esquema do O_2. A evolução de esquemas é permitida se não houver dados carregados.

Este capítulo está organizado da seguinte forma: a seção 5.2 apresenta uma visão geral sobre o funcionamento e a implementação da *Camada de Gerenciamento Temporal.* Nas seções seguintes, é descrito o processo de mapeamento. A seção 5.3 apresenta como a *Camada de Gerenciamento Temporal* implementa a criação e a manipulação de esquemas e bases a partir de solicitações do usuário. A seção 5.4 descreve a implementação e a manipulação de classes.

5.2 O Funcionamento da *Camada de Gerenciamento Temporal*

Em função da organização lógica imposta pelo O_2 (sistemas, esquemas e bases) e em função da impossibilidade de se construir um código interno ao O_2, definiu-se a seguinte estratégia de implementação: durante o processo de definição da *Camada de Gerenciamento Temporal* em um *sistema* O_2, são criados um esquema e uma base com os quais a *Camada de Gerenciamento Temporal* irá trabalhar. Neste instante, também são definidas as Meta-classes (descritas no capítulo anterior) com suas respectivas extensões, para este mesmo esquema e mesma base. Por convenção, este esquema é denominado de **E_BDT** e a base de **B_BDT**. A definição da *Camada de Gerenciamento Temporal* é efetuada pela execução de um *include file* (um arquivo *Unix* que contem todos os comandos necessários à definição da *Camada de Gerenciamento Temporal*) em uma *shell* O_2.

A versão do O_2 disponível no DCC é mono-usuário. Assim, decidiu-se implementar a interação do usuário com a camada da seguinte forma. A camada transforma os comandos do usuário em comandos DDL do O_2 e os grava em um arquivo *Unix* intermediário. Este arquivo é a seguir executado em uma *shell* do O_2 sob a forma de *include files.*

Estes arquivos *Unix* intermediários também armazenam códigos do tipo:

$$run\ body\ \{\ <código\ O_2C>\ \}\ .$$

A função deste tipo de código é a de manipular as Meta-classes.

Em resumo, cada arquivo *Unix* é composto de dois grupos de comandos:

- Comandos de manipulação de esquema cuja função é atualizar esquemas O_2, e;
- Comandos imperativos cuja função é atualizar as Meta-classes.

Convencionou-se que a definição de nomes para estes arquivos *Unix* é feita com base nas seguintes regras:

1. Para cada esquema existirá um arquivo que armazenará somente os comandos de criação deste esquema. O nome deste arquivo é construído da seguinte forma: **<nome-esquema>.o2**;
2. Todos os comandos subsequentes de manipulação deste esquema são armazenados em arquivos cujos nome apresentam a seguinte regra de formação:
   <nome-do-esquema>_<sequencial>.o2.
   O componente **sequencial** é obtido através do atributo **sequencial** da Meta-classe *Meta-classe-esquemas* (ver definição no capítulo 4, seção 4.6) e tem como valor inicial o inteiro **0** (zero).

Como pode ser observado, para cada esquema criado através da *Camada de Gerenciamento Temporal* há um conjunto de arquivos que conterão todos os comandos DDL responsáveis pela evolução deste esquema. Isto implica que estes arquivos comportam-se também como um *Log*, que apresenta toda a evolução de um dado esquema O_2.

Uma das diretrizes traçadas para implementar a *Camada de Gerenciamento Temporal* foi a de manter as propriedades do O_2, não limitando os recursos apresentados por este SBDOO, inclusive a *recursividade* na definição de tipos complexos.

As Meta-classes só são atualizadas após a efetivação das transações que modificam o esquema. Isto tem por objetivo evitar inconsistências entre as informações contidas nas Meta-classes e os estados de esquemas bases e classes.

5.3 Esquemas e Bases

Através da *Camada de Gerenciamento Temporal* o usuário pode definir, alterar e remover esquemas e bases[1], informando apenas nomes destes elementos. A partir da definição de um esquema, a *Camada de Gerenciamento Temporal* inicia o gerenciamento dos metadados deste esquema e de seus componentes (classes e bases).

→*Exemplo.* Criar o esquema *e_teste* e uma base pertencente ao esquema, a *base_teste.* A *Camada de Gerenciamento Temporal* irá gerar o arquivo **e_teste.o2** com o seguinte conteúdo.

```
set base B_BDT;
run body
{
  o2 Meta_classe_esquemas esquema=new Meta_classe_esquemas;
  esquema→nome_esquema="e_teste";
  Repositorio_Esquemas+=set(esquema);
  esquema→bases_esquema+=list("base_teste");
};
create schema e_teste;
create base base_teste;
commit;
set base B_BDT;
```

→*Exemplo.* Alterar o nome da base *base_teste* para *b_teste.* Será gerado o arquivo **e_teste_0.o2** com o seguinte conteúdo.

[1]No caso de bases, há, ainda, a opção de copiar bases.

```
set schema e_teste;
rename base base_teste as b_teste;
set base B_BDT;
run body
{
  o2 Meta_classe_esquemas esquema;
  esquema=trata_esquema("e_teste");
  esquema→altera_nome_base("base_teste","b_teste");
};
commit;
```

Observe que o *include file*, que contem as especificações para a criação do esquema *e_teste*, terá que, necessariamente, ser executado antes de qualquer comando que manipule este esquema, conforme afimado anteriormente. Isto implica em encerrar a sessão da *Camada de Gerenciamento Temporal*, executar o *include file e_teste.o2* e, em seguida, ativar uma nova sessão da *Camada de Gerenciamento Temporal*. A nova ativação será responsável pela geração do *include file e_teste_0.o2*. Este fluxo de processamento é consequência do fato de que a versão do sistema O_2 com que trabalhou-se é *mono-usuário*.

5.4 Classes

A *Camada de Gerenciamento Temporal* permite criar, alterar definição, remover e alterar nomes de classes temporais ou instantâneas. As opções de remover uma classe ou alterar o nome desta apresentam implementações que não trazem implicações no gerenciamento do contexto temporal.

Caso a *Camada de Gerenciamento Temporal* incorporasse também o contexto temporal nas Meta-classes com a função de gerenciar a evolução temporal de esquema, a remoção de uma classe apresentaria, neste caso, uma implementação que teria implicações no gerenciamento do contexto temporal. Uma possível implementação poderia ser a seguinte: a remoção de uma classe não implicaria simplesmente na remoção física de seus objetos, e sim, na criação de um novo estado para o objeto que representa classe a ser removida na Meta-classe *Meta_classe_Classes*. Este estado apresentaria como *Tempo de Transação* um valor que indicaria que a classe havia sido removida do esquema, não podendo mais ser manipulada ou consultada.

As opções de criação e alteração de uma classe apresentam a mesma funcionalidade: *criar uma classe com base nas especificações fornecidas pelo usuário* (esta característica foi herdada do próprio O_2). Isto significa que o algoritmo implementado para estas duas opções é o mesmo. Como este algoritmo é a base da incorporação do contexto temporal no O_2, iremos, a seguir, descrevê-lo:

Passo 1: Gerar o nome da classe fornecido pelo usuário no formato convencionado pelo O_2, ou seja, primeira letra maiúscula e o restante do nome em letras minúsculas;

Passo 2: Verificar se a opção do usuário está consistente com os meta-dados. Por exemplo, se o usuário solicitar criar uma classe já existente, ou então, modificar uma classe não definida, será enviada ao usuário uma mensagem de erro;

Passo 3: Solicitar o tipo temporal da classe, se *Instantânea*, *Tempo-de-Transação*, *Tempo-Válido* ou *Bitemporal*;

Passo 4: Solicitar lista de superclasses (propriedade de *herança múltipla*). Para cada superclasse, verificar sua existência no esquema e se há conflitos entre o tipo temporal da classe e o de sua superclasse. Caso a superclasse não tenha sido ainda definida, ou então, haja conflitos entre os tipos temporais, será enviada uma mensagem de erro ao usuário (a classe não será criada enquanto não forem retificados os erros indicados);

Passo 5: Solicitar o tipo O_2 (*tuple, list, set, unique set*) para a classe. São ativadas funções específicas para o **tuple** e para os tipos **list** e **set**;

Passo 6: Gerar arquivo *Unix* contendo os comandos de definição de classe e os códigos O_2C para atualizar as Meta-classes.

