UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Proposta de um *Framework* Conceitual para Apoiar a Criação de Técnicas de Indexação para Banco de Dados Temporais

por

ALEXANDRE MACHADO LEHNEN

Dissertação submetida à avaliação, como
requisito parcial para a obtenção do grau
de Mestre em Ciência da Computação.

Prof. Dra. Nina Edelweiss
Orientadora

Porto Alegre, janeiro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lehnen, Alexandre Machado
Proposta de um Framework Conceitual para Apoiar a Criação de Técnicas de Indexação para Banco de Dados Temporais / por Alexandre Machado Lehnen. – Porto Alegre: PPGC da UFRGS, 2002.
100 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande Sul. Programa de Pós-Graduação em Computação – Porto Alegre, BR – RS, 2001. Orientadora: Edelweiss, Nina.

1. Técnicas de Indexação Temporal 2. Técnicas de Indexação. 3. Framework. 4. Banco de Dados Temporais. I. Edelweiss, Nina. II. Título.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

BD	Banco de Dados
BDT	Banco de Dados Temporal
BD2T	Banco de Dados Bitemporal
IST	*Interval Spatial Transformation*
SGBD	Sistema Gerenciador de Banco de Dados
OO	Orientado a Objetos
TT	Tempo de Transação / *Transaction Time*
TV	Tempo de Validade / *Valid Time*
UML	*Unified Modeling Language*
TF-ORM	*Temporal Functionality in Objects With Roles Model*

Resumo

Bancos de Dados Temporais (BDTs) surgiram para tentar suprir a necessidade de se obter um melhor aproveitamento das informações que circulam atualmente. Porém, ao mesmo tempo em que é benéfico o seu uso, uma vez que armazenam o histórico das informações, existe um problema neste tipo de banco de dados, que é o desempenho. Além do grande volume de dados armazenados, este problema se agrava ainda mais devido à complexidade nas operações que governam os BDTs, como por exemplo, inclusão, remoção, alteração e consulta. Portanto, focalizando o problema, existe a necessidade de melhorar o desempenho dos BDTs frente às operações de manipulação de dados. Técnicas de indexação apropriadas para dados temporais podem amenizar este problema de desempenho.

Técnicas consagradas de indexação são largamente usadas, amparadas no seu alto grau de desempenho e portabilidade. São exemplos B-Tree, B^+-Tree e R-Tree, entre outras. Estas técnicas não suportam indexar os complexos BDTs, mas são fundamentais para que sirvam de base para novas estruturas que suportem esses tipos de dados.

As técnicas de indexação para dados temporais existentes não conseguem suprir a semântica temporal na sua totalidade. Existem ainda algumas deficiências do tipo: poucas técnicas que abrangem ao mesmo tempo tempo de validade e tempo de transação; não existe uma técnica que oferece informações do seu desempenho; a maioria não distingue ponto no tempo de intervalo de tempo; entre outras. Entretanto, possuem características relevantes em cada uma delas. Assim, um estudo das características mais importantes se tornou um fator importante para que possa ser desenvolvido um modelo capaz de auxiliar na criação de novas técnicas de indexação para dados temporais, a fim de contemplar melhor estes tipos de dados.

O objetivo deste trabalho é, com base nas características das técnicas estudadas, desenvolver um *framework* conceitual capaz de auxiliar na criação de novas técnicas de indexação para dados temporais. Esta estrutura apresenta as características mais relevantes das técnicas existentes, agregando novas idéias e conceitos para contemplar os dados temporais.

O *framework* conceitual desenvolvido agrega características de diferentes técnicas de indexação, possibilitando de variar a arquitetura de um índice para dados temporais, ajustando-os para um melhor desempenho em diferentes sistemas.

Para validar o *framework* proposto é apresentada uma especificação de índices para o modelo de dados TF-ORM (*Temporal Functionality in Objects With Roles Model*).

Palavras-Chave: técnicas de indexação temporal, técnicas de indexação, *framework* conceitual, banco de dados temporal.

TITLE: “THE PROPOSAL CONCEPTUAL FRAMEWORK TO HELP IN THE CREATION OF INDEXING TECHNIQUES FOR TEMPORAL DATABASES”

Abstract

Temporal databases (TDBs) were created in order to try to supply the need of obtaining a better use of information. However, at the same time in that its use is important, once they store the history of information, there is a generic problem for this kind of database: the performance. Besides the huge volume of stored data, this problem becomes still worse due to the complexity of the operations that govern these TDBs, for instance, insert, delete, update and queries. Therefore, focusing the problem, there is the need of improving the performance of the TDBs concerning to the operations of data manipulation. Appropriate indexing techniques for temporal data can soften this performance problem.

Traditional indexing techniques are used broadly, based in their high performance degree and portability. As examples we have: B-Tree, B+-Tree and R-Tree, among others. These techniques don't support TDBs indexing but they are fundamental to serve as a base for new structures to support these kinds of data.

The existing indexing techniques for temporal data can´t supply the temporal semantics in its totality. There are still some defaults such as: few techniques include valid time and transaction time at the same time; a technique that offers information of its performance doesn't exist; most of them don't distinguish timepoint from temporal interval; etc. However, there are relevant characteristics in each one of them. This way, a study of the most important characteristics became important to define a model capable of aiding in the creation of new indexing techniques for temporal data, in order to contemplate better theses kinds of data.

Based on the characteristics of the studied techniques, the objective of this work is to develop a conceptual framework able to help in the creation of new indexing techniques for temporal data. This structure presents the most relevant characteristics of the existent techniques, adding new ideas and concepts to support temporal data.

The proposed conceptual framework joins characteristics of different indexing techniques in an unique model, allowing to vary the architecture of an index for temporal data, adjusting it for a better performance in different systems.

To validate the proposed framework, a specification of indexes is presented for the TF-ORM (Temporal Functionality in Objects With Roles Model) data model.

Keywords: temporal techniques indexing, techniques indexing, conceptual framework, temporal databases.

1 Introdução

Devido ao crescimento da exigência de mercado para o surgimento de novas tecnologias a fim de tentar amenizar a demanda de informações que circulam atualmente, surgiram os Bancos de Dados Temporais (BDTs) [TAN93, EDE94]. As vantagens do um BDT só podem surtir efeito se existir um método eficiente para pesquisas temporais. Mesmo amenizando a exigência, armazenando a história das informações, primícia básica dos BDTs, existe uma dificuldade natural quando é tratado um volume considerável de dados, que é o desempenho. Ou seja, o quão eficiente se torna um BDTs para que sua funcionalidade seja pelo menos aceitável. Uma aplicação, por mais funcional que seja, só pode ser considerada eficiente se, e somente se, o tempo de resposta comporta o propósito da aplicação. Isto significa que as técnicas que manipulam grandes volumes de dados devem ser estudadas a parte, para cada tipo de objetivo.

Como ponto de partida é focalizado o problema: otimização de aplicações sobre BDTs. A otimização de uma aplicação, para torná-la viável no seu uso, pode se dar em três camadas: (i) nos programas, (ii) no banco de dados, nível conceitual e (iii) no nível físico. Sem dúvidas que o nível físico é o mais apropriado para esse tipo de objetivo, uma vez que a aproximação das estruturas de dados que são utilizadas para torná-las persistentes e o meio em que isso ocorre é mais estreita. Com isso, ganha-se na portabilidade, fator fundamental para o estudo. Se a otimização for feita na parte física, como por exemplo, na técnica de indexação utilizada, o trabalho fica independente da plataforma. Isto não ocorre quando a otimização é feita no nível conceitual ou nos programas, onde fica explícita a dependência com o modelo utilizado e/ou com as linguagens com as quais se trabalha.

Estudos [SAL97] demonstram que técnicas tradicionais de indexação, especialmente a B^+-Tree [KNU73], são insuficientes para indexar dados temporais. Para suprir esta demanda, técnicas de indexação para dados complexos são alvos de pesquisas. Estas sugerem técnicas de indexação para dados do tipo espaciais, *data mining* e dados multimídia. Entre esses tipos de dados se destacam os dados espaciais, isto porque se pode fazer uma analogia de dados bitemporais com os dados espaciais, possibilitando a utilização de técnicas de indexação de dados espaciais também para dados temporais.

Diversas técnicas específicas de indexação para dados temporais têm sido propostas nos últimos tempos [ELM90, LÜ95, NAS95, NAS96, GOH96, BLI98]. Porém, apesar de todos estes estudos, existem ainda algumas deficiências a serem superadas [LÜ95], a citar: (i) a maioria suporta somente uma dimensão de tempo, (ii) a maioria não distingue ponto no tempo e intervalo no tempo, e (iii) falta de informações de desempenho. É fundamental o desenvolvimento de técnicas de indexação para dados temporais que consigam prover todas as características essênciais a modelos temporais. Como as técnicas propostas possuem pontos comuns, positivos ou negativos, surgiu a idéia de se desenvolver um *framework* conceitual para agregar tais características. Esta poderá servir de base para o surgimento de novas técnicas que superem as atuais: tempo de resposta, forma de armazenamento, contemplação de ambos os tempos (tempo de transação e de validade), portabilidade, entre outras características.

Neste trabalho são analisadas seis técnicas de indexação específicas para dados temporais, das mais diferentes origens e formas: Time Index [ELM90], GR-Tree [BLI98], NB^+-Tree [LÜ95], NR-Tree [LÜ95], B^+-Tree para Dados Temporais [GOH96] e M-ITVT [NAS96]. Com base nesse estudo e nas técnicas tradicionais de indexação como B^+-Tree e B-Tree, e ainda R-Tree, foi desenvolvida uma estrutura capaz de prover as características mais pertinentes para o desenvolvimento de novas técnicas para dados temporais.

O objetivo do trabalho é propor um *framework* conceitual [JOH92, JOH97, GAM94] para ser utilizado por pesquisadores e/ou implementadores de BDTs na criação de nova técnicas de indexação temporal. Algumas vantagens para a utilização deste *framework* podem ser citadas, por exemplo, (i) padronização dos conceitos de indexação temporal, (ii) independência de plataforma, (iii) diagramação universal [KOB99, BOO99, UML99], (iv) possibilidade de inclusão de novas funcionalidades, (v) informações de desempenho dos métodos de indexação, entre outras.

As características das técnicas de indexação, tanto das básicas como das específicas para dados temporais, foram expressas sobre um modelo hierárquico de classes no padrão UML [KOB99, BOO99, UML99]. Essa metodologia foi escolhida por apresentar uma grande variedade de diagramas capaz de contemplar todos os tipos de sistemas, como abordado no capítulo três. Além desse ponto, o fato da padronização ser direcionada para o paradigma da orientação a objetos garante um reaproveitamento dos atributos e métodos através do seu principal conceito, a herança. Assim, um modelo bem estruturado e amplamente hierárquico faz com que seja oferecida uma variação grande de conceitos para a criação de uma nova técnica de indexação para dados temporais.

Uma visão global dos modelos correspondentes às técnicas estudadas individualmente, reunidas em um modelo unificado, garante a possibilidade de agregação de novos conceitos ao estudo, como por exemplo, informações de desempenho dos índices.

Para validar a estrutura proposta foi projetada a especificação de um índice para o modelo de dados TF-ORM [EDE94]. A eficiência do índice só poderá ser comprovada no momento em que testes exaustivos forem feitos, implementando o índice de fato.

Portanto, o objetivo desse trabalho é disponibilizar um *framework* conceitual capaz de auxiliar na criação de novas estruturas de indexação temporal. Inicialmente é feito um estudo amplo das técnicas de indexação para BDTs, envolvendo as possibilidades para viabilizar o uso de uma aplicação. É definido um modelo, no padrão UML, dos métodos pertinentes à indexação de dados, tanto dos básicos como dos específicos para dados temporais. De posse desses modelos individuais, é definido um modelo unificado, com o objetivo de possibilitar a criação de novas técnicas que contemplem melhor os dados temporais. Este modelo constitui o *framework* conceitual proposto. Para validar a idéia de como pode ser utilizada essa estrutura, é exemplificada a criação de uma estrutura de indexação para o TF-ORM.

O trabalho está dividido como segue.

Para maiores entendimentos do ambiente dos bancos de dados temporais e sua complexidade quando da manipulação dos dados, é feita, no capítulo dois, uma breve

explicação do funcionamento de um banco de dado temporal, bem como sua categorização quanto ao tipo, característica fundamental para a escolha de um índice. É apresentada a manipulação dos dados na ótica das operações que regem sobre eles, inclusão, remoção, alteração e consulta. Neste capítulo também é apresentado o foco do problema, onde são abordados os níveis onde pode ser feita a otimização das aplicações e a justificativa da escolha do estudo das técnicas de indexação.

A metodologia UML é abordada no capítulo três, onde são apresentadas as justificativas da escolha desta metodologia e seus principais conceitos, os quais serviram de base para o trabalho.

No quarto capítulo são apresentadas duas técnicas consagradas de indexação para dados lineares. Estas técnicas, denominadas B-Tree e B^+-Tree, são de fundamental importância por serem bases de muitas outras subseqüentes a elas. É apresentada também a técnica R-Tree, que faz uso de algoritmos para indexação de dados espaciais. Faz-se necessário este estudo por existir uma grande semelhança entre dados temporais e dados espaciais.

No capítulo quinto é apresentado um amplo estudo das técnicas específicas para dados temporais as quais serviram de base para o cerne do trabalho. São apresentados os conceitos básicos de cada técnica, seu funcionamento, características mais relevantes e uma proposta de modelo no padrão UML.

O capítulo seis apresenta a proposta deste trabalho, o *framework* conceitual constituído a partir das técnicas vistas no capítulo cinco. São analisadas as características do modelo, vantagens, desvantagens e seu funcionamento.

No capítulo sete é apresentado um estudo de caso referente ao *framework* conceitual aplicado ao modelo TF-ORM. Não é objetivo garantir a eficiência do índice, apenas a sua especificação.

As conclusões e trabalhos futuros são apresentados no capítulo oito, onde é abordado todo o panorama do trabalho.

2 Banco de dados temporais

Este capítulo tem o intuito de mostrar os conceitos temporais relevantes ao estudo das técnicas de indexação para dados temporais. Em especial, são apresentados os tempos de validade e tempos de transação, bem como a sua ortogonalidade e seus conceitos de funcionamento. É apresentada também, a classificação dos bancos de dados temporais. Estas características são essenciais para a escolha do índice, uma vez que cada tipo de estrutura é baseado nas características dos tipos de BDTs ora classificados.

Em seguida são apresentados alguns aspectos da gerência dos dados temporais, sendo feita uma rápida explicação de como funcionam as quatro operações de manipulação dos dados, (i) inserção, (ii) remoção, (iii) alteração e (iv) consulta dos mesmos. Esta explicação se torna relevante porque todas essas operações têm como conseqüência a manipulação de índices sobre os dados, fator objetivo do trabalho.

Como consequência da análise dos aspectos temporais relevantes para a determinação dos índices para este tipo de banco de dados e da influência direta no comportamento dinâmico dos dados governados pelas suas operações, pode-se determinar o problema da otimização de dados temporais e focalizar o objetivo do estudo.

2.1 Representação de aspectos temporais

Segundo [EDE94], a modelagem de um sistema de informação deve representar tanto a estrutura de dados, como a sua evolução ao longo do tempo. Esta dinâmica ao longo do tempo deve ser identificada ainda na fase de especificação de requisitos.

Em SGBDs convencionais, o mundo real é descrito através dos estados **presentes** das estruturas de representação desta realidade no BD. Desta forma, quando ocorre uma mudança no comportamento de um certo atributo, este é atualizado sobrepondo o novo valor ao valor original, sem levar em consideração aspectos temporais de ambos os valores.

Em BDTs, a atualização se faz, na verdade, requerendo uma inserção de um novo valor, para que não seja perdido o seu valor original, preservando a história da informação. Rótulos temporais são adicionados às informações, identificando sua validade.