Os conflitos entre os tipos temporais em um grafo de herança (ver Passo 4) originam-se da premissa de que uma classe herda as propriedades de suas superclasses [Mal89], [Ban88], [Vos91b]. O contexto temporal é inserido em uma classe através de atributos que representam as dimensões temporais. Portanto, pela propriedade de herança, uma classe também herda as dimensões temporais de suas superclasses.

Pode-se afirmar, portanto, que uma classe terá que apresentar, no mínimo, as dimensões temporais de suas superclasses. Isto poderá gerar conflitos (citados no Passo 4) durante a definição de classes. Por exemplo, suponha que um usuário definiu uma classe **C** como *Instantânea* e incluiu na lista de superclasses uma classe do tipo *Bitemporal*, pela propriedade de herança, a classe **C** apresentaria as duas dimensões temporais para os atributos definidos na superclasse, gerando, desta forma, uma inconsistência com o mundo real (uma classe *Instantânea* apresentando propriedades temporais). Como pode ser observado na descrição do *Passo 4*, optou-se por evitar estes conflitos, ao invés de propor regras para resolvê-los.

Para garantir a recursividade na definição de tipos complexos (conjuntos de conjuntos ou tuplas de tuplas, por exemplo), as funções para processar estes tipos apresentam a propriedade da recursividade.

A seguir serão apresentados exemplos de arquivos *Unix* gerados pela *Camada de Gerenciamento Temporal*. Serão utilizadas as definições de classes apresentadas no capítulo an-

terior.

→*Exemplo.* Definir a classe *Pessoa*, do tipo *Bitemporal*, para o esquema *e_teste*. Definir como extensão desta classe o repositório de dados *Repo_Pessoa*. Os atributos da classe *Pessoa* são fornecidos pelo usuário.

```
set schema e_teste;
create class Pessoa public
create class Pessoa public
type tuple ( nome: list(tuple( tt_nome: real,
                               historico_nome: list(tuple( valor_nome: string,
                                                           inicio_tv_nome: real,
                                                           fim_tv_nome: real)))),
             endereco: list(tuple( tt_endereco: integer,
                                   historico_endereco: list(tuple( valor_endereco: string,
                                                                   inicio_tv_endereco: real,
                                                                   fim_tv_endereco: real)))),
             data-nascimento: Date,
             cpf: integer,
             historico_objeto: list(tuple( inicio_tv: real,
                                           fim_tv: real)))
end;
name Repo_Pessoa: set(Pessoa);
export schema class Pessoa;
set base B_BDT;
run body
{
  o2 Meta_classe_esquemas esquema;
/* - Inicialização do novo objeto (a classe Pessoa) na classe Meta_classe_classes - */
  o2 Meta_classe_classes classe=new Meta_classe_classes;
/* ---------------------------------------------------------- */
  o2 tuple(nome_atributo:string, tipo:string, temporal:boolean,
           granularidade:list(string)) tupla_aux;
/* -ativação da função que retorna um objeto (o esquema da classe Pessoa) -
/* -da classe Meta_classe_esquema - */
  esquema=trata_esquema("e_teste");
/* ---------------------------------------- */
/* -Inicialização dos atributos que compõem o novo objeto a ser incluído- */
  classe→nome_classe="Pessoa";
  classe→tipo_temporal="Bitemporal";
```

```
/* - Definição das propriedades do atributo < nome > - */
  tupla_aux.nome_atributo= "nome";
  tupla_aux.tipo= "string";
  tupla_aux.temporal= true;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
/* - Definição das propriedades do atributo < endereco > - */
  tupla_aux.nome_atributo= "endereco";
  tupla_aux.tipo= "string";
  tupla_aux.temporal= true;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
/* - Definição das propriedades do atributo < data − nascimento > -*/
  tupla_aux.nome_atributo= "data-nascimento";
  tupla_aux.tipo= "Date";
  tupla_aux.temporal= false;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
/* - Definição das propriedades do atributo < CPF > - */
  tupla_aux.nome_atributo= "cpf";
  tupla_aux.tipo= "integer";
  tupla_aux.temporal= false;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
/* - Inicialização do conjunto de superclasses da classe Pessoa - */
  classe→lista_superclasses= unique set();
/* ---------------------------------------- */
/* Inclusão da classe Pessoa no conjunto de classes que compõem o esquema e_teste */
  esquema→inclui_classe(classe);
/* ---------------------------------------- */
/* - tornando o novo objeto (a classe Pessoa) persistente- */
  Repositorio_Classes+=set(classe);
};
import schema e_teste class Pessoa;
commit;
#"efetua_consulta";
```

Para os dois atributos temporais (<*nome*> e <*endereco*>) da classe *Pessoa*, foi especificada uma granularidade do tipo composta, cujos componentes são: <*ano,mês,dia*>. O

atributo <*data-nascimento*> foi especificado como um objeto pertencente à classe *Date* (classe interna do O_2, cuja função é representar objetos do tipo *data*). Este é um exemplo de um atributo temporal definido no eixo *Tempo-de-Usuário*. Apesar de ser um atributo relativo a Tempo, sua semântica temporal será gerenciada pelo próprio usuário. Considerando que a classe *Pessoa* está sendo especificada logo após os comandos de definição do esquema *e_teste* e de alteração do nome da base *base_teste* para *b_teste*, o arquivo *Unix* a ser gerado será o **e_teste_01**.

A execução do comando **export schema class** tem como objetivo viabilizar a execução do comando **import base class** dentro do esquema **E_BDT** (esquema para o qual foi definida a *Camada de Gerenciamento Temporal*). A execução destes dois comandos capacita a *Camada de Gerenciamento Temporal* executar operações de consultas sobre objetos armazenados em bases de dados diferentes da base **B_BDT** (base para a qual foi definida a *Camada de Gerenciamento Temporal*). Em outras palavras, a execução destes dois comandos (export e import) tem como objetivo tornar visível para a *Camada de Gerenciamento Temporal* objetos armazenados em outras bases do sistema.

→*Exemplo.* Definir a classe *Funcionario* do tipo *Bitemporal*. Esta classe deverá pertencer ao grafo de herança da classe *Pessoa*. Definir como extensão desta classe o repositório de dados *Repo_Funcionario*:

```
set schema e_teste;
create class Funcionario inherit Pessoa public
        type tuple(salario:tuple(salario_base:
                        list(tuple(tt_salario_base:real,
                                historico_salario_base:
                                list(tuple(valor_salario_base:integer,
                                        inicio_tv_salario_base:real,
                                        fim_tv_salario_base:real)))),
                        comissao:
                        list(tuple(tt_comissao:real,
                        historico_comissao:
                        list(tuple(valor_comissao:integer,
                                inicio_tv_comissao:real,
                                fim_tv_comissao:real)))),
                        historico_objeto: list(tuple(inicio_tv:real,
                                                fim_tv:real))),
                lotacao: list(tuple(tt_lotacao:real,
                                historico_lotacao:
                                list(tuple(valor_lotacao:Departamentos_empresa,
                                        inicio_tv_lotacao:real,
```

```
                                                fim_tv_lotacao:real)))),
                        dependentes:list(tuple(valor_dependentes:
                                                set(list(tuple(tt_Pessoa:real,
                                                        historico_Pessoa:
                                                        list(tuple(valor_Pessoa:Pessoa,
                                                                inicio_tv_Pessoa:real,
                                                                fim_tv_Pessoa:real))))) ,
                                                tt_set_list:real)),
                        data_admissao: Date,
                        historico_objeto: list(tuple(inicio_tv:real,
                                                fim_tv:real)))
end;
name Repo_Funcionario: set(Funcionario);
export schema class Funcionario;
set base B_BDT;
run body
{
  o2 Meta_classe_esquemas esquema;
  o2 Meta_classe_classes classe=new Meta_classe_classes;
  o2 tuple(nome_atributo:string, tipo:string, temporal:boolean,
          granularidade:list(string)) tupla_aux;
  esquema=trata_esquema("e_teste");
  classe→nome_classe="Funcionario";
  classe→tipo_temporal="Bitemporal";
  tupla_aux.nome_atributo= "salario";
  tupla_aux.tipo= "tuple";
  tupla_aux.temporal= true;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
  tupla_aux.nome_atributo= "salario_base";
  tupla_aux.tipo= "integer";
  tupla_aux.temporal= true;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
  tupla_aux.nome_atributo= "comissao";
  tupla_aux.tipo= "integer";
  tupla_aux.temporal= true;
  tupla_aux.granularidade= list("dia","mes", "ano");
  classe→lista_atributos+=list(tupla_aux);
```

```
tupla_aux.nome_atributo= "lotacao";
tupla_aux.tipo= "Departamentos_empresa";
tupla_aux.temporal= true;
tupla_aux.granularidade= list("dia", "mes", "ano");
classe→lista_atributos+=list(tupla_aux);
tupla_aux.nome_atributo= "dependentes";
tupla_aux.tipo= "set";
tupla_aux.temporal= true;
tupla_aux.granularidade= list("dia", "mes", "ano");
classe→lista_atributos+=list(tupla_aux);
tupla_aux.nome_atributo= "";
tupla_aux.tipo= "Pessoa";
tupla_aux.temporal= true;
tupla_aux.granularidade= list();
classe→lista_atributos+=list(tupla_aux);
tupla_aux.nome_atributo= "data-admissao";
tupla_aux.tipo= "Date";
tupla_aux.temporal= false;
tupla_aux.granularidade= list("dia", "mes", "ano");
classe→lista_atributos+=list(tupla_aux);
classe→lista_superclasses= unique set("Pessoa");
esquema→inclui_classe(classe);
Repositorio_Classes+=set(classe);
};
import schema e_teste class Funcionario commit;
#"efetua_consulta.o2"
```

Os atributos temporais desta classe, <*salario*>, <*lotacao*> e <*dependentes*>, são compostos, ou seja, são definidos por construtores de tipos do O_2. Através da utilização da propriedade de recursividade, a *Camada de Gerenciamento Temporal* incorpora o contexto temporal nestes atributo de forma idêntica à incorporação em nível de objetos. Para o atributo <*salario*>, definido como uma tupla, o gerenciamento temporal é realizado sobre a tupla (como se esta tupla se comportasse como um objeto) e sobre os atributos desta tupla, <*salario.salario_base*> e <*salario.comissao*>. O mesmo princípio se aplica para os atributos <*lotacao*> e <*dependentes*>. Por exemplo, no caso específico do atributo <*dependentes*>, o gerenciamento temporal será efetuado de duas maneiras distintas:

1. sobre o conjunto de objetos do tipo *Pessoa* (serã armazenados os vários estados deste conjunto ao longo do eixo *Tempo-de-Transação*), e;

2. sobre cada elemento deste conjunto (neste caso, a evolução temporal será representada nos dois eixos temporais).

Capítulo 6

Consultas Temporais

6.1 Introdução

A partir do conjunto de operadores e construtores temporais propostos na linguagem **TOOL** [Lin93], foi desenvolvido um **Gerenciador de Consultas** como parte do trabalho de dissertação. Este gerenciador caracteriza-se por construir consultas com base em especificações fornecidas pelo usuário. O princípio de funcionamento deste gerenciador assemelha-se ao de gerenciadores como *QMF* e *QBE*.

O Gerenciador de Consultas constrói consultas que devem ser executadas sobre as estruturas de armazenamento definidas pela *Camada de Gerenciamento Temporal*. Portanto, funcionalmente, o *Gerenciador de Consultas* é um módulo da *Camada de Gerenciamento Temporal*.

O *Gerenciador de Consultas* implementado mapeia consultas temporais da linguagem **TOOL** em consultas que possam ser processadas pelo O_2query (processador de consultas apresentado pelo O_2).

As seguintes propriedades básicas estão presentes no *Gerenciador de Consultas*:

- Geração de consultas que operam sobre o eixo *Tempo-de-Transação*. Muitos autores denominam este tipo de consultas como operação de **Rollback**. Estas consultas retornam valores de tempo ou objetos, dependendo da cláusula temporal a ser utilizada;
- Geração de consultas que operam sobre o eixo *Tempo-Válido*. Este tipo de consulta também são referenciadas como consultas **Históricas**. Estas consultas retornam valores de tempo ou objetos;
- Geração de consultas atemporais.

Este capítulo descreve a implementação do *Gerenciador de Consultas*, bem como, as características e propriedades deste gerenciador. Este capítulo foi estruturado da seguinte

forma: a seção 6.2 descreve as claúsulas temporais que podem ser aplicadas através do *Gerenciador de Consultas*; a seção 6.3 descreve como está implementado o gerenciador, detalhando-se no processo de mapeamento de consultas TOOL em consultas O_2; a seção 6.4 apresenta exemplos de consultas temporais que podem ser construídas através do *Gerenciador de Consultas*; na seção 6.5 é descrito o *Benchmark* proposto em [ea93]; a seção 6.6 apresenta alguns problemas de implementação surgidos durante o processo de desenvolvimento do gerenciador.

6.2 As Cláusulas Temporais

A linguagem **TOOL** propõe diversas cláusulas temporais. Na realidade, cada cláusula temporal pode representar um conjunto de consultas básicas sobre um banco de dados. Com base nesta premissa, foram projetados e desenvolvidos algoritmos para implementar as cláusulas temporais da linguagem **TOOL** no *Gerenciador de Consultas* onde a função de cada algoritmo é gerar um conjunto de consultas básicas O_2. Este conjunto de consultas O_2 apresenta, obrigatoriamente, semântica equivalente à semântica da claúsula temporal utilizada na consulta solicitada pelo usuário. Em outras palavras, este conjunto de consultas O_2 deve apresentar um resultado igual ao resultado da aplicação direta da cláusula temporal sobre o banco de dados.

Para facilitar a definição do conjunto de consultas temporais que deveriam ser construídas pelo *Gerenciador de Consultas*, agrupou-se as consultas temporais em função da cláusula temporal presente na consulta:

- Consultas que retornam **valores de tempo** sobre o eixo *Tempo-de-Transação*. Este tipo de consulta utiliza a cláusula **TWhen**;
- Consultas que retornam **valores de tempo** sobre o eixo *Tempo-Válido*. Este tipo de consulta utiliza a cláusula **VWhen**;
- Consultas que retornam **objetos** para um determinado valor de tempo sobre o eixo *Tempo-de-Transação*. Este tipo de consulta utiliza a cláusula **TSlice**;
- Consultas que retornam **objetos** para um determinado valor de tempo sobre o eixo *Tempo-Válido*. Este tipo de consulta utiliza a cláusula **VSlice**;
- Consultas atemporais. Este tipo de consulta retorna objetos ou valores para o último histórico do último estado do objeto no BD (considerando uma consulta sobre uma classe Bitemporal). Em outras palavras, este tipo de consulta opera em classes temporais como se estas fossem instantâneas. Este tipo de consulta é caracterizado pela ausência de cláusulas temporais.

A seguir serão descritos os conjuntos de operações básicas de cada cláusula temporal implementada no *Gerenciador de Consultas*. Estas operações foram implementadas considerando-se o modelo proposto para incorporar o conceito de Tempo no O_2.

6.2.1 Cláusula TWhen

Esta cláusula comporta-se como uma projeção temporal sobre um conjunto de objetos. As seguintes operações básicas estão embutidas nesta cláusula:

1. Selecionar um conjunto de objetos (esta operação de seleção está presente nos conjuntos de operações básicas de todas cláusulas temporais definidas para o *Gerenciador de Consultas*);

2. Executar uma projeção sobre cada objeto do conjunto definido no item anterior. Esta projeção deve ser aplicada aos atributos que representam os valores relativos a dimensão *Tempo-de-Transação*.

O usuário pode desejar aplicar uma consulta temporal a uma determinada extensão de uma classe, ou então, apenas um subconjunto desta extensão. Por este motivo, torna-se necessária uma operação de *seleção* para restringir o conjunto de objetos sobre os quais deve-se aplicar a cláusula **TWhen**. Esta operação de *seleção* é a operação padrão de seleção existente no O_2. Em outras palavras, é uma operação que seleciona objetos do BD a partir de um predicado não-temporal (item 1).