O termo temporal refere-se ao passado, presente e futuro; informação temporal ou dado temporal inclui duas partes: (i) o estado de uma entidade, (ii) e o tempo da informação na qual o estado é verdadeiro. Por exemplo, “T.S. tem 45 anos” é verdadeiro somente no ano específico e os valores usados para descrever o “tempo” são chamados de informação temporal.

A implementação do conceito de tempo, para determinar os aspectos temporais que vão governar o BDT, pode ser feita de três formas:

- o controle dos dados temporais é realizado explicitamente pelo usuário. O SGBD foi projetado somente para armazenar dados tradicionais do tipo inteiro, *string*, reais, entre outros. Todo o entendimento de tempo está contido nos programas de aplicação. Neste nível, é de total responsabilidade do usuário controlar a lógica de tempo; para isto é preciso conhecer a sua semântica dentro da aplicação e assegurar a validade das operações sobre os dados temporais;

- a manipulação dos dados temporais é realizada por meio de ações associadas a propriedades definidas como temporais. Isto corresponde a extensões semânticas de tipos de dados normais. Esta solução pode ser aplicada em SGDB's extensíveis pela definição de ações semânticas associadas a tipos de dados temporais. Neste caso todas as aplicações compartilham o código associado aos novos tipos de dados; e

- as propriedades temporais são tratadas por uma extensão do modelo de dados e da linguagem de manipulação. Assim, o entendimento de tempo se torna um aspecto estrutural, isto é, a semântica temporal é agregada ao modelo de dados e, portanto, independe das alterações feitas pela aplicação. Por estas propriedades estarem em uma extensão do modelo de dados, a sua identificação é feita na fase de especificação de requisitos.

Sem dúvidas que, para um maior aproveitamento dos aspectos temporais, o último nível é o mais indicado, visto que o entendimento do tempo fica inerente às modificações feitas pela aplicação, obtendo-se uma maior segurança nas operações e uma manutenção facilitada. Talvez a principal desvantagem deste nível seja a necessidade de ser desenvolver uma nova versão do SGBD incluindo as extensões, ou até mesmo, um *middleware* que faça o mapeamento dos aspectos temporais do modelo de dados para o SGBD.

2.1.1 Tempo de transação e tempo de validade

A semântica de tempo, em aplicações de banco de dados pode ser entendida de três formas [SNO85, 86]: (i) tempo de transação (TT), (ii) tempo de validade (TV) e (iii) tempo definido pelo usuário. O tempo de transação é facilmente entendido, pois consiste no instante de tempo em que a transação é efetivada no BD. Entende-se por transação efetivada, a conclusão da mesma sem ferir o conceito ACID1[1]. O tempo de validade consiste no intervalo de tempo em que uma informação é valida. Tempo definido pelo usuário são conceitos de tempo pertinentes somente àquela aplicação, e são definidas explicitamente pelo usuário em um certo domínio temporal; são controlados pelos programas de aplicação.

É consenso entre os pesquisadores de BDTs que os tempos de validade e tempos de transação são ortogonais, isto é, tratados independentemente um do outro.

[1] ACID: atomicidade, consistência, integridade e durabilidade

Existem algumas formas de representação do tempo no BD. Basicamente esta representação depende do elemento temporal utilizado no modelo de dados. Em geral, este elemento consiste em:

- instante no tempo - o tempo de validade é representado através de um ponto no tempo indicando o início da validade, permanecendo o valor válido até que inicie o tempo de validade de um outro valor;

- intervalo temporal - neste caso o intervalo de tempo de validade fica naturalmente representado pelo próprio intervalo temporal. Existem implementações [HÜB2000, EDE2000] em que o ponto no tempo pode ser representado por um intervalo temporal; e

- elemento temporal - formado por um conjunto disjunto de intervalos temporais. Por exemplo, t1 U t2 U t4, sendo t1, t2 e t4, intervalos temporais.

2.1.2 Tipos de banco de dados temporais

Um banco de dados temporal corresponde a um BD que possui alguma forma de representar aspectos temporais [EDE94]. No início dos anos 90 houve uma unificação de termos para BDTs, cuja versão atualizada pode ser encontrada em [JEN98]. Isto ocorre porque pesquisadores da área utilizavam o mesmo conceito com terminologias diferentes, o que dificultava o entendimento. Um dos exemplos desta unificação foi a classificação dos tipos de BDTs, primeiramente proposta por [SNO85]. Os critérios da classificação foram mantidos, mas a sua denominação sofreu algumas alterações, apresentada em [JEN98].

Basicamente, esta classificação fica em torno da forma utilizada para armazenar informações temporais, tempo de validade e tempo de transação, a saber:

- banco de dados instantâneos;
- banco de dados de tempo de transação;
- banco de dados de tempo de validade; e
- banco de dados bitemporais.

A utilização de um modelo temporal para especificação de requisitos não implica na utilização de um SGBD específico para o modelo [HÜB2000, 2000a]. Bancos de dados comerciais podem perfeitamente ser utilizados para representar uma aplicação modelada sob aspectos temporais, desde que para tal exista uma estrutura capaz de mapear o modelo para o SGBD de forma adequada.

2.1.2.1 Banco de dados instantâneo

Banco de dados instantâneos são representados através de BD tradicionais, onde o estado do BD está nos únicos valores armazenados. Em cada modificação no valor de

uma propriedade, a informação antiga é sobreposta pela informação atual, não tendo qualquer estrutura de armazenamento histórico da informação.

A representação temporal, nestes casos, só é possível mediante controle explícito dos programas de aplicação. São adicionados atributos sobre o domínio temporal para se ter a semântica de tempo, porém esta semântica só é interpretada pelos programas e não pelo SGBD.

2.1.2.2 Banco de dados de tempo de transação

Ao contrário dos BDTs instantâneos, todos os valores definidos neste tipo de banco de dados ficam armazenados permanentemente, sendo válidos a partir do instante de sua definição, constituindo desta forma, um BD histórico. O tempo em que foi efetivada a transação é associado com o valor da informação, por isso é chamado de BD de Tempo de Transação. Este tempo é definido sob um rótulo temporal, e é informado automaticamente pelo SGBD.

Os valores atuais das informações armazenadas e das informações definidas em algum instante no passado, estão sempre associados ao seu rótulo temporal. Portanto, o estado atual do BD é dado pelos últimos valores definidos para cada uma das propriedades, fornecidos pelo rótulo temporal mais atual de cada propriedade. Nestes tipos de BD não é possível fazer alterações de dados passados ou futuros, apenas a recuperação destas informações.

2.1.2.3 Banco de dados de tempo de validade

A associação do tempo de transações às informações muitas vezes não é suficiente para representar a realidade de forma correta. Uma forma de representar com mais naturalidade o mundo real é associar um tempo de validade a cada informação. BDTs de tempo de validade contemplam esta representação.

O tempo de validade corresponde ao tempo em que uma informação é verdadeira no mundo real, podendo ser igual ou diferente ao tempo de transação. Nestes tipos de BDTs não se tem acesso ao tempo em que a informação foi definida no BD (tempo de transação).

O tempo de validade é fornecido pelo usuário, sendo que existem regras para a informação deste tempo. Por exemplo, uma informação não pode ter mais de dois períodos válidos ao mesmo tempo. Ao contrário do BD de tempo de transação, é possível realizar alterações sobre dados do passado, presente e do futuro.

2.1.2.4 Banco de dados bitemporais

Bancos de Dados Bitemporais (BD2T) possuem as características de BD de tempo de validade e de transação, pois armazenam as duas formas de representação

temporal. A história da informação fica armazenada sobre todos os prismas, tanto quanto à sua validade como aos instantes de sua criação.

É possível recuperar valores atuais, passados e futuros, tanto para o atual estado de validade como para outros estados de validade, e ainda obter combinações dos tempos de criação (tempo de transação). O estado atual do BD é definido pelos valores válidos de cada propriedade. Um dos fatores mais importantes do BD2T é a evolução das suas formas de representações de tempo de forma ortogonal, ou seja, independente uma da outra.

2.2 Manipulação dos dados temporais

As operações de manipulação sobre os dados são fundamentais em um BD. A gerência dos dados é feita através de uma linguagem associada a cada modelo de dados, e deve propiciar a manipulação dos mesmos da melhor forma possível, tanto em funcionalidade como em desempenho.

Para atingir este objetivo em BDTs, a tarefa é mais complexa do que em BDs tradicionais, tendo em vista o volume das informações e a semântica do tempo em relação a cada aplicação, ou seja, o significado da evolução dos dados ao longo do tempo. Existem particularidades em cada operação da gerência de dados em BDTs.

2.2.1 Inserção em BDTs

Esta operação em BD tradicionais é vista como uma transação relativamente simples porque não envolve a história da informação que está sendo inserida, apenas o seu valor no exato momento. Algumas precauções básicas são tomadas, tais como: chaves primárias, integridade referencial, integridade de entidade, entre outras.

Para se ter um significado de tempo em BDs tradicionais, pode-se utilizar vários artifícios. O mais comum é o conceito de tempo mapeado em atributos. Nestes casos o controle fica a cargo da aplicação e não do SGBD.

Em BDTs, a operação de inserção pode causar grandes preocupações além daquelas usuais em BD tradicionais. Tudo isso, porque existe um significado de tempo no contexto da aplicação, dependendo do tipo de BDTs que está sendo utilizado. Quando uma operação de inserção é realizada, deve ser armazenado, além do novo valor em questão, o valor temporal correspondente ao tempo de transação e/ou validade inicial, da tupla. Assim, um conjunto de regras analisa esses valores, para garantir a consistência temporal do banco de dados. A questão do TT é relativamente fácil de ser resolvida, pois é avaliado apenas se o exato momento em que o dado foi transacionado no BD é válido no domínio temporal. Já o TV deve ser fornecido pelo usuário, sendo seu controle mais complexo. Por exemplo, não se pode ter dois valores válidos ao mesmo tempo para o mesmo atributo. Em [HÜB2000, 2000a] pode ser encontrado um estudo detalhado de gerenciamento destas operações.

2.2.2 Remoção em BDTs

Uma remoção em BD tradicionais é feita através da exclusão física da tupla, levando-se em consideração o conceito de transação. Uma vez feita esta operação, ela não pode ser desfeita. Isto implica que a informação não pode ser recuperada posteriormente. Enquanto na inserção devem ser observadas pelo menos três regras básicas, chave primária, integridade referencial e integridade de entidade, na remoção de dados a principal regra que rege esta operação é a integridade referencial: não se pode remover uma tupla que está relacionada com tuplas subordinadas a ela.

Já em BDTs, esta operação é tratada de forma diferente, pois deve ser fornecida a possibilidade de recuperar informações tanto do passado, quanto do presente e futuro. Assim, a remoção em BDTs consiste em uma remoção lógica: o tempo de validade da informação é encerrado. Todos os valores permanecem sempre armazenados.

Porém, a remoção pode ser um mecanismo importante quando se deseja excluir fisicamente do banco de dados informações muito antigas ou sem relevância. Esta operação recebe o nome de *vaccuming* e é de responsabilidade do administrador do BD (DBA).

2.2.3 Atualização em BDTs

Em BD tradicionais, a atualização inspira alguns cuidados, tais como: a recuperação da tupla correta, chaves primárias, integridade referencial, integridade de entidade, concorrência de acesso a dados modificados, entre outros. Como ocorre nas outras operações da gerência de dados para BDTs, além dos cuidados tradicionais, existe a semântica do tempo.

A atualização em BDTs compreende sempre pelo menos duas operações: (i) o encerramento do tempo de validade da informação anteriormente válida, e (ii) a inserção de uma nova tupla, com o novo valor e os rótulos temporais adequados. Quando é utilizado o TV cuidados adicionais devem ser tomados pelo SGBDT para garantir a integridade temporal. Um estudo detalhado das diversas situações de erro que podem ocorrer, e de como o SGBDT pode solucioná-las, pode ser encontrado também em [HÜB2000, 2000a].

2.2.4 Consultas em BDTs

A recuperação de informações é de vital importância em BDs. Cada modelo de dados deve apresentar uma linguagem de recuperação de informações associada às características do modelo [EDE94].

A fundamental diferença da recuperação de dados entre BDs tradicionais e BDTs, é que neste último o tempo sempre está presente na consulta, mesmo que implicitamente. Desta forma, pode-se ter três situações diferentes na recuperação de informações [EDE94]: (i) recuperar valores de propriedades cujo domínio é temporal; (ii) se referir a um determinado instante ou intervalo temporal; e (iii) recuperar valores com base em restrições temporais.

Dependendo do tipo de BDT, é possível fazer diferentes tipos de consultas, de acordo com o TT e/ou TV em questão. A tabela 2.1 apresenta algumas características relativas à consulta.

TABELA 2.1 - Possibilidade de Consulta de Acordo com o Tipo de BDTs

Banco de Dados Instantâneos	Como não representam aspectos temporais, não permitem consultas temporais.
Banco de Dados de Tempo de Transação	Além de recuperar valores atuais, podem recuperar valores definidos no passado. Esta possibilidade se deve ao fato de que o instante da transação é armazenado juntamente com o seu valor, fazendo um histórico das informações.
Banco de Dados de Tempo de Validade	Utiliza o TV, que possui uma maior veracidade em relação ao tempo de transação. Através deste tempo se pode recuperar: (I) valores atuais, (ii) valores válidos no passado e no futuro.
Banco de Dados Bitemporal	Combinam o TT e TV. Recuperam informações de valores atuais, do passado e do futuro, tanto para o estado atual de validade como para outros estados de validade do passado.

Para se recuperar informações que envolvam tempo, conforme mostra a tabela 2.1, é preciso que a linguagem de consulta contemple alguns operadores que manipulem os aspectos temporais.

2.3 Determinação do problema

Um aspecto importante a ser focalizado é o problema de otimização sobre BDTs. Um dos fatores vitais para a satisfação do usuário, além da funcionalidade do sistema, é o desempenho. Para se obter um desempenho satisfatório em sistemas de BD deve-se ter como principal requisito o nível físico (figura 2.2), mais precisamente, as estruturas de armazenamento e o quanto o sistema é capaz de operar sobre estas estruturas de armazenamento.

Quando se estuda uma estrutura de armazenamento eficiente, dois parâmetros são avaliados, o espaço e o tempo de resposta. Em BDTs, toda a história das informações é armazenada. Deduz-se então que existe um grande volume de dados. O acesso a estes dados de forma correta e rápida é fundamental. Para viabilizar este acesso são definidas técnicas de indexação de dados. Estas técnicas de indexação para dados temporais devem ocupar o mínimo de espaço possível, já que, por natureza, tem-se um grande volume de dados já armazenados.

Abstraindo-se um pouco mais o nível conceitual (figura 2.2), pode-se obter a forma pela qual uma aplicação faz o acesso aos esquemas de BD. Este canal é denominado de linguagem de manipulação de dados. Portanto, o caminho natural da aplicação até o nível físico consiste em passar por: (i) uma linguagem de manipulação de dados, para acessar (ii) os esquemas do BD; este por sua vez, através de seu sistema gerenciador, se comunica com (iii) as estruturas de armazenamento de dados (nível

físico).