A execução da operação apresentada no item 2 representa percorrer a lista que armazena os diversos estados de um atributo de um determinado objeto (veja definição desta estrutura no capítulo 4).

6.2.2 Cláusula VWhen

Esta cláusula é a contrapartida da cláusula *TWhen* para o eixo temporal *Tempo-Válido*. Portanto, comporta-se também como uma projeção temporal, mas que opera sobre o eixo *Tempo-Válido*.

O conjunto de operações básicas desta cláusula difere do conjunto de operações básicas da cláusula *TWhen* apenas no processamento da operação de projeção.

A operação de projeção para esta cláusula tem processamentos distintos, dependendo se a cláusula está sendo aplicada sobre objetos ou sobre atributos de um objeto. Esta diferenciação na execução da operação de projeção decorre do fato de que os atributos que representam a dimensão *Tempo Válido* em um objeto são diferentes dos atributos que representam esta dimensão em atributos que compõem este objeto (veja definição dos atributos que armazenam o histórico de objetos de seus atributos no capítulo 4).

A execução da operação de projeção para a cláusula **VWhen** é feita da seguinte maneira:

- Caso a cláusula seja aplicada a objetos, percorrer a lista que armazena o histórico do objeto;
- Caso a cláusula seja aplicada a atributos, percorrer a lista que armazena os diversos estados do atributo até um estado especificado pelo usuário. Posicionado o estado, percorrer a lista que armazena o histórico do atributo.

6.2.3 Cláusula TSlice

Esta cláusula tem a função de recuperar objetos para um dado valor de tempo sobre o eixo *Tempo-de-Transação*.

As operações básicas envolvidas no processamento da cláusula **TSlice** são as seguintes:

1. Selecionar um conjunto de objetos;
2. Para cada objeto do conjunto definido no item anterior, faça:
   (a) Para cada atributo do objeto, retornar o valor do atributo armazenado em um *Tempo-de-Transação*, cujo valor é o maior do conjunto de valores menores ou iguais ao especificado pelo usuário.

Quando a cláusula **TSlice** é aplicada a objetos pertencentes a uma classe *Bitemporal*, o item 2(a) retorna um conjunto de valores para cada atributo. Este conjunto de valores corresponde ao histórico do atributo para o estado posicionado.

O modelo utilizado para incorporar o contexto temporal no O_2 introduz o conceito que denominamos de **Composição Temporal**. Através desta propriedade, um objeto pode ser composto por objetos armazenados no BD em estados distintos, ou seja, objetos que apresentam valores de *Tempo-de-Transação* distintos.

Por exemplo, considere um objeto do tipo tupla composto pelos atributos <*nome*> e <*endereço*>. Suponha que foi atribuído ao atributo <*nome*> um valor **x** em um *Tempo-de-Transação* **TT1**, e que foi atribuído ao atributo <*endereço*> um valor **y** em um *Tempo-de-Transação* **TT2**. Portanto, o objeto, em questão, é composto por atributos cujos valores foram armazenados no BD em estados distintos. Isto implica que, para recuperar o último estado deste objeto, deve-se acessar valores de tempo distintos para a dimensão *Tempo de Transação*. Supondo **TT2** > **TT1**, os possíveis conjuntos de valores para este objeto são:

- vazio, em um estado **t** < **TT1**;
- { **x** }, em um estado **t**, tal que **TT1** ≤ **t** < **TT2**. Em outras palavras, no estado **t**, apenas o atributo <*nome*> possui valor determinado;

- { x,y }, em um estado t $\geq$ **TT2**. Em outras palavras, no estado t, os atributos <*nome*> e <*endereço*> possuem valores definidos.

Portanto, a recuperação do estado de um objeto em um tempo t deve considerar a lista dos tempos de transação de cada um de seus atributos. São recuperados os valores dos atributos armazenados no maior *Tempo de Transação* do conjunto de valores de *Tempo de Transção* menores ou iguais a t.

6.2.4 Cláusula VSlice

Esta cláusula é a contrapartida da cláusula *TSlice* para o eixo *Tempo-Válido*. Portanto tem a função de recuperar objetos para um determinado valor de tempo sobre o eixo *Tempo-Válido*.

As operações básicas que funcionam como suporte para a cláusula **VSlice** são as seguintes:

1. Selecionar um conjunto de objetos;
2. Para cada objeto definido no item 1, faça:
   (a) Recuperar o valor do atributo correspondente ao histórico especificado pelo usuário.

Quando a cláusula **VSlice** é aplicada a objetos pertencentes a uma classe *Bitemporal*, o item 2(a) retorna um conjunto de valores para cada atributo. Este conjunto de valores corresponde aos diversos estados do atributo para o histórico especificado.

6.2.5 Cláusulas *x*When e *y*Slice em uma mesma consulta

Existem consultas que requerem a combinação das cláusulas **xWhen** e **ySlice**.

Uma consulta que apresenta a utilização simultânea destas cláusulas (não importando a ordem em que aparecem na consulta) comporta-se da seguinte forma: *a cláusula especificada para o nível mais interno da consulta funcionará como argumento (predicado temporal) para a cláusula mais externa.*

Considere a seguinte consulta: *Quando os funcionários, que recebiam salários maior que US$ 1,200 em 20/12/1990, tiveram seu último aumento salarial ?* Observe que a formulação desta consulta exige que seja utilizada uma cláusula *VSlice* para obter o conjunto de funcionários que recebiam salários maior que US$ 1,200 em 20/12/1990. Após isto, aplicar a cláusula *VWhen* ao conjunto de objetos resultante da cláusula *VSlice*.

Neste exemplo, a cláusula *VSlice* comporta-se como argumento da cláusula *VWhen*. Em outras palavras, a cláusula *VSlice* deve ser especificada no nível mais interno da consulta e a cláusula *VWhen*, em um nível mais externo.

As operações básicas envolvidas em consultas que apresentam cláusulas **xWhen** e **ySlice** são a união das operações básicas de cada cláusula isoladamente. Por exemplo, na consulta apresentada como exemplo, as operações básicas representam a união das operações básicas da cláusula *VSlice* com as operações básicas da cláusula *VWhen*. Esta união deve ocorrer nesta ordem.

6.2.6 Cláusulas TWhen_View e VWhen_View

Por definição, as cláusulas **TWhen** e **VWhen** comportam-se como uma projeção temporal, em outras palavras, retornam apenas valores de tempo para as dimensões *Tempo-de-Transação* ou *Tempo-Válido*, dependendo da cláusula utilizada.

Muitas da respostas das consultas que utilizam as cláusulas **TWhen** e **VWhen** retornam um conjunto de valores de tempo. Em alguns casos, consultas deste tipo podem não atender as necessidades do usuário. Por exemplo, suponha que um usuário deseja efetuar a seguinte consulta: *Quando o funcionário de nome André Vitorino foi contratado pela empresa.* Para efetuar esta consulta seria necessária a utilização da cláusula **VWhen.** Considerando que na empresa podem existir vários funcionários com este mesmo nome, a consulta retornaria um conjunto de valores de *Tempo-Válido* que correspondem a data de contratação do funcionário. Contudo, o usuário deseja ter a informação de apenas um dos funcionários. Em outras palavras, o usuário necessita apenas de um valor de tempo. Porém a consulta irá retornar um conjunto de valores sem identificar a que funcionários pertencem os valores de tempo.

Para solucionar este tipo de problema, introduziu-se na sintaxe da linguagem TOOL as cláusulas temporais **TWhen_View** e **VWhen_View**. Estas cláusulas são extensões às cláusulas **TWhen** e **VWhen.**

As cláusulas **TWhen_View** e **VWhen_View** retornam valores de tempo, dando a opção ao usuário de "enxergar" também os objetos que possuem os valores de tempo retornados como resposta à consulta. Na realidade, estas cláusulas permitem ao usuário acessar apenas alguns valores de atributos dos objetos para que seja garantida a segurança no acesso de informações do Banco de Dados. Em outras palavras, estas cláusulas apresentam apenas uma visão (pré-definida) do objeto.

6.3 Implementação

O princípio básico de funcionamento do *Gerenciador de Consultas* é o seguinte: através de janelas, o gerenciador solicita ao usuário a especificação de parametros necessários à construção de uma consulta temporal. Com base nestes parametros, o *Gerenciador de Consultas* faz o mapeamento da consulta temporal em uma consulta da sintaxe O_2SQL. Após isto, submete a consulta ao processador de consultas do O_2.

Os parametros de consulta necessários à formulação de uma consulta temporal são os seguintes:

- *Nome da classe* a ser consultada (**Obrigatório**);
- *Nome da extensão* da classe a ser consultada (**Obrigatório**);
- *Cláusula Temporal* a ser utilizada - *TWhen, VWhen, TSlice, VSlice, Atemp*[1] (**Obrigatório**);
- *Predicado atemporal* (**Opcional**);
- *Predicado temporal* (**Opcional**);
- *Operadores temporais* (**Opcional**);
- *Construtores temporais* (**Opcional**);

O parametro *predicado atemporal* tem a função de selecionar um subconjunto de objetos da extensão da classe a ser consultada. Por exemplo, um usuário deseja aplicar uma consulta temporal aos funcionários de *sobrenome "Vitorino"*. Para formular esta consulta, o usuário deve especificar um predicado *em sintaxe* O_2SQL para restringir a consulta ao subconjunto de funcionários de sobrenome "Vitorino". Caso o usuário não especifique nenhum predicado atemporal, o conjunto de objetos ao qual será aplicada a consulta temporal corresponderá à extensão da classe a ser consultada.

O parametro *predicado temporal* tem a função de restringir *o intervalo de tempo* a ser consultado. Por exemplo, um usuário pode desejar aplicar uma consulta temporal a um conjunto de funcionários **quando** estes ganhavam um salário maior que US$ 1,200. Para formular esta consulta, é necessário que o usuário especifique um predicado para restringir o intervalo de tempo sobre o qual será aplicado a consulta para o intervalo correspondente ao período no qual cada funcionário recebia salário maior que 1,200.

6.3.1 Operadores Temporais

Os operadores temporais implementados são os seguintes:
Before, Before_Overlap, After, After_Overlap.

O operador **Before** tem como função testar se um conjunto de eventos ou intervalos de tempo ocorreu **antes** de um determinado evento. Considere a seguinte sintaxe como sintaxe para o operador *Before*:

< *operando_1* > **Before** <*operando_2*>

[1]Esta opção refere-se à aplicação de consultas não temporais, caracterizadas pela ausência de cláusulas temporais.

Para o *Gerenciador de Consultas*, <*operando_1*> representa um conjunto de eventos ou intervalos de tempo. <*operando_2*> representa um evento utilizado como parametro de comparação.

<*operando_1*> pode ser especificado através de um predicado temporal cuja funcionalidade é retornar um conjunto de eventos ou de intervalos de tempo. Por exemplo, suponha uma consulta temporal a um conjunto de funcionários quando estes recebiam salários maiores que US$ 1,200 antes de 12/11/1990. Para esta consulta, o conjunto de valores de tempo (eventos ou intervalos, dependendo do modelo de representação da dimensão *Tempo Válido*) correspondente a <*operando_1*> é obtido através do predicado que retorna os valores de tempo para os quais cada funcionário recebia mais que US$ 1,200.

Para operações com intervalos, o operador **Before** retornará o valor *booleano* **True** se, e somente se, o valor do evento correspondente ao limite superior do intervalo é menor que o valor do evento correspondente ao limite inferior do <*operando_2*>. Formalmente:

$$[t_x, t_y] \textbf{ Before } [t_z, t_w] = \text{True} \Longleftrightarrow t_y < t_z$$

O operador **Before_Overlap** apresenta sintaxe semelhante à do operador **Before**. Semanticamente, a diferença reside no fato que o operador **Before_Overlap** testa se algum sub-intervalo do intervalo representado por <*operando_1*> ocorreu antes do intervalo representado por <*operando_2*>.

O operador **Before_Overlap** deve ser aplicado apenas quando:

- há a manipulação de intervalos de tempo, e quando;
- há a necessidade de que a consulta retorne um sub-intervalo (do intervalo consultado). Por exemplo, uma consulta pode solicitar os intervalos de tempo nos quais um determinado funcionário recebia salário menor que US$ 1,200 *antes* de 12/11/1990. Supondo que este funcionário recebia US$ 900 no período de 08/01/1986 a 25/12/1991, então a resposta para esta consulta deve ser o sub-intervalo [08/01/1986,12/11/1990]. Para este tipo de consulta deve ser aplicado o operador **Before_Overlap** e não, o operador **Before**.

Como pode ser observado, o operador **Before_Overlap** retorna valores de tempo (sub-intervalos do intervalo de tempo consultado). A opção por defini-lo como operador temporal é justificada pelo fato de que a operação principal executada é a comparação. Após esta operação de comparação, são retornados valores de tempo. Caso não haja sobreposição, é retornado um intervalo vazio.

O operador **Before_Overlap** retorna um intervalo não vazio se, e somente se, o valor do evento correspondente ao limite inferior do intervalo consultado é menor que evento correspondente ao limite inferior do <*Operando_2*>. Formalmente:

$$[t_x, t_y] \textbf{ Before\_Overlap } [t_z, t_w] \neq \emptyset \Longleftrightarrow t_x < t_z$$

Quando $t_x < t_z$ e $t_y > t_z$, o operador **Before_Overlap** retornará o sub-intervalo $[t_x, t_z]$ como solução. Em outras palavras, a consulta resultante da aplicação do operador **Before_Overlap** apresentará, como resposta, neste caso, o sub-intervalo $[t_x, t_z]$ e não, o intervalo $[t_x, t_y]$.

O operador **After** é o inverso do operador **Before**. Em outras palavras, o operador **After** tem a função de testar se um conjunto de eventos ou intervalos de tempo ocorreu **após** um determinado evento. A sintaxe é semelhante à sintaxe do operador **Before**:

$$<operando\_1> \textbf{ After } <operando\_2>$$

As mesmas regras sintáticas apresentadas para o operador **Before** também são válidas para o operador **After**.

Para operações com intervalos, o operador **After** retornará o valor *booleano* **True** se, e somente se, o valor do evento correspondente ao limite inferior do intervalo é maior que o valor do evento correspondente ao limite superior do *<operando_2>*. Formalmente:

$$[t_x, t_y] \textbf{ After } [t_z, t_w] = \text{True} \Longleftrightarrow t_x > t_w$$

O operador **After_Overlap** é o inverso semântico para o operador **Before_Overlap**. Formalmente, o operador **After_Overlap** retornará um intervalo não vazio na seguinte situação:

$$[t_x, t_y] \textbf{ After\_Overlap } [t_z, t_w] \neq \emptyset \Longleftrightarrow t_y > t_w$$

Quando $t_y > t_w$ e $t_x < t_w$, o operador **After_Overlap** retornará o sub-intervalo $[t_w, t_y]$ como solução. Em outras palavras, a consulta resultante da aplicação do operador **After_Overlap** apresentará, como resposta, neste caso, o sub-intervalo $[t_w, t_y]$ e não, o intervalo $[t_x, t_y]$.

6.3.2 Construtores Temporais

Os construtores temporais implementados são os seguintes:
First_Instant, Last_Instant, Max_Interval, Min_Interval, All_Interval.

A semântica destes construtores assemelha-se à semântica dos operadores de conjunto. A seguir, é descrita a semântica de cada construtor:

First_Instant: retorna o menor valor de tempo para um conjunto destes valores (*Tempo-Válido* ou *Tempo-de-Transação*). Pode ser utlizado, por exemplo, em consultas do tipo:

- *(i)* Quando um determinado funcionário teve seu contrato efetivado (ou seja, o instante em que o funcionário tornou-se válido);

- *(ii)* Quando foi a primeira alteração de salário de cada funcionário após 08/01/1986.

Last_Instant: retorna o maior valor de tempo para um conjunto de valores de tempo (*Tempo-Válido* ou *Tempo-de-Transação*). Pode ser utilizado em consultas do tipo:

- *(i)* Quando um determinado funcionário foi demitido (ou seja, o último instante em que o funcionário era válido);
- *(ii)* Quando foi a última alteração de salário de cada funcionário do departamento DCC.