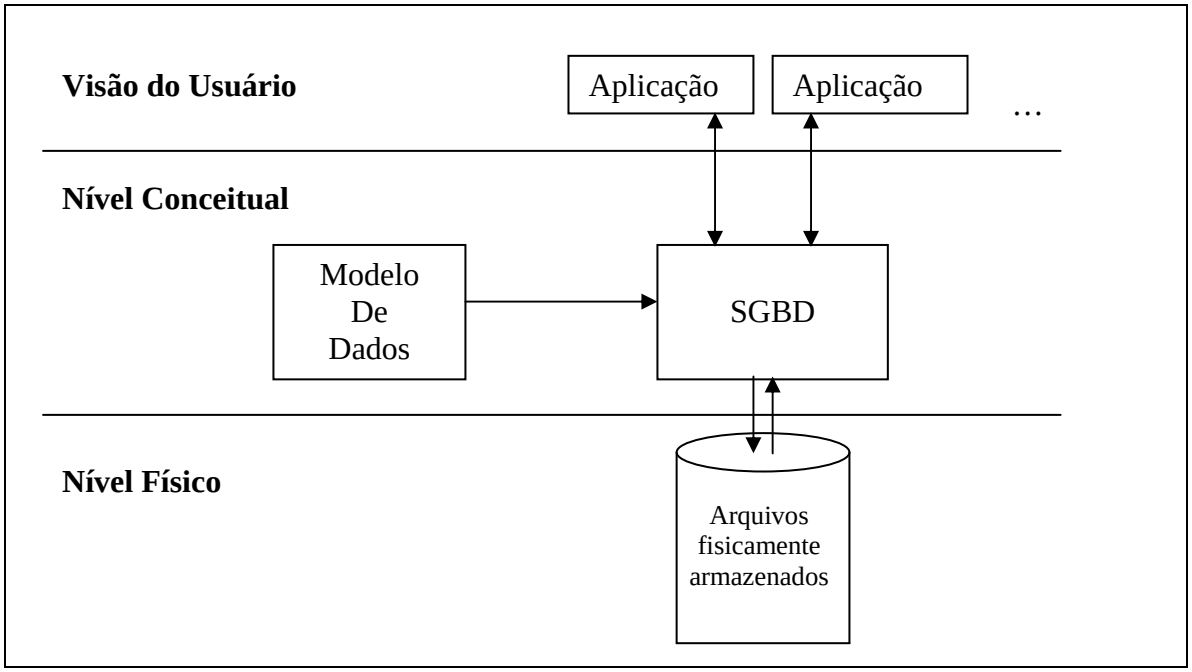


FIGURA 2.2 – Divisão de Níveis (usuário, conceitual e físico)

É possível obter a otimização de uma aplicação partindo-se dos níveis mais próximos da aplicação até chegar ao nível físico. Porém, existem alguns problemas neste caminho:

1. linguagem de manipulação de dados:
   - as linguagens de consultas estão intimamente ligadas aos modelos utilizados;
   - como existem características diferentes em cada tipo de linguagem, a otimização fica dependente dos conhecimentos do programador da aplicação.
2. esquema de BD:
   - a otimização feita no nível conceitual de esquema de BD fica dependente das características dos SGBD.

O nível físico é o mais indicado para se obter uma otimização independente das características dos modelos de dados e dos SGBD. Além disso, existe outro fator fundamental, que é a problemática do *hardware*. Sem dúvidas que o *hardware* em questão, tipicamente o disco rígido, é objeto complicador de uma otimização. Fazer com que sejam aproveitadas características deste componente é parte fundamental, e nada melhor do que trabalhar no meio físico para atingir este objetivo.

As razões aqui apresentadas foram a motivação do estudo de otimização de aplicações para BDTs, que resultou no estudo de técnicas de indexação para dados temporais.

2.4 Considerações finais

Neste capítulo foram apresentadas as principais semânticas temporais, tais como TT e TV, bem como os tipos de BDTs, classificados em instantâneo, Banco de Tempo de Validade, Banco de Tempo de Transação e Bancos Bitemporais. Frente a cada tipo de BDT ora classificado, existe um mecanismo próprio para a manipulação dos tempos pertinentes. Estes mecanismos são aspectos funcionais que podem complicar a manipulação dos dados, e conseqüentemente, piorar o desempenho destes tipos de banco.

Foi analisada também, a manipulação dos dados de um BDT. Esta manipulação pode e deve piorar o desempenho frente aos índices do BD. Por exemplo, na inserção existe um significado de tempo no contexto da aplicação, dependendo do tipo de BDTs que está sendo utilizado. Quando uma operação de inserção é realizada deve ser fornecido, além do novo valor em questão, o valor temporal correspondente ao tempo de transação e/ou validade inicial, da tupla.

A remoção de dados em BDTs é de fundamental importância, pois é tratada de forma completamente diferente da remoção de BDs tradicionais. Isto porque em BDTs deve ser fornecida a possibilidade de recuperar informações tanto do passado, quanto do presente e do futuro.

As alterações de dados necessitam de algumas regras funcionais. Estas regras visam assegurar a consistência das informações armazenadas.

Na consulta, a fundamental característica dos BDTs é que o tempo sempre está presente, mesmo que implicitamente, ou seja, a otimização desse tipo de banco de dados se torna necessária para o bom funcionamento do mesmo.

Com esse panorama, é possível prever que uma estrututura de índices para dados temporais deve ser mais complexa do que para dados tradicionais e que uma possível adequação seja necessária para contemplar esta complexidade.

3 Unified Modeling Language (UML) – notação gráfica

Este capítulo aborda os conceitos básicos da representação UML [KOB99, BOO99, UML99] que foram utilizados nos modelos ao longo do trabalho, bem como um breve histórico do surgimento desta notação. É justificada, também, a sua escolha.

Os sistemas orientados a objetos podem representar melhor o mundo real, uma vez que a percepção e o raciocínio do ser humano estão relacionados diretamente com o conceito de objetos. Este fato permite uma modelagem mais próxima do real e natural. Por esta simplicidade de abstração e principalmente, por alguns conceitos da orientação a objetos, tais como hierarquia, polimorfismo, e da facilidade de reutilização das classes em outras estruturas, foi escolhido o conceito da orientação a objetos para dar suporte a este estudo. Além disso a notação UML se tornou um padrão da OMG (*Object Management Group*).

Existem inúmeras vantagens para se modelar uma estrutura apoiada na orientação a objetos, porém algumas se tornam decisivas, a citar:

- esta abordagem incentiva os desenvolvedores a trabalharem e pensarem em termos do domínio da aplicação durante a maior parte do desenvolvimento, ou seja, proporciona dedicação maior à fase de análise;

- ocorre uma redução na quantidade de erros com conseqüente diminuição do tempo despendido nas etapas de codificação e testes, visto que os problemas são detectados mais cedo e corrigidos antes da implementação;

- no desenvolvimento baseado em objetos há uma redução no tempo de manutenção, pois as revisões são mais fáceis e mais rápidas já que o problema é melhor localizado;

- favorece a reutilização, já que ocorre a construção de componentes mais gerais, estáveis e independentes; e

- facilidade de extensão, visto que objetos têm sua interface bem definida. A criação de novos objetos que se comunicam com os já existentes não obriga o desenvolvedor a conhecer o interior destes últimos.

Para representar melhor os conceitos que este paradigma oferece, utilizou-se a técnica de modelagem UML - *Unified Modeling Language*.

3.1 Modelagem de sistemas em UML

Em todos os ramos de *software*, os desenvolvedores estão constantemente preocupados com o planejamento, com a análise de como as tarefas serão criadas e desenvolvidas de forma rápida, otimizada e precisa. A modelagem é uma técnica de engenharia aprovada e bem-aceita que visa auxiliar esse planejamento. Consiste em criar modelos que são **simplificações da realidade** e ajudam a compreender melhor o problema. No caso deste estudo, a modelagem representa a simplificação individual de técnicas de indexação, tanto das básicas como das específicas para BDTs. Dividindo e separando o foco do problema, fazendo modelagens das técnicas separadamente, garante-se maior qualidade ao estudo. Uma visão global do modelo proposto engloba estas técnicas individuais.

O *framework* conceitual proposto neste trabalho é representado no padrão UML. O grande problema do desenvolvimento de novos sistemas utilizando a orientação a objetos nas fases de análise de requisitos, é que não existe uma notação padronizada e realmente eficaz que abranja qualquer tipo de aplicação que se deseje. O padrão UML engloba um conjunto de conceitos capaz de suprir esta dificuldade.

3.2 Surgimento da UML

As linguagens de modelagem orientadas a objetos surgiram entre a metade da década de 1970 e o final da década de 1980, diante de um novo gênero de linguagens de programação orientadas a objetos e de aplicações cada vez mais complexas. Um conjunto de novas metodologias surgiu de forma desordenada, causando um colapso no entendimento entre elas.

A UML (*Unified Modeling Language*) [KOB99, BOO99, UML99] é uma tentativa de padronizar a modelagem orientada a objetos de forma que qualquer sistema, seja qual for o seu tipo, possa ser modelado corretamente, com consistência, sendo fácil de se comunicar com outras aplicações, simples de ser atualizado e compreensível.

UML acabou com esta guerra trazendo as melhores idéias de cada uma das diferentes metodologias propostas, e mostrando como deveria ser a migração de cada uma para a sua representação. Os esforços para a criação da UML iniciaram oficialmente em outubro de 1991, quando Rumbaugh se juntou a Booch. O foco inicial do projeto era a unificação dos métodos Booch [BOO94,BOO96] e *Object Modeling Techinique (OMT)* [RUM91]. O esboço da versão 0.8 do Método Unificado (como então era chamado) foi lançado em outubro de 1993. Mais ou menos na mesma época, Jacobson se associou à Rational e o escopo do projeto da UML foi expandido com a finalidade de incorporar o *Objec-Oriented Software Engineering (OOSE)* [JAC92]. Os esforços dos três pesquisadores resultaram no lançamento dos documentos da versão 0.9 da UML em junho de 1996. Durante 1996 foi criado um consórcio de UML, entre a Rational e várias empresas que desejavam dedicar recursos com o propósito de trabalhar a favor de uma definição mais forte e completa da UML. Assim, a UML tem como principais colaboradores as idéias de Booch [BOO94,BOO96], Rumbaugh [RUM91] e Jacobsen

[JAC92].

UML é uma linguagem-padrão para a elaboração da estrutura de projetos de *software*. Pode ser empregada para a visualização, a especificação e a construção da documentação de artefatos que façam uso de sistemas complexos.

A UML é adequada para a modelagem de sistemas cuja abrangência poderá incluir sistemas de informação, corporativos, distribuídos e aplicações baseadas em *web* e até sistemas completos de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas.

Os objetivos da UML são:

- a modelagem de sistemas (não apenas de *software*) usando os conceitos da orientação a objetos;
- estabelecer uma união, fazendo com que métodos conceituais fiquem totalmente executáveis; e
- criar uma linguagem de modelagem utilizável tanto pelo homem quanto pela máquina.

3.3 Utilização da UML

A UML não está restrita apenas à modelagem de *software*. Na verdade, a UML é suficientemente expressiva para modelar sistemas que não sejam simples, como aplicativos tradicionais.

É utilizada no desenvolvimento dos mais diversos tipos de sistemas. Abrange sempre qualquer característica de um sistema em um de seus diagrama, sendo aplicada em diferentes fases do desenvolvimento, desde a especificação da análise de requisitos até a finalização com a fase de testes.

O objetivo da UML é descrever qualquer tipo de sistema, em termos de diagramas orientados a objetos. Naturalmente, o uso mais comum é para criar modelos de sistemas de *software*, mas a UML também é usada para representar sistemas mecânicos sem nenhum *software*.

Exemplos de sistemas que podem ser representados em algum dos diagramas da metodologia UML, e de duas funcionalidade [UML2000]:

- Sistemas de Informação - armazenar, pesquisar, editar e mostrar informações aos usuários. Manter grandes quantidades de dados com relacionamentos complexos, que são guardados em bancos de dados relacionais ou orientados a objetos;

- Sistemas Técnicos - manter e controlar equipamentos técnicos como de telecomunicações, equipamentos militares ou processos industriais. Devem possuir interfaces especiais do equipamento e menos programação de *software* de que os sistemas de informação. Sistemas Técnicos são geralmente sistemas *real-time*;

- Sistemas *Real-time* Integrados - executados em simples peças de *hardware* integrados a telefones celulares, carros, alarmes, entre outros. Estes sistemas implementam programação de baixo nível e requerem suporte *real-time*;

- Sistemas Distribuídos - distribuídos em máquinas onde os dados são transferidos facilmente de uma máquina para outra. Eles requerem mecanismos de comunicação sincronizados para garantir a integridade dos dados e geralmente são construídos em mecanismos de objetos como CORBA, COM/DCOM ou J-Beans/RMI;

- Sistemas de *Software* - definem a infra-estrutura técnica que outros *softwares* utilizam. Sistemas Operacionais, bancos de dados e ações de usuários que executam em baixo nível no hardware, ao mesmo tempo em que disponibilizam interfaces genéricas de uso de outros *softwares;* e

- Sistemas de Negócios - descrevem os objetivos, especificações (pessoas e computadores), as regras (leis e estratégias de negócios) e o atual trabalho desempenhado nos processos do negócio.

A possibilidade de aplicação em sistemas de *software* mostra a possibilidade de se utilizar este metodologia neste trabalho. É importante perceber que a maioria dos sistemas não possui apenas uma destas características acima relacionadas, mas várias delas ao mesmo tempo. Sistemas de Informações de hoje, por exemplo, podem ter tanto características distribuídas como *real-time* e/ou as características de sistemas de *software*, e a UML suporta modelagens de todos estes tipos de sistemas.

3.4 Componentes básicos das UML

Classe – as classes são os blocos de construção mais importante de qualquer sistema orientado a objetos. Uma classe é uma descrição de uma conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. É utilizada para abstrair o sistema que está sendo desenvolvido, dependendo do nível de entendimento. Por exemplo, podem existir classes que representem itens de *software*, de *hardware* e até itens que sejam puramente conceituais. A sua identificação é dada por um nome unívoco no modelo. Geralmente são substantivos ou expressões breves que identificam uma parcela conceitual da realidade. A representação gráfica de uma classe é apresentada na figura 3.1.

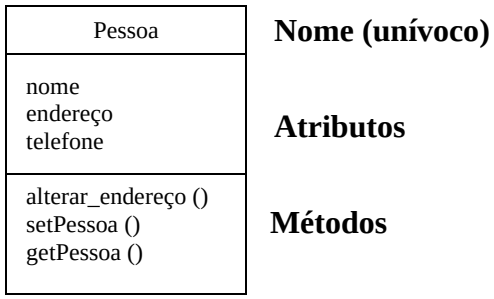


FIGURA 3.1 – Classe em UML

Atributo - representados na parte do meio da classe, os atributos são propriedades que descrevem os intervalos de valores que as instâncias da classe podem apresentar. Uma classe pode ter qualquer número de atributos ou até mesmo nenhum atributo. São compartilhados por todos os objetos instanciados nesta classe. Eles podem ser representados apenas por seu nome ou por seu nome e tipo de valores a que pertencem estes atributos.

Método – métodos ou operações correspondem à implementação de um serviço que pode ser executado por algum objeto da classe, representando seu comportamento. Em outras palavras, uma operação é uma abstração de algo que pode ser feito com um objeto e que é compartilhado por todos os objetos dessa classe. A classe pode ter inúmeros métodos ou até mesmo nenhum método. Os métodos podem ser representados exibindo somente seus nomes ou indicando sua assinatura, que contém o nome, o tipo e o valor-padrão de todos os parâmetros e, no caso de funções, o tipo a ser retornado. Quando a operação não possui nenhum parâmetro a ser passado, são representados apenas pelos parênteses logo após o nome da operação, conforme figura 3.1.

Relacionamento – ao decorrer da modelagem é notório que pouquíssimas classes trabalham isoladas. A maioria colabora entre si de várias formas. Portanto, ao fazer a modelagem de um sistema, será necessário não somente identificar os itens que formam o vocábulo do sistema, mas também modelar como esses itens se relacionam entre si. Em uma modelagem orientada a objetos existem três tipos de relacionamentos que merecem uma maior atenção: (i) dependência, que representa relacionamento de utilização entre as classes; (ii) generalização, que relaciona classes generalizadas a suas especializações; e (iii) associação, que representa relacionamentos estruturais entre objetos. Cada um desses relacionamentos fornece uma forma diferente de combinações e subtipos.

Neste trabalho são utilizados basicamente dois tipos de relacionamentos, a generalização e a associação. Sendo assim, a seguir são explicados e representados graficamente somentes estes dois relacionamentos.

Generalização - é um relacionamento entre itens gerais (chamadas superclasses ou classe-mãe) e tipos mais específicos desses itens (chamadas subclasses ou classes-filhas). Muitas vezes as generalizações são chamadas de relacionamentos “é um tipo de”. A generalização significa que os objetos da subclasse podem ser utilizados em qualquer

local em que a superclasse ocorra, mas não vice-versa. Em outras palavras, a subclasse herda as propriedades da superclasse, principalmente seus atributos e operações (ou métodos). Freqüentemente as subclasses têm atributos e operações além daqueles encontrados nas respectivas superclasses. A operação de uma subclasse, que tenha a mesma assinatura de uma operação da superclasse, prevalecerá em relação à operação da superclasse.

Uma classe que tem exatamente uma única superclasse é identificada como de herança simples (figura 3.2); as classes com mais de uma superclasse apresentam herança múltipla (figura 3.3).

Graficamente a generalização é representada por uma linha sólida apontando a superclasse com um triângulo sem preenchimentos, conforme figura 3.2.

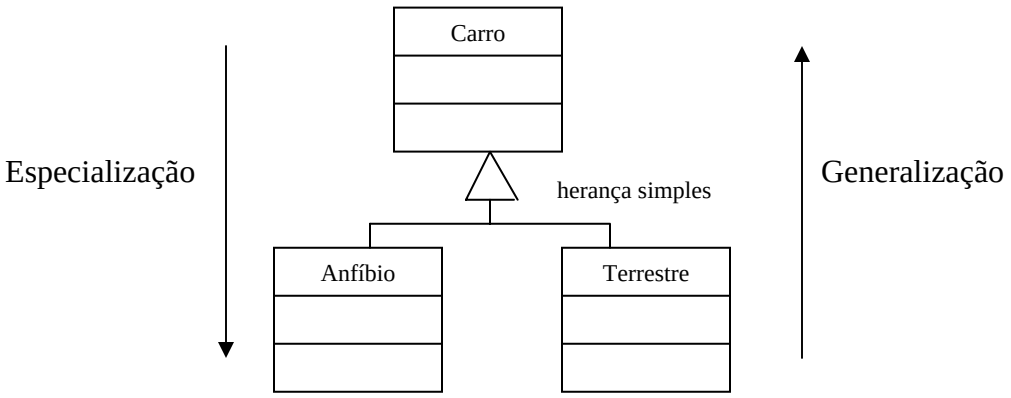


FIGURA 3.2 – Generalização na UML

Toda generalização gera uma abstração contrária que e à **especialização**.

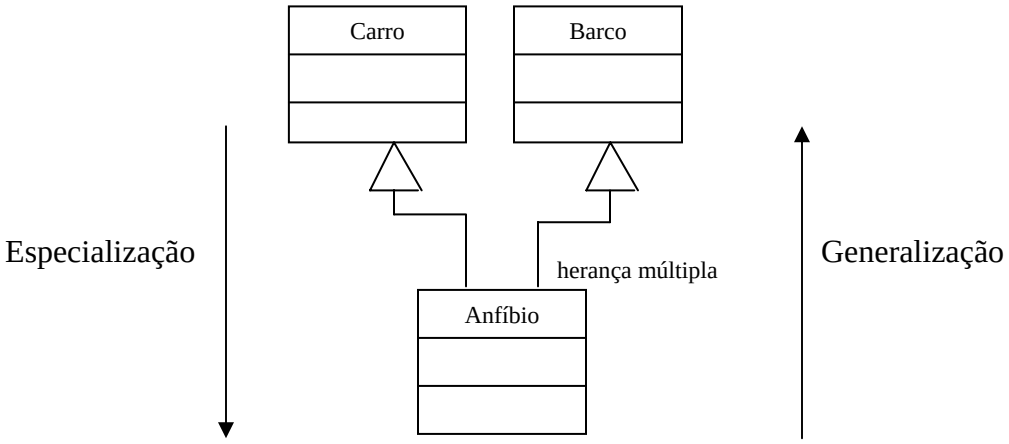


FIGURA 3.3 – Herança Múltipla

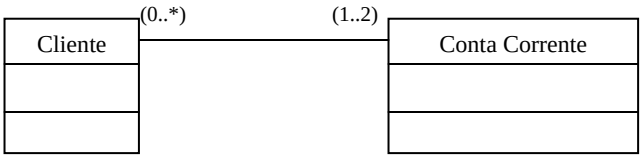


FIGURA 3.4 – Relacionamento do tipo Associação.

Associação - uma associação é um relacionamento estrutural que especifica objetos de um item associados a objetos de outro item. Associações são sempre usadas para garantir consistência entre os objetos.

Graficamente, a associação é representada como uma linha sólida conectando as classes associadas. Pode ou não ter uma nome, para descrever melhor o entendimento do relacionamento. Na figura 3.4 a classe cliente possui uma associação semântica com a classe Conta Corrente.

Multiplicidade - em muitas situações de modelagem, é importante determinar a quantidade de objetos que podem ser determinados pela instância de uma associação (figura 3.4). Essa quantidade é denominada de multiplicidade do papel de uma associação. É escrita como uma expressão equivalente a um intervalo de valores ou a um valor específico. Ao determinar a multiplicidade em uma das extremidades de uma associação, está sendo especificado que para cada objeto da classe encontrada na extremidade oposta, deve haver a mesma quantidade de objetos na próxima extremidade. São exemplos:

- exatamente um (1) ;
- zero ou um (0..1) ;
- muitos (0..*) ;
- um ou Mais (1..*) ;
- número exato (3);
- intervalos (1..5) ; e
- qualquer quantidade, exceto o 2 e o 5 (0..1, 3..4, 6..*) .

Agregação – a agregação indica que uma das classes do relacionamento é uma parte, ou está contida em outra classe. As palavras-chaves usadas para identificar uma agregação são “consiste em”, “contém”, “é parte de”, entre outras. É considerada um tipo especial de uma associação.

Graficamente é representada por um losângulo sem preenchimento na parte da classe que representa o “todo”, conectado por uma linha sólida até a classe a qual representa a “parte”.

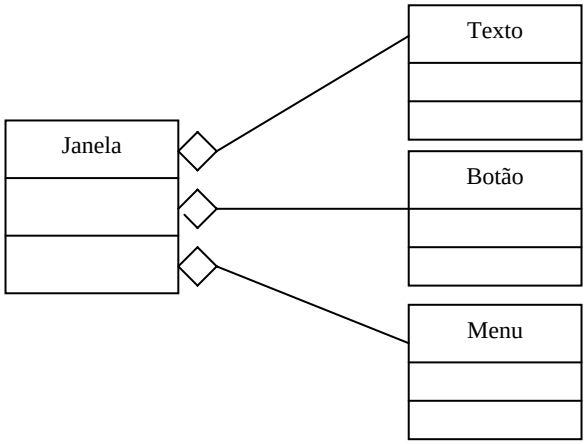


FIGURA 3.5 – Exemplo de Agregação na UML

Como exemplo, a figura 3.5 indica que a classe Janela é composta por Texto, Botão(ões) e/ou Menu, mas que não necessariamente seja composta por só estas três representações conceituais.

Colaborações – é uma sociedade de classes, interfaces e outros elementos que trabalham em conjunto para fornecer algum comportamento cooperativo maior do que a soma de todas as suas partes, ou seja, é um artifício legal de representar uma "caixa preta" sobre um determinado assunto ou visão. É uma especificação de como uma operação é realizada por um conjunto de associações ou especializações, desempenhando um papel específico dentro do contexto. A colaboração é representada graficamente por uma elipse com linhas tracejadas (figura 3.6).

A colaboração tem dois aspectos, (i) uma parte estrutural que especifica as classes, interfaces e outros elementos que seriam representados no modelo, e (ii) uma parte comportamental que especifica a dinâmica de como esses elementos interagem.

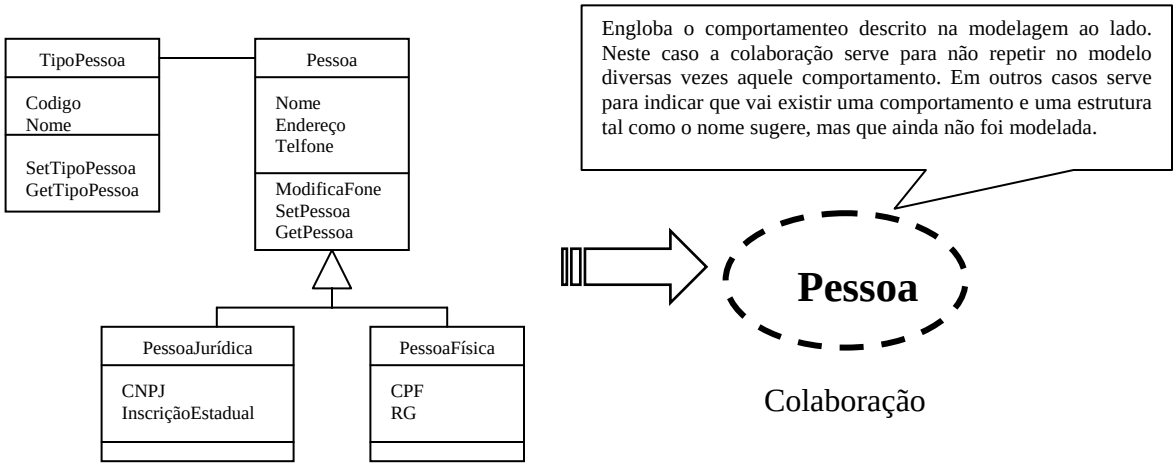


FIGURA 3.6 – Exemplo de Colaboração

Na figura 3.6 a colaboração Pessoa pressupõe um comportamento tal qual descrito no modelo. Porém, existem casos em que a modelagem não está abstraída o suficiente para apresentar a estrutura já definida. Neste caso a colaboração serve como artifício de uma “caixa preta”, podendo existir até mesmo um outro diagrama de classes representando esta colaboração.

3.5 Diagramação da UML

Como a UML é uma unificação de vários conceitos de diferentes metodologias da modelagem orientada a objetos, é natural que esta linguagem de representação apresente uma grande variedade de diagramas. Sua classificação é bem variada e conforme [UML99] é apresenta como segue:

1) Estruturas Estáticas

1.1) Diagrama de Classes - sem dúvidas o mais importante dos diagramas e o mais utilizado. Ele mostra um conjunto de classes, interfaces (item do diagrama que define um comportamento), colaborações e seus relacionamentos. Serve para dar a visão estática do projeto;

1.2) Diagrama de Objetos – representam a modelagem das instâncias de itens contidos em diagramas de classes. Um diagrama de objetos mostra um conjunto de objetos e seus relacionamentos em determinado ponto no tempo;

2) Comportamentais (estruturas dinâmicas)

2.1) Diagrama de Casos de Uso – são diagramas para modelar o comportamento de um sistema, de um subsistema ou de uma classe. Cada um mostra um conjunto de casos de uso e atores e seus relacionamentos;

2.2) Diagrama de Gráfico de Estados - também representam aspectos dinâmicos. Mostram uma máquina de estados dos objetos. Uma máquina de estados representa um comportamento que especifica a seqüência de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos;

2.3) Diagrama de Atividades - serve para modelar aspectos dinâmicos do sistema. É essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra;

2.4) Diagrama de Interação - mostra uma interação formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser trocadas entre eles. Uma interação é um comportamento que compreende um conjunto de mensagens

trocadas entre um conjunto de objetos em determinado contexto para realização de um propósito. Está dividido em:

2.3.1) Sequência – é um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Graficamente, um diagrama de seqüência é uma tabela que mostra objetos distribuídos no eixo X e mensagens, em ordem crescente em tempo, no eixo Y; e

2.3.2) Colaboração – é um diagrama de interação que dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens. Graficamente um diagrama de colaboração é uma coleção de vértices e arcos;

3) Implantação

3.1) Diagrama de Componentes - representam aspectos físicos do sistema. Mostram a organização e as dependências existentes entre um conjunto de componentes. Apresentam itens físicos que residem em um nó, por exemplo: executáveis, biblioteca, tabelas, arquivos, entre outros;

3.2) Diagrama de Implantação - também representam aspectos físicos do sistema. Mostram a configuração de nós de processamento em tempo de execução e os componentes que neles existem. Envolve topologia do *hardware*, entre outros itens físicos.

Existe, portanto, uma grande variabilidade de diagramas podendo representar diversas faces do sistema, não sendo geralmente representados todos para uma mesma modelagem. Em geral, dependendo do tipo de sistemas que se deseja representar, um tipo de diagrama é enfocado. No caso deste trabalho resolveu-se trabalhar com os diagramas de classes por melhor representar a realidade do modelo proposto.

Na figura 3.7 está um exemplo resumido de diagrama de classe com as principais características utilizadas neste trabalho, como especialização e generalização, associação e agregação, representação das classes, seus atributos e métodos.

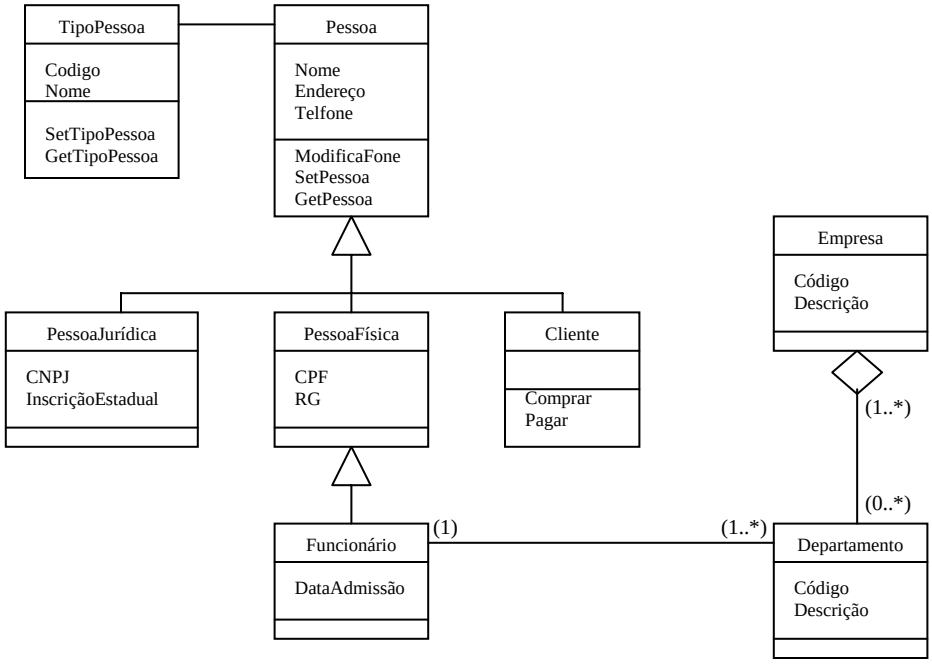


FIGURA 3.7 – Diagrama de Classes Modelo UML [UML2000]

3.6 Considerações finais

Neste capítulo foram apresentados os conceitos fundamentais da UML para o entendimento dos modelos que seguem ao longo do trabalho. Um fato importante é a capacidade da notação UML em diagramar e conceituar qualquer tipo de sistema. Por exemplo, podem ser modelados através da UML *Sistemas de Software* que definem uma infra-estrutura técnica que outros *softwares* utilizam, ou ainda sistemas operacionais, bancos de dados e ações de usuários que executam em baixo nível no *hardware* - o que ampara a escolha desta metodologia neste trabalho.

4 Técnicas-base de indexação

Existem várias propostas de formas de indexação para dados lineares, entre os quais destacam-se as seguintes: B-Tree [BAY70] e B^+-Tree [KNU73]. Em especial uma técnica para dados espaciais se tornou bastante conhecida pelo seu desempenho denominada de R-Tree [GUT84]. Este capítulo tem como objetivo apresentar estas técnicas de indexação. São técnicas consagradas pelo seu alto grau de desempenho e por apresentarem uma estrutura relativamente simples. Seu funcionamento serve de base para muitas outras técnicas, inclusive para as técnicas propostas para dados temporais.