Max_Interval: por definição este operador só se aplica a valores de tempo sobre o eixo *Tempo-Válido*[2]. Este construtor tem a função de retornar o maior intervalo de tempo pertencente a um conjunto de intervalos para um dado conjunto de objetos. Aplica-se a consultas do tipo:

- *(i)* Qual o funcionário com maior tempo de serviço ininterrupto na empresa;
- *(ii)* Qual o funcionário que permaneceu o maior período sem ter aumentos salariais;
- *(iii)* Qual o maior período que algum funcionário ficou sem receber aumento de salário.

Min_Interval: este construtor é o inverso do construtor **Max_Interval.**

All_Interval: retorna todos os intervalos de tempo de um conjunto de intervalos para um determinado objeto. Em outras palavras, tem a função de retornar o histórico de um objeto, ou então, de um atributo em um determinado estado do BD. Pode ser aplicado em consultas do tipo:

- *(i)* Listar o histórico salarial de todos os funcionários do departamento DCC.

6.3.3 Geração de Consultas

Após a interação com o usuário, o *Gerenciador de Consultas* gera uma consulta com a sintaxe O_2SQL. Esta consulta O_2 caracteriza-se por ter sido gerada a partir de uma consulta TOOL virtual, garantindo-se a semântica desta consulta virtual. Na realidade, esta manutenção da semântica da consulta virtual representa garantir as especificações fornecidas pelo usuário, para que seja retornado exatamente o que usuário deseja.

A consulta em O_2SQL é, então, submetida ao processador de consultas do O_2 através da função O_2**query** e o resultado da consulta é finalmente apresentado ao usuário.

[2]Considerando-se o modelo de representação das dimensões temporais implementado nesta dissertação.

Os valores de tempo para consultas sobre o eixo *Tempo-Válido* são manipulados conforme granularidade (virtual) informada no momento de criação da classe (veja explicação no capítulo 4). Os valores de tempo armazenados internamente pela *Camada de Gerenciamento Temporal* apresentam uma granularidade que utiliza a métrica **segundo**. Para compatibilizar a manipulação de valores de tempo que apresentem granularidade virtual diferente de **segundo**, foi desenvolvida uma função para realizar a conversão entre a granularidade virtual e a granularidade interna dos valores de tempo. Por exemplo, suponha que o atributo <*salário*>, definido para objetos pertencentes à classe *Funcionário* apresenta uma granularidade virtual <**ano, mês, dia**>. Portanto, o usuário irá manipular valores de tempo para este atributo através desta granularidade virtual. A função de conversão encarregar-se-á de converter valores de tempo nesta granularidade virtual para a granularidade **segundos**. Esta é a implementação do conceito de **Granularidade Virtual** descrita no capítulo 4.

Durante uma consulta, o usuário poderá especificar a granularidade com que irá manipular valores de tempo sobre o eixo *Tempo-de-Transação*. A função de conversão mapeará valores de tempo expressos em uma granularidade diferente de **segundo** para a granularidade interna da *Camada de Gerenciamento Temporal*. Esta é a implementação da operação de **Projeção Temporal sobre Granularidade** descrita no capítulo 4.

6.4 Exemplos de Consultas Temporais

Considere que as consultas a serem apresentados nesta seção são aplicadas à classe *Funcionário* cuja definição é apresentada a seguir:

```
class Funcionario public type
    tuple(nome: list(tuple(tt_nome:real,
                           historico_nome:list(tuple(valor_nome:string,
                                                     inicio_tv_nome:real,
                                                     fim_tv_nome:real)))),
          salario: list(tuple(tt_salario:real,
                              historico_salario:list(tuple(valor_salario:real,
                                                           inicio_tv_salario:real,
                                                           fim_tv_salario:real)))),
          historico_objeto: list(tuple(inicio_tv:real,
                                       fim_tv:real)))
end;
name Funcionarios[3]: set(Funcionario);
```

[3]Repositório de dados da classe *Funcionário*.

Consulta 1:

→ *Quando os funcionários de sobrenome "Vitorino" receberam salário maior que US$ 1,200 ?*

→ Consulta na sintaxe TOOL

*VWhen_View (nome like "*Vitorino*")*
From Funcionário
INDB (during (salario > 1,200))

→ Consulta O_2SQL gerada

—— A consulta O_2SQL é executada em quatro etapas:

1. Seleção;
2. Seleção;
3. Projeção;
4. Agrupamento dos valores de tempo por objeto[4].

1. Seleção dos objetos que possuem sobrenome "Vitorino"

```
define q1 as select o from o in Funcionarios
        where (element( select (last(last(p.nome).historico_nome)).valor_nome
                        from p in Funcionarios
                        where o=p)) like "*Vitorino*" and
              (element( select (last(q.historico_objeto)).fim_tv
                        from q in Funcionarios
                        where o=q)) = 0 and
              (element( select (last(q.historico_objeto)).inicio_tv
                        from q in Funcionarios
                        where o=q)) != 0
```

2. Seleção dos objetos de q1 que em algum período ganhavam mais que 1,200 e
3. Projeção sobre os atributos de tempo referentes a estes períodos

[4]Esta etapa é necessária, pois uma consulta temporal pode retornar um conjunto de intervalos de tempo pertencentes a objetos distintos. O processo de agrupamento visa unir os intervalos de tempo de cada objeto antes de retornar o resultado da consulta para o usuário. Na tela, são apresentados um conjunto de intervalos por objetos.

define q2 as select tuple(objeto: t, inicio:l.inicio_tv_salario, fim:l.fim_tv_salario)
from t in q1,
p in (select last(o.salario) from o in q1
where o=t),
l in p.historico_salario
where l.valor_salario > 1,200

4. Agrupamento dos valores de tempo por objeto

q3: group resposta in q2 by (objeto: resposta.objeto)
with (intervalos: select tuple(inicio:p.inicio, fim:p.fim)
from p in partition)

Consulta 2:

→ *Quando foi a última vez que os funcionários, que recebem salário maior que US$ 1,200, tiveram aumento salarial ?*

→ **Consulta na sintaxe TOOL**
Last_Instant(VWhen_View (salario > 1,200)
From Funcionário)

→ **Consulta O_2SQL gerada**

—— A consulta O_2SQL é executada em três etapas:

1. Seleção;
2. Seleção;
3. Projeção, e;
4. Seleção.

1. Seleção dos objetos que possuem salário maior que 1,200

q1: define q1 as select o from o in Funcionarios
where (element(select (last(last(p.salario).historico_salario)).valor_salario
from p in Funcionarios
where o=p)) > 1,200 and
(element(select (last(q.historico_objeto)).fim_tv
from q in Funcionarios
where o=q)) = 0 and
(element(select (last(q.historico_objeto)).inicio_tv
from q in Funcionarios
where o=q)) != 0

2. Seleção dos objetos de q1 que em algum período ganhavam mais que 1,200 e
3. Projeção sobre os atributos de tempo referentes a estes períodos

```
q2: define q2 as select tuple(objeto: t, inicio:l.inicio_tv_salario, fim:l.fim_tv_salario)
                from t in q1,
                     p in (select last(o.salario) from o in q1
                           where o=t),
                     l in p.historico_salario
```

4. Seleção do maior valor de tempo referente ao limite inferior dos intervalos de tempo que representam a dimensão Tempo-Válido

```
q3: select tuple(objeto: t.objeto, inicio:t.inicio) from t in q2
    where t.inicio=max(select o.inicio from o in q2 where o.objeto=t.objeto)
```

Consulta 3:

→ *Listar o histórico salarial de todos funcionários*

→ Consulta na sintaxe TOOL

```
VWhen_View ()
From Funcionário
```

→ Consulta O_2SQL gerada

—— A consulta O_2SQL é executada em dois passos:

1. projeção;
2. agrupamento dos valores de tempo por objeto.