4.1 B-Tree

Um tipo de índice particularmente comum e importante é a B-Tree [BAY70] (Árvore-B). Embora seja verdade que não existe uma única forma de armazenamento ótima para todos os casos, sem dúvida que esta estrutura, ou alguma variação dela, é bastante adequada.

Os nodos de uma árvore do tipo B-Tree (figura 4.1) são compostos por chaves (Xi), ponteiros (Pi) e informações (&i), onde:

- as chaves representam o atributo pelo qual os dados destão sendo indexados (ordenados) os dados;
- os ponteiros são ligações que apontam outro nodo da árvore; e
- informações representam os endereços onde pode ser encontrado o restante da tupla relativa à chave de busca (Xi).

P1	X1	&1	P2	....	Pn-1	Xn-1	&n-1

FIGURA 4.1 - Organização de um nodo em uma B-Tree

A cada chave de busca está associada somente uma informação. A B-Tree é caracterizada por dois parâmetros fundamentais [DAT91], a saber:

- **h:** indica a altura da árvore, ou seja, a distância em nodos entre a raiz e qualquer uma das folhas da estrutura. Esse parâmetro varia em função das operações efetuadas sobre a árvore; e

- **d:** indica a ordem da árvore, isto é, o número mínimo de chaves armazenadas em cada nodo. Esse é o parâmetro mais importante do processo de operações sobre a árvore.

O número de chaves de busca em cada nodo varia entre **d** e 2**d** (onde **d** é a ordem da árvore), com exceção da raiz que varia entre 1 e 2**d**.

A B-Tree aumenta e diminui uniformemente através da divisão de um nodo em dois irmãos, ou através da união de dois nodos irmãos em um só. O processo de divisão do nodo é conhecido como método de *split*. O aumento ou a diminuição da árvore são processos que sempre iniciam nos nodos folhas e podem propagar-se até a raiz. Se o nodo raiz também é afetado com esta operação, então ocorre uma mudança na altura da árvore, podendo aumentar ou diminuir, dependendo da operação. A figura 4.2 exemplifica uma árvore B-Tree.

O balanceamento da B-Tree é uma característica importante. As estrutura de árvores em geral têm um problema chamado desbalanceamento. Diz-se que uma árvore é desbalanceada quando algum dos nodos folhas apontados pelo nodo pai, não se encontra no mesmo nível dos outros nodos folhas, isto é, se nodos folhas diferentes da mesma entrada de nodos encontram-se em diferentes distâncias da raiz até esses nodos. O que não ocorre com a B-Tree devido a eficiência dos algoritmos de inserção, alteração e remoção.

Para simplificar a representação da estrutura da árvore, na figura 4.2 não são representados os campos de informações que correspondem a cada uma das chaves, sendo apenas mostrados os ponteiros e a chave de busca.

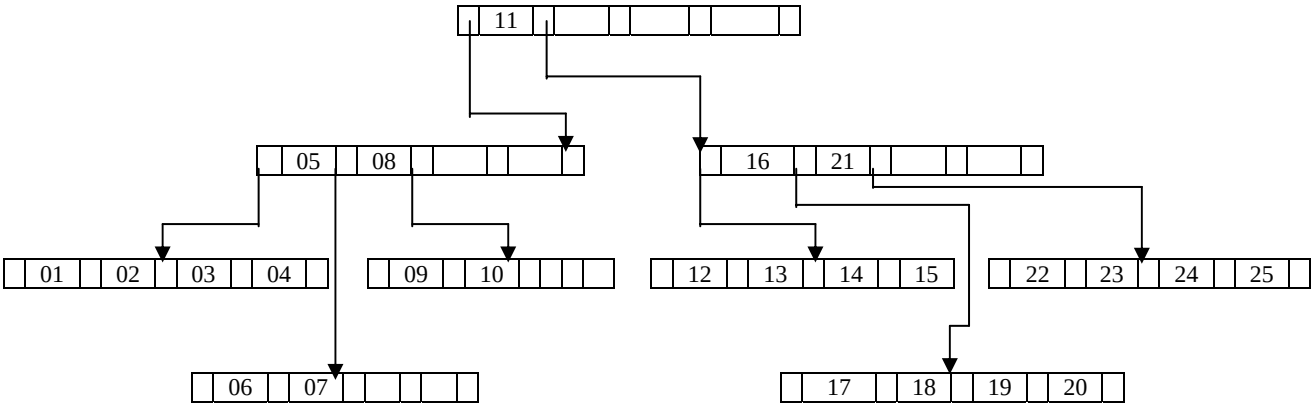


FIGURA 4.2 – Exemplo de uma B-Tree de ordem 2.

A B-Tree é largamente usada por SGBDs comerciais. O que explica esse fato são as vantagens que a B-Tree proporciona [HOR84], por exemplo:

- a memória alocada é liberada a medida em que a árvore se expande ou se contrai. Não há problemas quanto ao congestionamento ou perda de eficiência se a ocupação de memória é muito grande;

- as operações realizadas sobre uma B-Tree têm como conseqüência uma ordenação natural das chaves de buscas, o que permite operações que aproveitam esta característica, como por exemplo encontrar sucessores e predecessores, pesquisas seqüencial, recuperação de um determinado número de registros a partir de uma chave de busca, entre outros exemplos; e

- a principal vantagem da B-Tree situa-se nos métodos de inserção e remoção de registros, que sempre deixam a B-Tree balanceada. O maior caminho de uma B-Tree de **N** chaves de busca, contém no máximo $\text{Log}_{\mathbf{d}}\ \mathbf{N}$ acessos, sendo **d** a ordem da B-Tree.

A característica primordial da B-Tree é o balanceamento natural da árvore. Como no caso de árvores binárias, inserções randômicas de registros em um arquivo podem deixar a árvore desbalanceada. Enquanto que uma árvore desbalanceada tem alguns caminhos longos e outros curtos, uma árvore balanceada possui a mesma distância entre a raiz e TODOS os seus nodos folhas.

Uma operação de pesquisa pode visitar **n** nodos em uma árvore desbalanceada indexando um arquivo de **n** registros. Em árvores balanceadas, a visita nunca ultrapassa mais que $\text{Log}_{\mathbf{d}}\ \mathbf{N}$ nodos para arquivo com **n** registros, onde **d** é a ordem da árvore. Como cada visita requer um acesso à memória secundária para carregar os nodos em memória principal, o fato de balancear a árvore é de suma importância para o ganho de desempenho.

O método de balanceamento da B-Tree utiliza um mecanismo extra para reduzir o custo do balanceamento (acesso à memória secundária tem um custo menor se comparada com o tempo de balanceamento). Portanto, as B-Trees apresentam as vantagens dos métodos de árvores balanceadas, enquanto evitam algum tempo de consumo com a manutenção.

4.1.1 Operações sobre B-Tree

4.1.1.1 Pesquisa

Em uma pesquisa binária, os ramos da árvore percorridos a partir de um nodo, dependem do resultado da comparação entre a chave de busca e a chave requerida. Se a chave requerida é menor do que a do nodo, o ramo da esquerda será o caminho a ser seguido; se é maior, então o ramo direito será seguido. Exemplificando, a figura 4.3, mostra uma B-Tree com duas chaves armazenadas em cada nodo; se o valor pesquisado for menor que 42, então deve ser seguido o ramo da esquerda; se o valor requerido for entre as chaves 42 e 81, o ramo central deve ser selecionado; e para as chaves maiores que 81, o ramo mais a direita é o indicado. Por exemplo, é requerido o valor 15, verifica-se que ele é menor que 42, então se toma o ramo da esquerda. No passo seguinte, verifica-se que está entre o 07 e o 24, então se pega o caminho central para acessar o dado fisicamente.

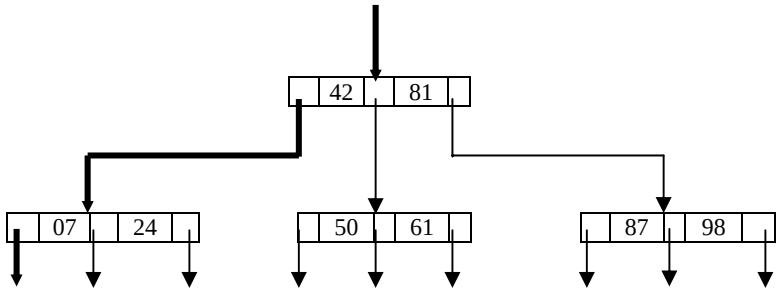


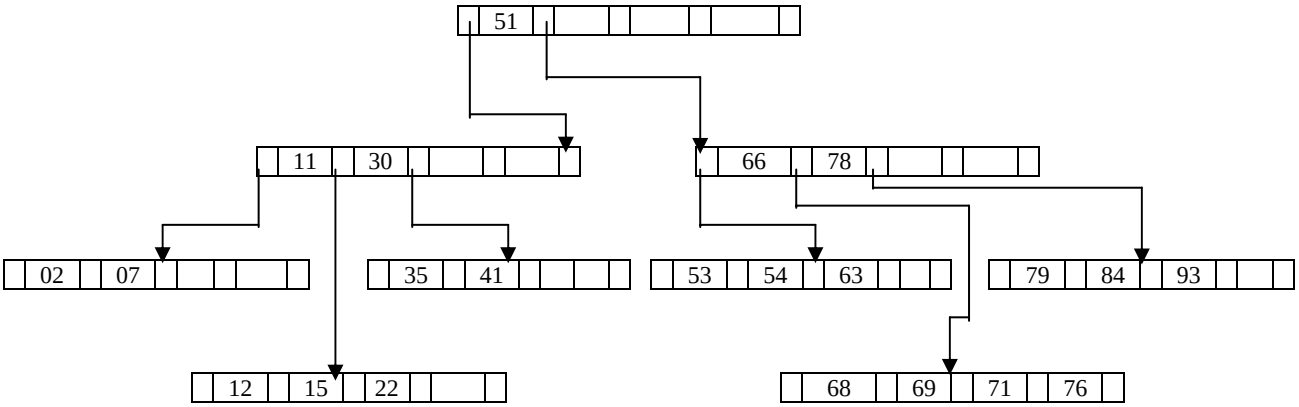
FIGURA 4.3 – B-Tree com 2 chaves e 3 ponteiros – Caminho para chave ‘15’

4.1.1.2 Inserção de uma chave de busca

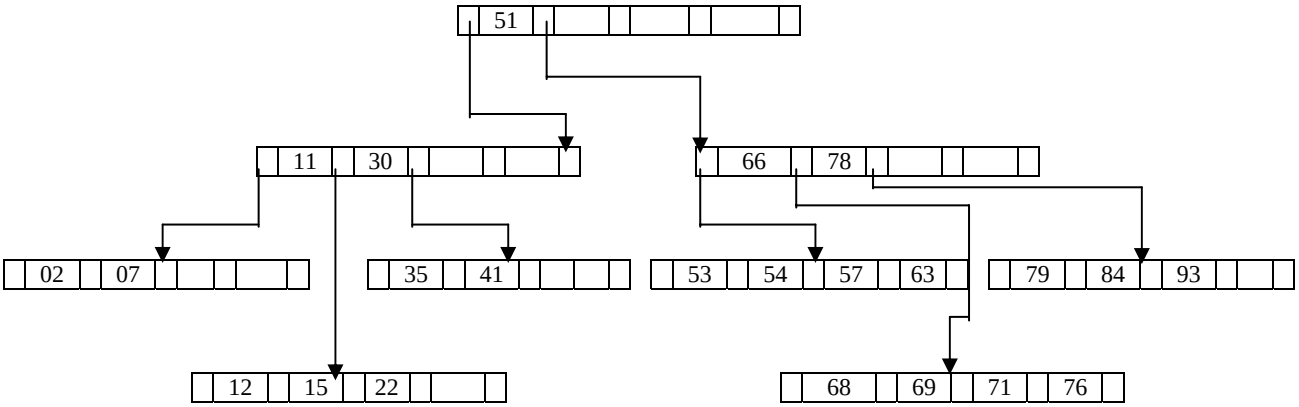
Seguindo a figura 4.4, uma vez que cada nodo em uma B-Tree de ordem **d** contém entre **d** e 2**d** chaves de buscas, excetuando a raiz que pode ter apenas uma chave de busca. Cada nodo na figura 4.4 contém entre 2 e 4 chaves. Em cada nodo, é necessário um indicador que marque o número corrente de chaves armazenadas no nodo, o que está implícito na figura por questões de clareza da mesma. A inserção de uma nova chave requer um processo de dois passos: (i) o primeiro é pesquisar a partir da raiz qual o nodo indicado que conteria a chave a ser inserida; (ii) no segundo passo pode existir duas possibilidades, existir espaço no nodo ou não existir espaço no nodo. No caso da primeira, o valor é simplesmente inserido e a operação termina.

Para a segunda alternativa, o nodo que conteria a chave é dividido (*split*), como mostra a figura 4.5. Das 2**d** + 1 chaves existentes, as **d** menores do nodo vão para um nodo e as **d** maiores vão para o outro nodo, sendo que o valor central (o do meio) é promovido ao nodo pai, e servirá como separador dos outros nodos. Geralmente o nodo pai irá acomodar uma chave adicional e o processo de inserção termina. Caso o nodo pai também necessite ser dividido, é aplicado o mesmo algoritmo, os **d** primeiros vão para um nodo, os **d’s** últimos para o outro nodo, e o valor central é promovido ao nodo pai. Esse processo pode ser replicado até o nodo raiz, acarretando um aumento na altura da árvore.

A figura 4.4a representa uma estrutura B-Tree de ordem 2. A inserção da chave de busca 57 gerou a estrutura da figura 4.4b. Na figura figura 4.5, isso é exemplificado. A inserção do valor 72 ocasionou um *split* no nodo, ficando os **d** primeiros valor no primeiro nodo, 68 e 69; os **d** últimos valores no nodo seguinte, 72 e 76; e o valor central, 71 é promovido ao nodo pai – **d**=2 (ordem da árvore).

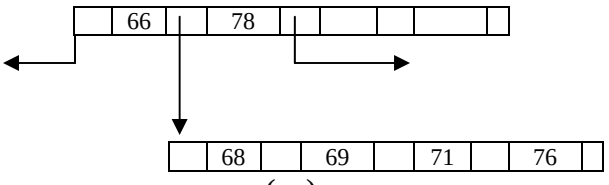


(a)

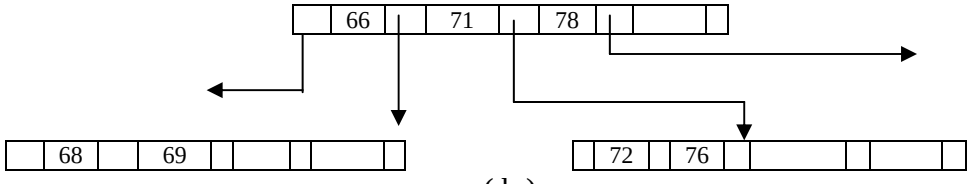


(b)

FIGURA 4.4 – Exemplo de Inserção na B-Tree (valor 57)



(a)



(b)

FIGURA 4.5 – A mesma B-Tree após a inserção da chave 72.

4.1.1.3 Remoção de uma chave de busca

A operação de remoção em uma B-Tree, em analogia à inserção, também requer uma pesquisa do dado a ser removido. Existem duas possibilidades: (i) a chave a ser removida está em um nodo folha, ou (ii) ela não está em um nodo folha. A remoção de um valor em um nodo não folha exige que uma chave adjacente seja encontrada e transferida para o local onde se encontra a chave a ser excluída. Para localizar a chave adjacente na ordem de seqüência das chaves simplesmente faz-se uma procura pela folha mais à esquerda da subárvore da direita a partir do espaço agora disponível.