1. projeção sobre os atributos de tempo

```
define q1 as select tuple(objeto: t, inicio:l.inicio_tv_salario, fim:l.fim_tv_salario)
             from t in Funcionarios,
                  p in (select last(o.salario) from o in Funcionarios
                        where o=t),
                  l in p.historico_salario
```

2. Agrupamento dos valores de tempo por objeto

```
q2: group resposta in q2 by (objeto: resposta.objeto)
          with (intervalos: select tuple(inicio:p.inicio, fim:p.fim)
                            from p in partition)
```

Um observação importante que deve ser feita, com base nos exemplos de consultas acima, é a seguinte: *uma única consulta TOOL é mapeada em uma sequência de consultas O_2.*

6.5 Benchmark para Linguagens de Consultas Temporais

Em [ea93] é proposto um *Benchmark* para linguagens de consultas temporais. Este *Benchmark* caracteriza-se por ser um *Benchmark* semântico, ou seja, seu objetivo principal é identificar o nível de facilidade que as linguagens de consultas avaliadas podem oferecer na interação com o usuário. Não há uma preocupação em avaliar o desempenho das linguagens testadas através do *Benchmark* [ea93].

Segundo [ea93], o *Benchmark* proposto fornece um suporte para comparar as características descritivas e operacionais e a capacidade dos modelos temporais e das linguagens de consultas temporais.

O *Benchmark* é composto de:

- um esquema de um BDT;
- um conjunto de valores para o esquema (instâncias);
- um conjunto de consultas que devem ser aplicadas ao BDT.

Apesar do suporte temporal (modelo utilizado para incorporar o conceito de tempo, por exemplo) não ser considerado nas avaliações realizadas pelo *Benchmark*, o modelo de dados para o qual *Benchmark* foi definido é o *Relacional.*

O *Benchmark* proposto em [ea93] ainda apresenta as seguintes restrições:

1. consultas restritas a valores sobre o eixo *Tempo-Válido*;
2. não são consideradas a evolução do esquema e o controle de versões;
3. não são consideradas consultas que utilizam *funções agregadas*;
4. não consideradas as atualizações sobre o BDT;

O *Gerenciador de Consultas* proposto nesta dissertação viabiliza a aplicação de consultas a valores sobre os dois eixos temporais. Através de testes realizados com a execução do protótipo, verificou-se que é possível executar todas as consultas definidas para o *Benchmark.* Os testes foram realizados sobre objetos da classe *Funcionário*, cuja definição foi apresentada na seção anterior.

Através do *Gerenciador de Consultas* proposto, conseguiu-se efetuar consultas do tipo:

- *Qual o funcionário que permaneceu o maior período sem receber aumentos de salário?*
- *Quando foi o último aumento de salário para cada funcionário antes de 10/11/1992?*
- *Quando foi o primeiro aumento de salário após 25/12/1991?*
- *Quando os objetos de sobrenome "Vitorino" foram incluídos no BD?*

Como pode ser observado, o *Gerenciador de Consultas* engloba um maior número de consultas em relação ao *Benchmark*, pois permite consultas sobre o eixo *Tempo-de-Transação.*

6.6 Problemas de Implementação

A estratégia de gerar consultas em tempo de execução gerou alguns problemas de implementação, discutidos nesta seção.

O *Gerenciador de Consultas* submete as consultas construídas ao processador de consultas do O_2 através da função **o2query**. Uma chamada a esta função apresenta a seguinte sintaxe:

o2query (resultado, consulta)

onde **resultado** é a variável através da qual é retornado o resultado da consulta, e **consulta** é a variável (do tipo *string*) através da qual é passado o corpo (texto) da consulta.

Por definição, as cláusulas **TWhen_View, VWhen_View, TSlice e VSlice** retornam objetos como resultado de consultas temporais. Isto implica que a variável **resultado** deve ser definida com um tipo que seja compatível com o tipo dos objetos a serem consultados. Isto significa que o tipo da variável **resultado** só é conhecido em tempo de execução do *Gerenciador de Consultas*, mais precisamente, no momento em que o usuário define a classe a ser consultada.

Contudo, o O_2 não permite a definição de variáveis em tempo de excução, apenas em tempo de compilação. Tornou-se necessário, então, a definição de um mecanismo que viabilizasse a definição do tipo da variável **resultado** em tempo de execução. O mecanismo definido apresenta o seguinte funcionamento:

1. Foi definida uma função genérica, denominada **efetua_consulta**, para apresentar o resultado de qualquer consulta;
2. Durante a manipulação de uma classe de um determinado esquema, a *Camada de Gerenciamento Temporal* inclui na função **efetua_consulta** o código correspondente à definição da variável **resultado** para a classe que está sendo manipulada;

3. A função **efetua_consulta** é compilada automaticamente pela *Camada de Gerenciamento Temporal*

Outros problemas de implementação surgiram em função de limitações apresentadas pela versão do O_2 existente no DCC, como, por exemplo, a não implementação do comando *import schema* para um esquema completo. Na versão existente no DCC, este comando só se aplica a classes.

Capítulo 7

Conclusões

Esta dissertação implementou um protótipo para manipular esquemas e consultar Bancos de Dados Temporais Orientados a Objetos. O protótipo foi baseado no modelo **TOODM** proposto em [Lin93]. Utilizou-se também a linguagem de consulta proposta para o TOODM, a linguagem **TOOL** [Lin93].

O protótipo foi implementado para o sistema O_2, através da criação da *Camada de Gerenciamento Temporal* e de um *Gerenciador de Consultas* sobre esta camada.

A *Camada de Gerenciamento Temporal* é responsável pelo gerenciamento dos esquemas e dados que incorporam o contexto temporal, suportando as dimensões temporais *Tempo de Transação* e *Tempo Válido*. Estas dimensões foram implementadas através dos construtores **Tuple** e **List** do SGBD O_2. Portanto, qualquer SBDOO, que apresente o mesmo conjunto de construtores de tipos, poderá ser utilizado como suporte operacional para implementar o protótipo apresentado nesta dissertação.

O *Gerenciador de Consultas* funciona como uma *interface* entre o usuário e o processador de consultas do O_2. É responsável pela aplicação de consultas temporais sobre um BD definido pela *Camada de Gerenciamento Temporal*. O *Gerenciador de Consultas* caracteriza-se pelo mapeamento de consultas de sintaxe TOOL em consultas de sintaxe O_2SQL que são submetidas ao processador de consultas do O_2.

O *Gerenciador de Consultas* permite executar as consultas temporais pertencentes ao conjunto definido para o *Benchmark* proposto em [ea93]. Além disso, pode construir outros tipos de consultas não presentes no *Benchmark* como, por exemplo, consultas sobre o eixo temporal *Tempo-de-Transação*. Desta forma, o sistema implementado é mais genérico e flexível que o especificado neste *Benchmark*.

O protótipo implementado tem cerca de 3.000 linhas de código fonte O_2C. Sua implementação apresentou uma série de dificuldades técnicas, discutidas no texto.

As principais contribuições apresentadas por este trabalho são:

Teóricas :

⟶ A análise de linguagens de consultas temporais de Snodgrass e de Medina foi estendida e nova taxonomia foi proposta;

⟶ Foram propostas estruturas de dados que viabilizam a incorporação do contexto temporal em SBDOO, permitindo, desta forma, a criação, manipulação e consulta de objetos temporais. Estas estruturas podem ser utilizadas para incorporar o contexto temporal em qualquer SBDOO que contenha os construtores de tipos **Tupla** e **Lista** em seu conjunto de construtores. Portanto, estas estruturas propostas não dependem do sistema O_2;

⟶ Foram desenvolvidos algoritmos para processar consultas temporais com base nas estruturas temporais propostas;

Práticas :

⟶ Foi desenvolvido um protótipo sobre o sistema O_2 que viabiliza o gerenciamento temporal em um SBDOO. O processo de desenvolvimento utilizou como suporte teórico os conceitos e propriedades propostos e apresentados nessa dissertação;

⟶ Foram identificados e resolvidos problemas inerentes ao processamento consultas temporais em um BDOO. Por exemplo, uma contribuição neste sentido foi a indicação de como deve ser o tratamento de consultas que apresentam como resultado um conjunto de valores temporais.