Uma vez que um espaço vazio tenha sido movido para uma folha, deve-se ter o cuidado de verificar se permanecem no mínimo **d's** chaves de busca em cada nodo. Se a folha sugere menos que **d** chaves de busca, então é necessária uma redistribuição das chaves. Para restaurar o balanceamento, somente uma chave é necessária, o que pode ser obtido por empréstimo de uma folha vizinha. Entretanto, no caso de remoções sucessivas sobre o mesmo nodo, para cada remoção é necessário um empréstimo de chave do nodo vizinho, aumentando o acesso à memória. Uma melhor redistribuição tende a diminuir este problema.

A redistribuição de chaves entre dois nodos vizinhos só é possível dentro da regras de mínimo e máximo número de chaves por nodo, **d** e 2**d**, respectivamente. Se não for possível seguir a regra, então uma união de nodos é a melhor solução. Nesta união as chaves são simplesmente combinadas em um dos nodos e o outro é desconsiderado. Uma vez que somente um nodo restou, a chave do antecessor que separava os dois nodos originais não é mais usada para tal e é também adicionada na folha restante. A figura 4.6 mostra um exemplo de união de nodos e a localização da chave separadora. Este processo de união pode alterar a altura da raiz.

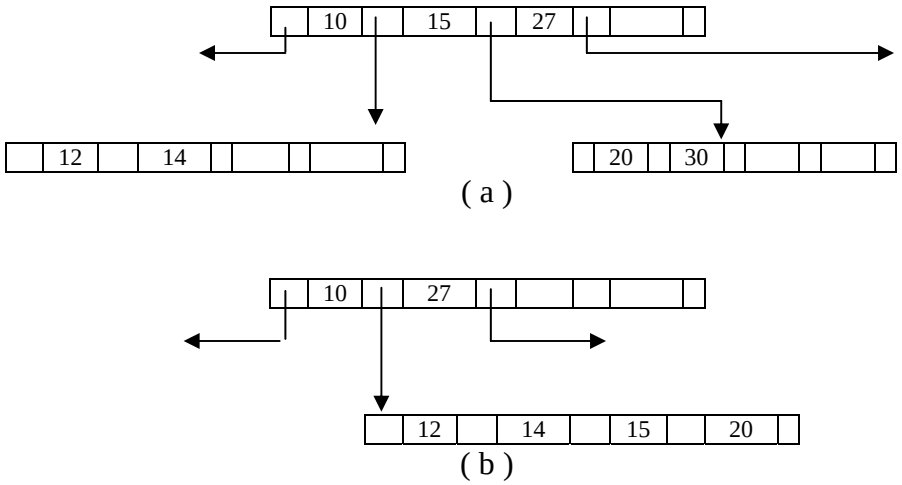


FIGURA 4.6 – Exemplo de Remoção na B-Tree (valor 30)

No exemplo da figura 4.6, o fato que ocasionou a concatenação dos nodos após a remoção do valor 30, foi que no nodo em que se encontrava esse valor, ficaria com

número de chaves menor que o valor mínimo de chaves permitido, como a ordem da árvore é 2 (**d**=2), o nodo não pode conter apenas 1 valor de chave.

4.1.2 Custo de uma B-Tree

Uma vez que acessar um nodo em uma B-Tree requer um acesso ao dispositivo de armazenamento secundário, o número de nodos visitados durante uma operação fornece uma medida de custo desta operação.

O custo de processamento de uma operação de busca cresce com o logaritmo do tamanho da árvore ($Log_{\mathbf{d}}$ **N**) [HOR84]. A seguir pode-se ter uma idéia da relação custo/benefício entre o tamanho do arquivo a ser indexado e a ordem da árvore que deve ser estipulada.

TABELA 4.1 – Número de Registro x Número de Chaves em uma B-Tree

Ordem B-Tree \ Tamanho Registro	10e +3	10e +4	10e +5	10e +6	10e +7
10	3	4	5	6	7
50	2	6	6	4	4
100	2	2	3	3	4
150	2	2	3	3	4

Na tabela 4.1 tem-se nas colunas o tamanho dos registros (10e + 3, 10e + 4, 10e + 5, 10e + 6 e 10e + 7), nas linhas a ordem da B-Tree (10, 50, 100 e 150), e o cruzamento destas informações resulta no número de acessos a disco no pior dos casos. Por exemplo, dados de uma B-Tree de ordem 50 que indexa um arquivo de 1 milhão de registros, no pior caso, podem ser recuperados com somente quatro acessos a disco.

4.2 B^+-Tree – uma evolução da B-Tree

A B^+-Tree [KNU73] é uma variação da B-Tree, uma evolução dos algoritmos e da estrutura dos nodos. Uma das principais características da B^+-Tree, assim como a B-Tree, é a manutenção do balanceamento de sua estrutura de forma natural. Em virtude dos seus algoritmos, a B^+-Tree mantém estrutura naturalmente balanceada, o que garante o desempenho eficiente da técnica. Como conseqüência direta desta característica, a B^+-Tree impõe algumas sobrecargas na inserção e remoção, bem como algum espaço adicional para fazer o processo de balanceamento da árvore. Todavia, estes pontos são totalmente aceitáveis para um arquivo com alta freqüência de modificação, uma vez que o custo da reorganização do arquivo é evitado, compensando qualquer outra falha. É sem dúvida a mais usada entre as diversas estrutura de arquivos que mantém sua eficiência

independente da inserção e/ou remoção de dados.

A principal diferença entre a B-Tree e a B^+-Tree é a composição da árvore. Enquanto na B-Tree os valores não se encontram necessariamente nos nodos folhas, características que tem melhor desempenho nas consultas, a B^+-Tree tem obrigatoriamente os seus valores posicionados nos nodos folhas, melhorando o desempenho para as inserções e alterações. Um exemplo de B^+-Tree pode ser encontrado na figura 4.7.

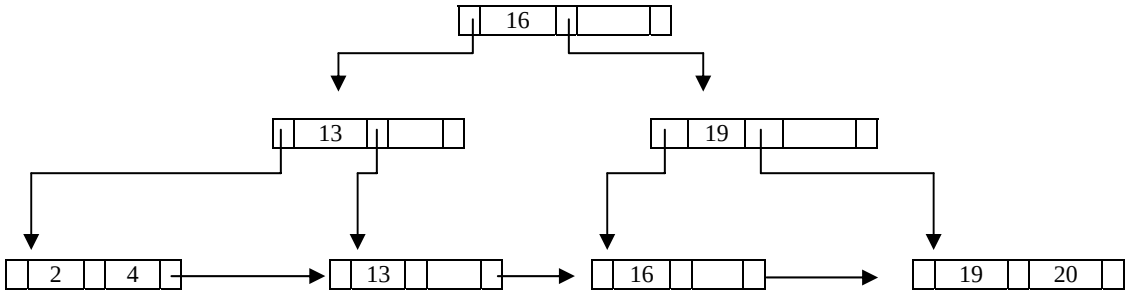


FIGURA 4.7 – Exemplo de B^+-Tree

Na figura 4.7 os valores de chave de busca 13, 16, 19 são encontrados nos nodos não folhas e obrigatoriamente nos nodos folhas, isso devido à estrtura da ávore B^+-Tree. Os algoritmos básicos desta técnica são os mesmos da B-Tree.

4.3 Comparação entre B-Tree e B^+-Tree

As B-Tree oferecem uma vantagem adicional sobre as B^+-Tree, além da ausência de armazenamento redundante de chaves de busca. Em uma busca na B^+-Tree é sempre necessário atravessar um caminho de uma raiz da árvore até algum nodo folha. No entanto, em uma B-Tree, às vezes, é possível encontrar o valor desejado antes de ler um nodo folha. Assim, uma busca é um pouco mais rápida em uma B-Tree, embora, em geral, o tempo de busca seja proporcional ao logaritmo do número de chaves de busca.

Essas vantagens da B-Tree sobre a B^+-Tree são compensadas por algumas desvantagens [SZW94]. Por exemplo, a remoção em uma B-Tree é mais complexa. Em uma B^+-Tree, a entrada a ser removida sempre está em uma folha. Em uma B-Tree, a entrada a ser removida pode estar em um nodo não folha. O valor adequado precisa ser selecionado como uma substituição da subárvore do nodo contendo a entrada removida. Especificamente, se uma chave de busca **Ki** é removida, a menor chave de busca aparece na subárvore do ponteiro **Pi** + 1 precisa ser movida para o campo anteriormente ocupado por **Ki**.

As vantagens da B-Tree são marginais para índices grandes. Assim sendo, a simplicidade estrutural de uma B^+-Tree é preferida por muitos implementadores de sistemas de bancos de dados.

4.4 R-Tree

Os índices R-Trees [GUT84] são utilizados em dados com características multidimensionais. No domínio geográfico, objetos como pontos, linhas e polígonos podem ser usados para representar as localizações de estradas, código de regiões ou outras entidades com propriedades bidimensionais (2D) ou tridimensionais (3D).

A técnica R-Tree é uma variação do B^+-Tree para aplicação em dados que possuem duas ou mais dimensões e que ao longo do tempo podem se sobrepor uns aos outros. Esses tipos de dados são em geral complexos, com muitos detalhes. Com isto, sua indexação também se torna complexa.

A técnica R-Tree foi desenvolvida primeiramente para criar uma hierarquia de objetos os quais não se encontram em uma disposição lógica espacial, conforme exemplo da figura 4.7. Pode ser construída basicamente de duas formas: (i) estática e (ii) dinâmica. No primeiro caso, só é montada a árvore após a leitura de todos os objetos em questão, ou seja, espera-se até que todos os objetos tenham sido computados para então construir a árvore de indexação. Na segunda forma, a árvore é montada à medida em que os objetos estão sendo lidos. Os resultados dos métodos estáticos são geralmente caracterizados como sendo um "pacote", contendo todos os dados de cada nodo da R-Tree, uma vez que a árvore só é processada após a leitura dos objetos.

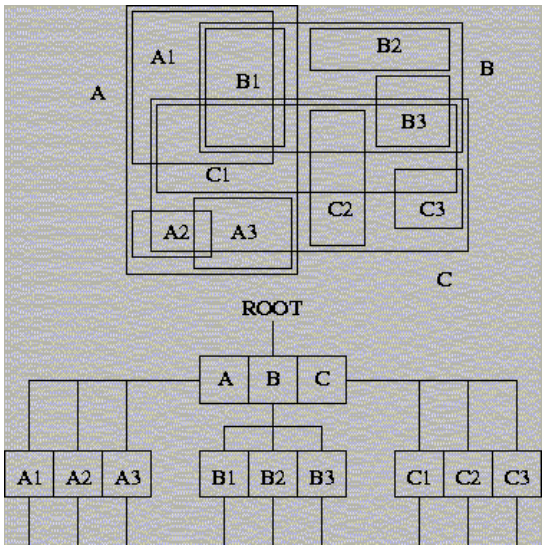


FIGURA 4.8 - Exemplo de Estrutura de Indexação R-Tree

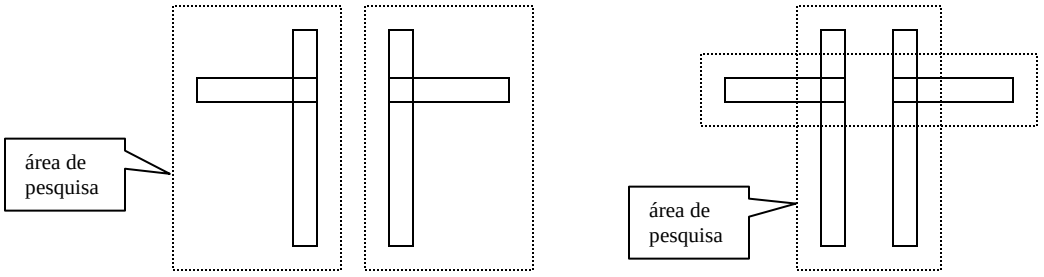
Os nodos da R-Tree são compostos pelos objetos, já hierarquicamente dispostos, que fazem parte da região espacial a ser indexada. Existem dois métodos de como montar os nodos R-Tree. O método mais natural é ocupar o espaço na hora da decisão de quais objetos serão agregados. Uma alternativa para decidir quais objetos serão agregados primeiro, seria ordenar por desempenho. Outra alternativa, seria preservar a ordem dos objetos a medida em que são lidos. Esta solução quase não é usada.

O método R-Tree é basicamente uma estrutura construída a partir de um número de objetos (retângulos) multidimensionais, e é balanceado dinamicamente com todos os objetos carregados no mesmo nível. Cada objeto de um nível maior é formado pelo retângulo que agrega todos os menores, ou seja do nível menor, fazendo assim, uma hierarquia de objetos (figura 4.8).

O grande intuito da R-Tree é obter uma forma computacional de ordenar e organizar objetos espaciais em uma estrutura de dados capaz de ser processada no primeiro instante, para achar a localização de qualquer ponto pertencente aos retângulos. A R-Tree deve manter essa estrutura organizada (balanceada) sempre que modificados os seus dados.

Portanto, o primeiro passo da técnica é dividir em retângulos maiores os objetos que contém as regiões menores completamente inclusas nos retângulos maiores. Por exemplo, a figura 4.8 foi dividida nas áreas maiores A, B e C que emglobam as áreas menores, A1,A2 e A3; B1, B2 e B3; C1, C2 e C3. Esta divisão é conhecida como *split.*

O conceito de *split* nas técnicas de indexação para dados espaciais tem um enfoque um pouco diferenciado do conceito da B-Tree e suas variantes. Enquanto nas técnicas para dados lineares o *split* é um processo em que o nodo tem seu número mínimo **(m)** ou número máximo **(M)** de entradas por nodo afetado, acarretando uma reorganização dos nodos, na R-Tree e variantes o *split* é o processo de divisão das áreas a serem indexadas em retângulos envolventes. A semelhança destes processos, e por isso o mesmo nome, é que assim como na B-Tree, a R-Tree também possui os parâmetros mínimo **(m)** ou número máximo **(M)** de entradas por nodo e a determinação das áres envolventes (método *split*) tem como regra manter íntegros esses dois parâmetros (**m** e **M**), conforme figura 4.9. Nesta, o *split* 2 tem melhor desempenho do que o *split* 1, por reduzir a área de pesquisa. Esse método foi realizado seguindo o parâmetro **M** = 2, ou seja, o número máximo de entradas por nodo é 2, por isso foi dividido em duas partes.



Split 1 *Split 2*

FIGURA 4.9 – Exemplo de *Split* [GUT84]

Esse método pode apresentar mais de um nível, dependendo da complexidade das regiões a serem indexadas. Na figura 4.10a é apresentado o estado inicial e o resultado final dessa separação em regiões. Na figura 4.10b é mostrado um nível a mais de detalhamento desse passo.

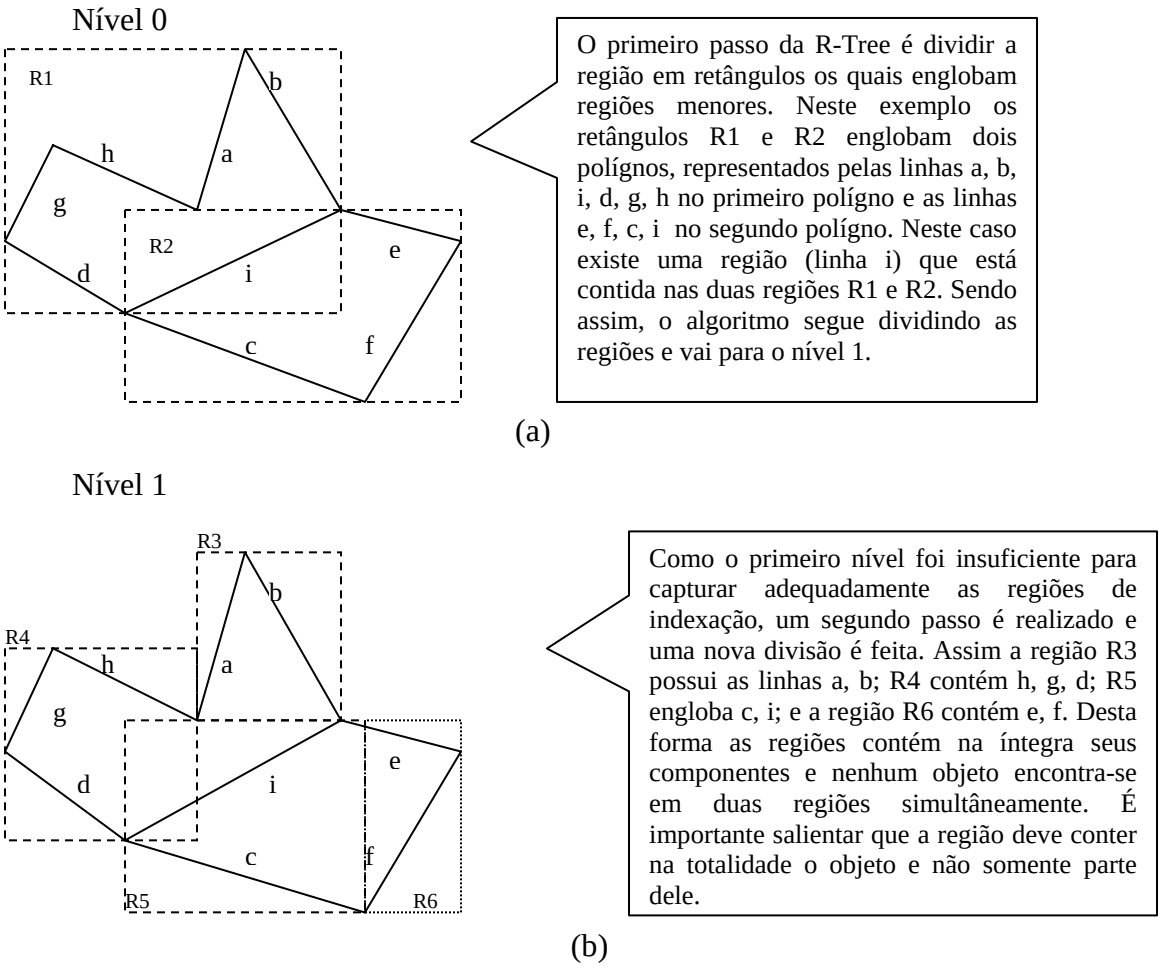


FIGURA 4.10 – Captura de Regiões na R-Tree

Após esta divisão, o segundo passo é montar uma estrutura com os objetos hierarquicamente dispostos, podendo desta forma já ser visualizada uma estrutura computacional capaz de ser processada, conforme figura 4.11.

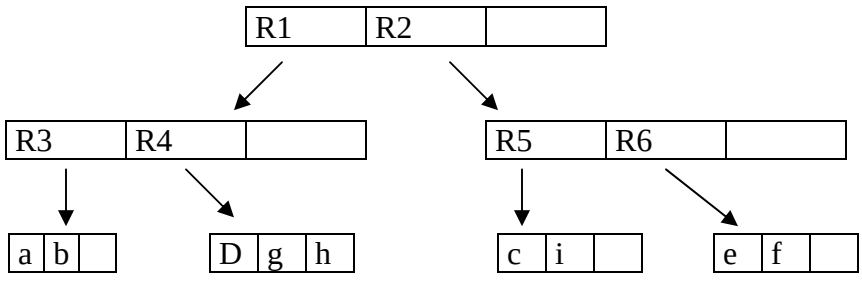


FIGURA 4.11 – Estruturação das regiões a serem indexadas

Uma terceira fase é organizar esses objetos hierarquicamente em nodos correspondendo às suas coordenadas. Fica assim caracterizada uma árvore capaz de indexar, segundo seus nodos, as figuras incialmente sem uma disposição lógica espacial. A figura 4.12 exemplifica o terceiro passo.

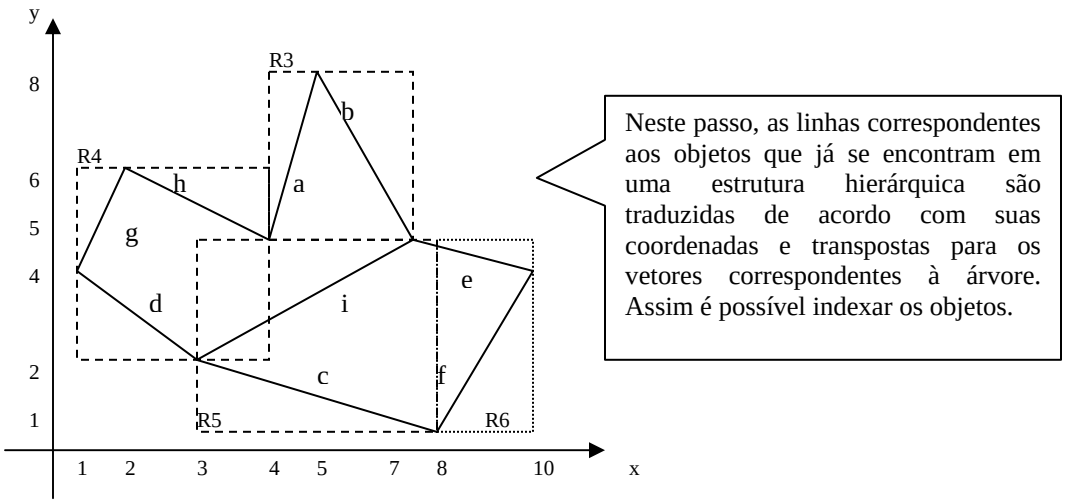


FIGURA 4.12 – Transposição dos objetos para coordenadas cartesianas

Esse passo resulta na árvore R-Tree representada na figura 4.13.

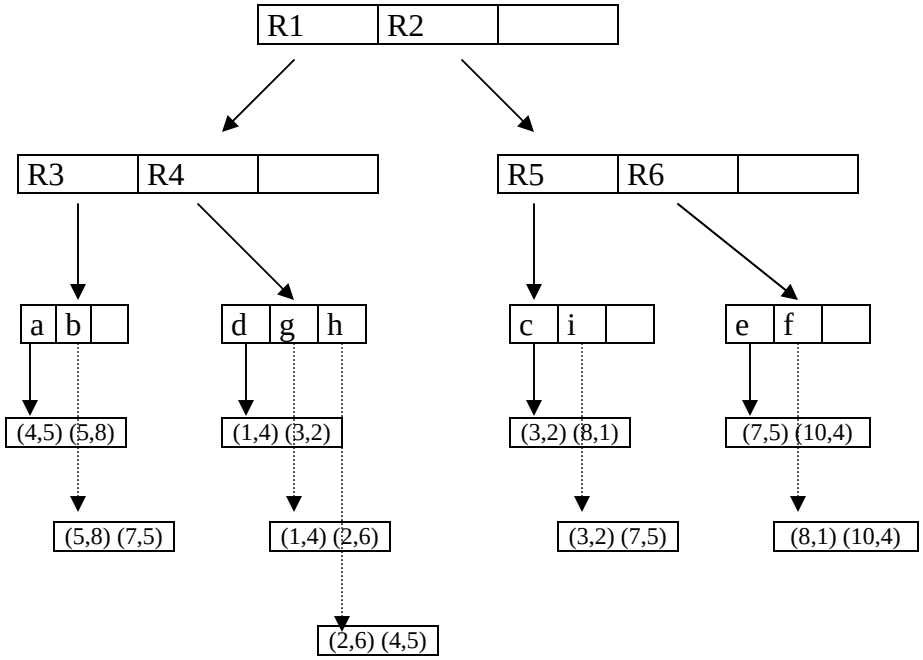


FIGURA 4.13 – Estruturação das regiões em vetores correspondente a uma R-Tree

Maiores detalhamentos dos algoritmos da R-Tree, tais como *search subtrees*,

search leaf node, insert, choose leaf, split node, adjust tree, delete, find leaf, condense tree, quadratic-cost, pick seeds, pick nex e *linear cost*, são encontrados em [GUT84].

A teoria principal da R-Tree para melhorar o desempenho é diminuir o tamanho dos espaços das regiões envolventes e diminuir os espaços em branco entre eles. Conseqüentemente, a probabilidade de fazer uma pesquisa em regiões "desocupadas" se torna menor, aumentando o desempenho do índice (figura 4.9). É notório que a técnica R-Tree possui um conjunto de conceitos e algoritmos complexos. Sendo assim, qualquer alteração para a melhoria desses algoritmos gera uma especialização do índice R-Tree. Existem muitas variantes desta técnica. Elas diferem basicamente nos algoritmos de construção dinâmica ou estática da árvore e na maneira de como fazer o *split* dos nodos.

O método que evoluiu da R-Tree mais utilizado é sem dúvidas o R*-Tree ou *RStar-Tree* [BEC90]. Ele se diferencia da R-Tree no processo de *split*, diminuindo as regiões de pesquisa e solucionando melhor as situações de sobreposições de regiões. Entretanto, traz problemas para a decomposição e por conseqüência a transposição para os vetores correspondentes à árvore. Esse método também é utilizado como base nas técnicas de indexação para dados temporais, por ter essa característica de melhor desempenho para regiões de menor probabilidade de pesquisa. Analogamente, são as regiões em que os dados históricos são menos requisitados do que nas regiões correspondentes a versões correntes dos BDTs.

4.5 Proposta de um modelo UML para B-Tree e B$^+$-Tree

Com base na análise das técnicas de indexação B-Tree e B$^+$-Tree, apresentamos aqui a modelagem destas técnicas através de um diagrama de classes UML (figura 4.14). Posteriormente será dada uma explicação de cada classe, atributo e/ou método pertinente.

O objetivo do modelo é servir de base para as técnicas de indexação que utilizam os conceitos da B-Tree e da B$^+$-Tree, tanto total como parcialmente. Geralmente as técnicas de indexação para dados temporais recaem sobre alguma técnica-base para indexar parte dos dados temporais. Assim, é necessário que estas técnicas sejam disponibilizadas previamente.

Mesmo que o principal objetivo do modelo seja servir de base para outras técnicas, não existe nenhum impedimento de que possa ser utilizada parte da técnica ou somente alguma estrutura semântica do modelo, como por exemplo, a estrutura de nodo, representada em **ComposiçãoNodo**, ou alguma das características das árvores, representadas na classe **Tree**. Sendo assim, esta primeira modelagem assume um papel fundamental para que as demais tenham uma base conceitual bem definida.

A utilização do modelo segue como qualquer outro modelo disponibilizado previamente. Com o padrão UML esta característica fica mais acentuada, por ser de fácil reutilização. A modificação de conceitos, tanto por inclusão como por alteração, fica a cargo do projetista da nova técnica de indexação apenas especializando novas classes e usufruindo as características conceituais do modelo. Por exemplo, para utilizar a estrutura em que são modeladas características das árvores, tais como ordem, nível, número

máximo de chave de busca, número mínimo de chave de busca (determinantes para o *split*), entre outras características. Pode-se especializar uma classe da classe **Tree**, e na nova classe apenas acrescentar as características particulares a ela. Esse exemplo toma maior proporção uma vez que esta classe **Tree** possui um método denominado **InformaçãoDesempenho**. Esse método pode prover informações de acesso segundo características básicas das árvores e do seu tipo. Para usufruir desse método, a fórmula que resulta no número de acessos deve ser implementada para cada tipo de árvore. Por exemplo, na B-Tree e na B^+-Tree o número de acessos é Log_d N [HOR84], conforme exemplo na tabela 4.1. Para árvores como R-Tree [GUT84], R*-Tree [BEC90], GR-Tree [BLI98], entre outras, deve ser disponibilizada a forma de cálculo do número de acessos.

Na modelagem apresentada na figura 4.14 as características comuns aos dois tipos de nodo da B-Tree e da B^+-Tree foram generalizadas na classe **ComposiçãoNodo**; e os atributos específicos de cada uma foram especializados nas classes **ComposiçãoNodoB** e **ComposiçãoNodoB+**. Assim, as características da composição dos nodos são passadas por herança para as classes mais específicas completando a sua semântica.

Para formarem os nodos, foram utilizados vetores do tipo de composição do nodo específico a cada técnica, representados nas classes **NodoB** e **NodoB+**. Assim a semântica referente à figura 4.1 (Organização de um nodo em uma B-Tree) foi contemplada e serve de base para a formação da árvore propriamente dita.

Para formar a estrutura de árvores foram utilizados vetores compostos pelos vetores dos nodos. A garantia de consistência conceitual dos vetores é garantida pelos relacionamentos do tipo associação.

A classe **Tree** tem o objetivo de prover as características comuns da B-Tree e da B^+-Tree no que tange à estrutura de árvore propriamente dita, como inicializações e informações de desempenho.

As classes **B-Tree** e **B+-Tree** materializam os índices de cada tipo de técnica. Herdam as características da classe **Tree** e possuem os relacionamentos pertinentes a cada tipo de nodo. Os métodos que operam cada tipo de árvore manipulam estruturas distintas, por isso não estão localizados na classe **Tree**. Haveria uma redefinição quase que integral dos métodos.

A cardinalidade das associações é basicamente definida como “um para muitos”, graficamente representada por (1..*). Por exemplo, no relacionamento entre composições e nodos, um nodo pode conter um ou mais composições e, por sua vez, uma composição pode estar contida em um ou mais nodos.

Um importante método é proposto nesta modelagem: **InformaçãoDesempenho**. Esse método, aparece como inovação, pois são poucas as técnicas que promovem a informação de desempenho na sua estrutura. São feitos estudos para se descobrir fórmulas que expressem esse desempenho, mas as simulações ficam a cargo dos operadores. Esse método traduz o interesse de projetar o número de acessos a disco dependendo de uma fórmula que acompanha a técnica utilizada. No caso das citadas, a fórmula é entendida com $Log_{\mathbf{d}}$ **N** [HOR84], onde **N** é o número de registro da simulação e **d** é o tamanho máximo de chaves de busca por nodo. Por exemplo, uma simulação de

1000 registros e máximo de chaves 10 por nodo, teria no pior caso três acessos a disco, $\text{Log}_{10} 1000 = 3$.