Podemos agrupar as extensões à este trabalho em dois diferentes níveis de abstração:

- No nível de abstração mais alto, agrupam-se as extensões que objetivam capturar mais elementos da semântica temporal. Podemos citar as seguintes extensões:
  - Incorporação do contexto temporal no gerenciamento da evolução de esquemas;
  - Incorporação do contexto temporal no controle de versões de métodos;
  - Implementação de outros tipos de consultas e do processo de temporalização;
- Em um nível de abstração mais baixo, trabalhar-se-ia diretamente com o SGBDOO utilizado como núcleo da implementação, no nosso caso o O_2. Neste nível situam-se as seguintes extensões:
  - Extensão da álgebra do O_2 para introduzir operações temporais;
  - Implementação uma linguagem de definição de dados e uma linguagem de consulta (ao invés de *interfaces*) para manipular o Tempo em um SBDOO.

Bibliografia

[Ari86] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[AS89] Isoo Ahn and Richard Snodgrass. Performance Analysis of Temporal Queries. *Information Sciences*, (49):103–146, November 1989.

[Ban88] François Bancilhon. Object-Oriented Database Systems. In *Proceedings ACM SIGMOD-SIGACT Symposium*, 1988.

[CC87] James Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, 1987.

[CD92] Sophie Cluet and Claude Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 383–392, 1992.

[Chr88] Christophe Lecluse, Philippe Richard and Fernando Velez. O2, an Object-Oriented Data Model. In *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pages 424–433, 1988.

[Chr92] Christian Jensen, Michael D. Soo and Richard T. Snodgrass. Unification of Temporal Data Models. Technical report, Department of Computer Science, University of Arizona, 1992.

[C.S92] C.S. Jensen, J. Clifford, S.K. Gadia, A. Segev and R. Snodgrass. A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3), September 1992.

[CW83] James Clifford and David S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[Dat88] C.J. Date. A Proposal for Adding Date and Time support to SQL. *ACM SIGMOD Record*, 17(2):53–76, June 1988.

[dddO91] Grupo de desenvolvimento do O2. O2. *Communications of the ACM*, 34(10):35–48, 1991.

[Dit86] Klaus R. Dittrich. Object-Oriented Database Systems - Workshop report. In *Proceedings of The Fifth International Conference on Entity-Relationship Approach*, 1986.

[DS92] Curtis E. Dyreson and Richard T. Snodgrass. Time-stamp Semantics and Representation. Technical report, Department of Computer Science, University of Arizona, 1992.

[DS93] Curtis E. Dyreson and Richard T. Snodgrass. Efficient time-stamp Input and Output. Technical report, Department of Computer Science, University of Arizona, 1993.

[DW92] Umeshwar Dayal and Gene T.J. Wuu. A Uniform Approach to Processing Temporal Queries. In *Proceedings of the 18th VLDB Conference*, pages 407–418, 1992.

[ea93] Christian S. Jensen et al. The TSQL Benchmark. In *Proceedings of the International Workshop on a Infrastructure for Temporal Databases*, pages QQ–1–QQ–28, June 1993.

[Eug76] Eugene Wong and Karel Youssefi. Decomposition - A Strategy for Query Processing. *ACM Trans. on Database System*, 1(3):223–241, September 1976.

[EW90] Ramez Elmasri and Gene T. J. Wuu. A TEMPORAL MODEL AND QUERY LANGUAGE FOR ER DATABASES. In *Proceedings of the 6th Conference on Data Engineering*, pages 76–83, February 1990.

[Gad88] Shashi K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[GS90] Himawam Gunadhi and Arie Segev. A framework for query optimization in temporal databases. In *Fifth International Conference on Statistical and Scientific Data*, pages 131–147, April 1990.

[Jac92] Jack Orenstein, Sam Haradhvala, Benson Margulies and Don Sakahara. Query Processing in the ObjectStore Database System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 403–412, 1992.

[Jay87] Jay Banerjee, Hong-T Chou, Jorge Garza, Won Kim, Darrell Woelk, Nat Ballou and H. J. Kim. Data Model Issues for Object-Oriented Applications. *ACM TOIS*, 1987.

[LEMS87] Jr. L. Edwin McKenzie and R. Snodgrass. Extending the Relational Algebra to Support Transaction Time. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 467–478, May 1987.

[LEMS91] Jr. L. Edwin McKenzie and Richard T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[Lin93] Lincoln Cesar Medina de Oliveira. Incorporação da Dimensão Temporal em Bancos de Dados Orientados a Objetos. Master's thesis, Universidade Estadual de Campinas, 1993.

[LM90] T.Y. Cliff Leung and Richard R. Muntz. Query Processing for Temporal Databases. In *Proceedings of the 6th International Conference on Data Engineering*, pages 200–208, 1990.

[LM92] T.Y. Cliff Leung and Richard R. Muntz. Temporal Query Processing and Optimizations in Multiprocessor Database Machines. In *Proceedings of the 18th VLDB Conference*, pages 383–393, 1992.

[Mal89] Malclom Atkinson, David Dewitt, David Maier, François Bancilhon, Klaus Dittrich and Stanley Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, 1989.

[Mat84] Matthias Jarke and Jürgen Koch. Query Optimization in Database System. *ACM Computing Surveys*, 16(2):111–152, June 1984.

[McL91] Dennis McLeod. Time in object databases. Technical report, Computer Science Department, University of Southern California, 1991.

[Mic92] Michael Kifer, Won Kim and Yehoshua Sagiv. Querying Object-Oriented Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–402, 1992.

[NA87] Shamkant B. Navathe and Rafi Ahmed. TSQL - A language interface for history databases. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 113–128, 1987.

[NA89] Shamkant B. Navathe and Rafi Ahmed. A temporal relational model and a query language. *Information Sciences*, (49):147–175, 1989.

[Nie88] O. M. Nierstrasz. A Survey of Object-Oriented Concepts. In *Object-Oriented Concepts and Databases*. Addison-Wesley, 1988.

[S. 79] S. Jones, P. Mason and R. Stamper. LEGOL 2.0: A relational specification language for complex rules. *Information Systems*, 4(4):293–305, November 1979.

[SA85a] Richard Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of the ACM SIGMOD International Conference on Manegement of Data*, pages 236–246, May 1985.

[SA85b] Richard Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, pages 35–42, September 1985.

[Sad87] Sadeghi,R., Samson, W.B., Deen, S.M. Hql: A historical query language. *Technical report, Dundee College of Technology*, 1987.

[Sar90] N.L. Sarda. Algebra and Query Language for a Historical Data Model. *The Computer Journal*, 33(1):11–18, 1990.

[SC91] Stanley Y. W. Su and Hsin-Hsing M. Chen. A temporal knowledge representation model OSAM*/T and its query language OQL/T. In *Proceedings of the International Conference on Very Large Data Bases*, pages 431–442, september 1991.

[Sha92] Shashi K. Gadia, Sunil S. Nair and Yiu-Cheong Poon. Incomplete information in relational temporal databases. In *Proceedings of the 18th VLDB Conference*, pages 395–406, 1992.

[SK86] Arie Shoshani and Kyoji Kawagoe. Temporal Data Management. In *International Conference on Very Large Databases Systems*, pages 79–88, 1986.

[Sno87] Richard Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno92] Richard T. Snodgrass. Temporal Databases. In *Theories and Methods of Spatial-Temporal Reasoning in Geographic Space*, pages 22–64. Spring-Verlag, 1992. Lecture Notes in Computer Science - Vol. 639.

[TA86] Adullah U. Tansel and M.E. Arkun. HQUEL: A query language for historical relational databases. In *Proceedings of the 3rd International Workshop on Statistical and Scientific Databases*, 1986.

[Tan86] Adullah U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[Tec92] O2 Technology. *The O2 User's Manual.* O2 Technology, 1992.

[Vos91a] Gottfried Vossen. Bibliography on Object-Oriented Database Management. *ACM SIGMOD Record*, 20(1):24–46, March 1991.

[Vos91b] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley, 1991.

[WD92] Gene T. J. Wuu and Umeshwar Dayal. A uniform model for temporal object-oriented databases. In *Proceedings of the International Conference on Data Engineering*, pages 584–593, 1992.

[Wol92] Wolfgang Käfer and Harald Schöning. Realizing a Temporal Complex-Object Data Model. In *Proceedings of the ACM SIGMOD International Conference on Management Data*, pages 266–275, 1992.