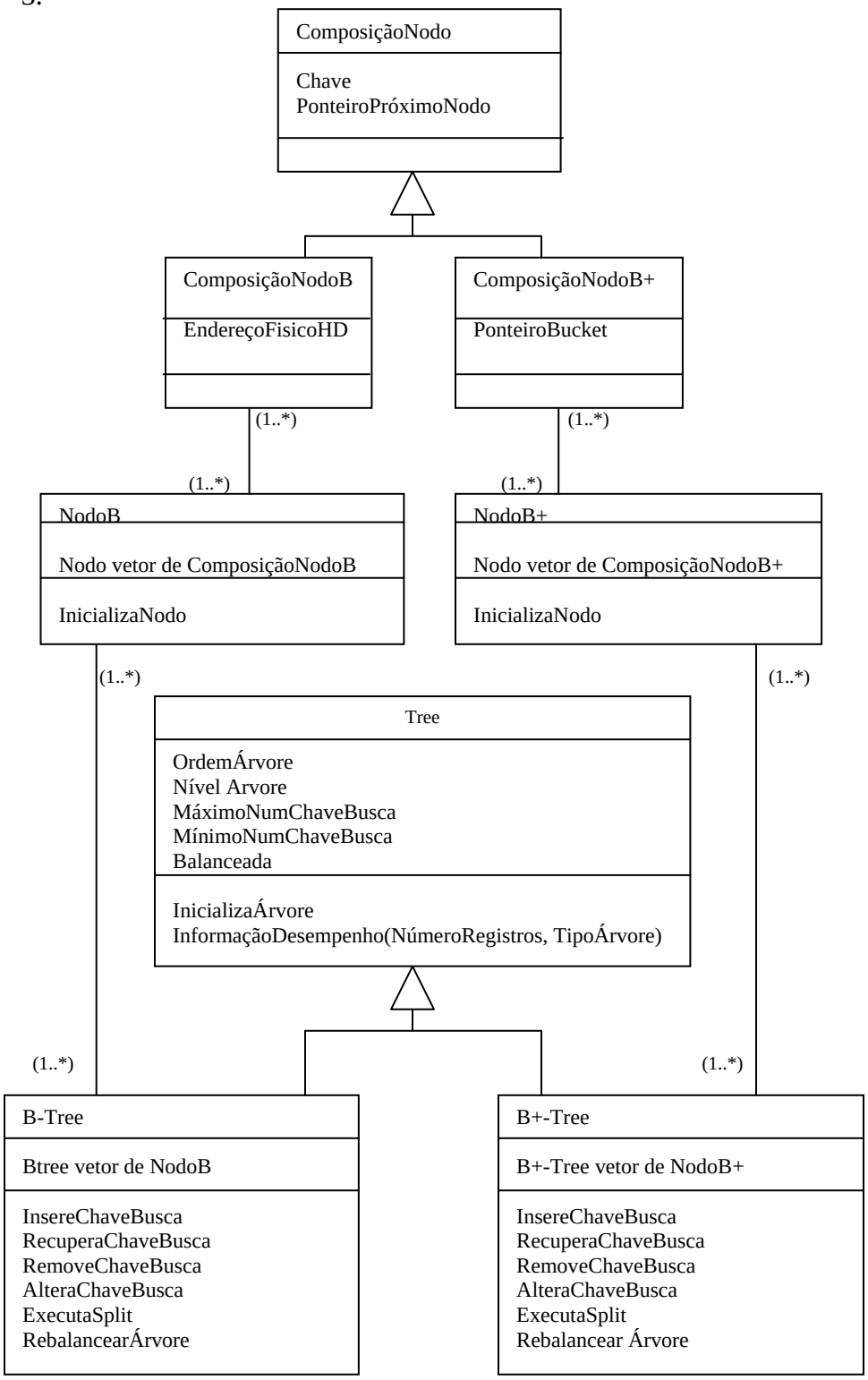


FIGURA 4.14 – Modelagem das Técnicas B-Tree e B^+-Tree

4.6 Considerações finais

O grande intuito desse capítulo foi explanar os conceitos que governam as principais técnicas de indexação que servirão de base para as técnicas específicas para dados temporais. Conceitos como nodo, número de entradas por nodo, ordem, *split*, entre outros. Foram abordados porque serão citados no restante do trabalho. Uma outra característica importante pode ser ressaltada: a B-Tree possui melhor desempenho em sistemas que dão ênfase às consultas e a B^+-Tree tem melhor eficiência para operações de inserção e alteração.

Outra técnica apresentada nesta seção foi a R-Tree. É relevante este estudo por conter conceitos diferenciados dos presentes na B-Tree e B^+-Tree e por apresentar uma característica similar aos dados temporais [LÜ95].

Outro fator de grande importância foi a apresentação do modelo da técnica B-Tree e B^+-Tree já na notação UML, configurando um primeiro passo para o modelo final (objetivo deste trabalho).

5 Técnicas específicas para dados temporais

Este capítulo apresenta as técnicas de indexação para dados temporais. Embora tenha sido proposto um número considerável de técnicas que visam contemplar a complexidade dos dados temporais, ainda existem algumas falhas nestas estruturas. Segundo [LÜ95], o principal problema dos índices existentes é que eles indexam "pontos no tempo", ao passo que deveriam indexar "intervalos de tempo". É um problema conceitual e de difícil resolução, porque além da semântica embutida em indexar intervalos de tempo e a manutenção da consistência desses intervalos, as operações realizadas deveriam ter um tempo de resposta satisfatório.

Existem estudos que comprovam que índices regulares, tal como B^+-Tree, são ineficientes para dados temporais [SAL97], porém servem de base para os índices específicos de dados temporais. A maioria das propostas aponta índices para TT, poucos atendem TV e quase nenhum contemplam uma indexação de TV e TT de maneira eficiente.

Muitas técnicas de indexação que têm sido propostas para acesso a dados temporais trabalham com um conjunto de "n" listas, cada uma contendo informações relevantes à história de um elemento. Em geral são usadas listas encadeadas, na qual a versão corrente de cada tupla tem um *timestamp* e um ponteiro para a lista encadeada das versões passadas das tuplas, como por exemplo o *Time Index* [ELM90].

Alguns problemas são identificados nestes métodos de indexação:

- a maioria suporta somente uma dimensão de tempo;
- a maioria não distingue ponto no tempo e intervalo no tempo; e
- faltam informações de desempenho.

Os objetivos de índices para dados temporais são:

- promover o acesso a dados modificados e logicamente removidos;
- promover pesquisas temporais pelo seu estado e pelo seu histórico, em particular com o suporte a dados temporais multidimensionais; e
- técnicas de indexação devem promover informações sobre o desempenho.

Cada técnica apresentada nas próximas seções possui alguma característica particular, e é com base nestas características que um *framework* conceitual será proposto no próximo capítulo.

Juntamente com a apresentação das características de cada técnica de indexação, é sugerida uma modelagem UML. Essa modelagem já visa uma padronização para o modelo unificado (*framework* conceitual).

É consenso dos pesquisadores que dados espaciais e dados temporais possuem muitas similaridades. Por isso, são analisados também índices baseados nas técnicas apresentadas para dados espaciais. Muitas das técnicas a seguir têm como ponto de

partida as técnicas R-Tree e R*-Tree. Outros fazem um mapeamento dos dados temporais para uma forma linear, e por conseqüência, podem adaptar técnicas convencionais como B-Tree e principalmente B^+-Tree.

5.1 Time Index

O *Time Index* [ELM90] propõe um acesso eficiente para dados temporais em intervalos de **tempo de validade**.

A idéia de Elmasri é manter um conjunto de pontos indexados linearmente, os quais podem variar em um intervalo de [0, *now*], onde *now* [CLI97] é o instante de tempo atual (tabela 5.1). Este conjunto é formado pela variável *now* e todos os pontos de tempos correspondentes a: (i) a *TVbegin* (início tempo de validade) e (ii) o ponto de tempo imediatamente depois do *TVend* (final do tempo de validade). O principal requisito do *Time Index* é que estes pontos estejam linearmente ordenados, podendo ser utilizada uma técnica tradicional de indexação, por exemplo a B^+-Tree.

Cada nodo folha da B^+-Tree contém um conjunto de entradas na forma *(Ti, Bi)*, onde *Bi* é um ponteiro para um *bucket* contendo ponteiros para as tuplas para as quais o tempo de validade *Ti* é verdadeiro, ou seja, cada tempo de validade significativo possui um ponteiro para um *bucket* o qual contém o conjunto de ponteiros para as tuplas relativas ao tempo de validade *Ti.*

TABELA 5.1 – Tabela de Entrada de Pessoas nos USA [ELM90]

Tupla	Id	Ponto de Entrada	Intervalo de Permanência
T1	P1	New York	[0, 3]
T2	P1	Los Angeles	[4, *now*]
T4	P2	San Francisco	[0, 5]
T7	P3	Los Angeles	[0, 7]
T8	P3	San Francisco	[8, 9]
T10	P4	New York	[2, 3]
T11	P4	Los Angeles	[8, *now*]
T12	P5	Los Angeles	[10, *now*]
T13	P6	New York	[12, *now*]
T14	P7	New York	[11, *now*]

Elmasri propõe que o *Time Index* possa ser agregado a índices regulares, para facilitar o processamento de consultas históricas envolvendo condições com atributos não-temporais. Por exemplo: “recuperar todas as pessoas que entraram nos USA por Los Angeles e permaneceram do dia 4 até o dia 6”.

No exemplo acima existem atributos não-temporais, envolvendo técnicas regulares de indexação – ponto de entrada Los Angeles; portanto, a melhor utilização do *Time Index* é agregar esta estrutura a cada nodo folha de um índice regular, como a B^+-Tree, conforme figura 5.1.

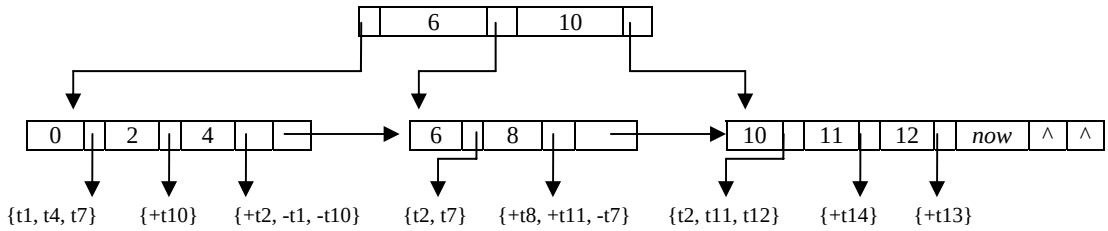


FIGURA 5.1 – *Time Index* a partir do *Snapshot* mais recente da Tabela 5.1 [ELM90]

Isto representa um grande aperfeiçoamento sobre uma busca seqüencial de BD. Entretanto, esta indexação suporta somente uma dimensão, a de tempo de validade, não sendo definido como o *Time Index* poderia contemplar a dimensão TT. Seria necessária uma extensão para o suporte ao tempo de transação, necessário em consultas temporais.

5.1.1 Proposta de um modelo UML para *Time Index*

Nesta seção é apresenta a modelagem UML da técnica *Time Index*. Este modelo será parte do *framework* conceitual apresentado posteriormente, com intuito de aproveitar suas características através de especializações, generalizações, associações e outros elementos da UML.

O modelo é basicamente composto por uma primeira árvore, indexando os tempos de validades, representada na classe **TTVree**; e uma segunda árvore encadeada dessa primeira, indexando a história da informação, bem como os índices de dados não temporais. A segunda árvore está representada na classe **TimeIndexTree**.

A técnica *Time Index* idealizada por Elmasri contempla somente tempos de validade. Para cada tempo de validade indexado, são apontadas as tuplas válidas dentro deste tempo. Para isso o primeiro passo é ordenar linearmente os tempos de validades através de uma das técnicas-bases apresentadas anteriormente, por exemplo a B^+-Tree, para só então se obter a chave de busca das tuplas relativa à história. Essa busca pode ser feita sobre outra árvore B^+-Tree, como apresentado no modelo da figura 5.2.

Na segunda operação sobre a árvore B^+-Tree pode-se agregar índices regulares, ou seja, dados que não sejam temporais, sendo essa busca feita somente para as tuplas de validade desejada, pois já foram selecionadas no índice acima.

Esta técnica é mais eficiente quando opera pela técnica B^+-Tree, por razões de implementação do código [ELM90]. Porém nada impede que conceitualmente possa ser utilizada outra técnica de índices lineares para indexar tantos os tempos de validade como os dados não temporais, como exemplificada no modelo pela colaboração de B^+-Tree **ou** por B-Tree. Por exemplo, sabe-se que a B-Tree é mais eficiente que a B^+-Tree no que diz respeito a consultas, porém tem um pior desempenho para a inserção e a atualização de tuplas. Se o sistema proposto opera muito mais em modo de consulta, seria interessante construir uma variação da técnica *Time Index* com base na estrutura B-Tree ao invés da B^+-Tree. Essa característica de variabilidade da estrutura consegue-se pelo fato de ser a modelagem totalmente hierárquica.

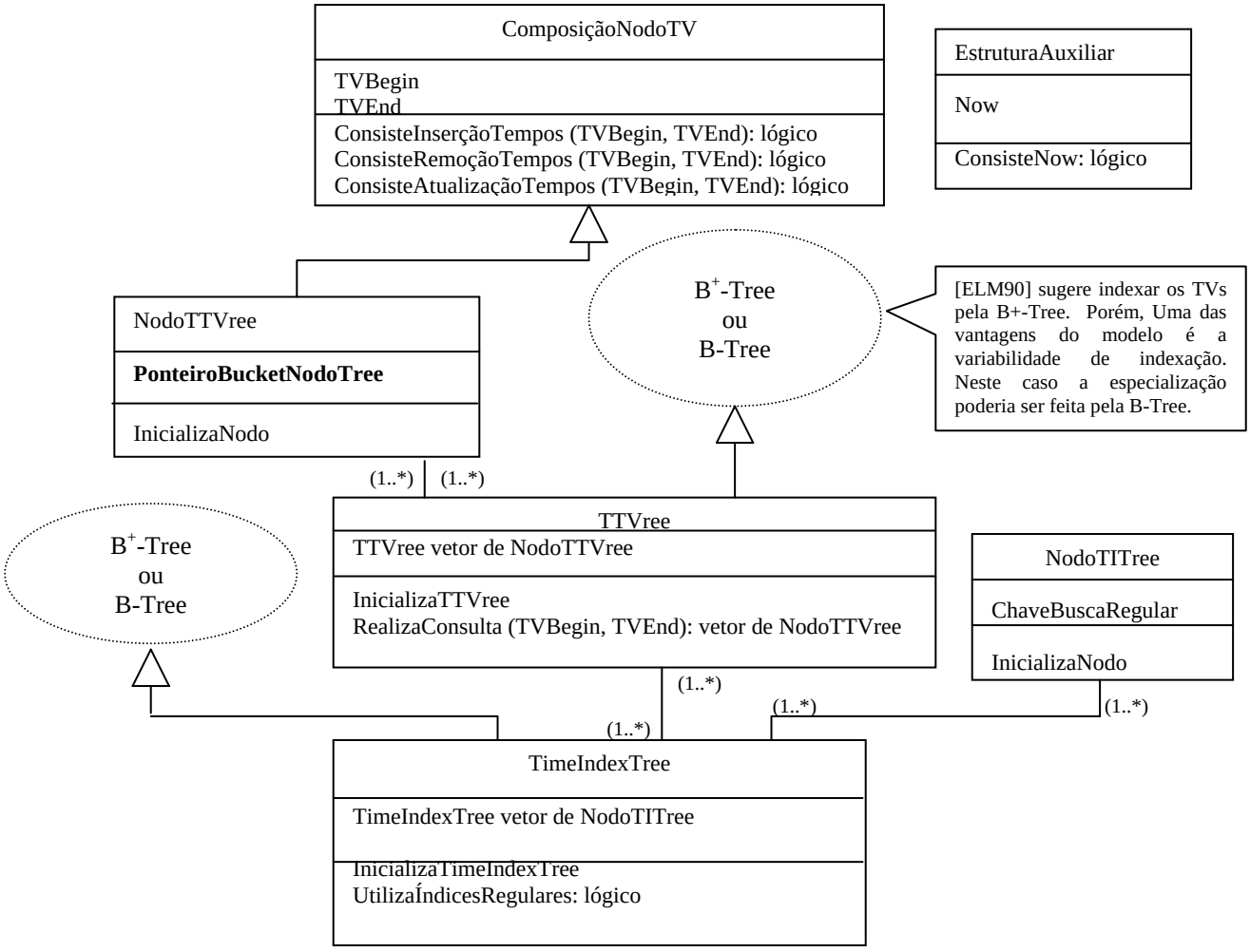


FIGURA 5.2 – Proposta do Modelo para *Time Index*

No modelo proposto para esta técnica, representado na figura 5.2, a classe **ComposiçãoNodoTV** representa os tempos de validade. Serve tão somente para fazer parte, através de herança, da composição do nodo de uma estrutura de árvore de tempos de validade, a qual será indexada pela técnica da B^+-Tree. Os métodos mais relevantes dessa classe são os métodos pertinentes à consistência das operações sobre os tempos:

- **ConsisteInserçãoTempo:** valida se os tempos, inicial e final, são válidos tanto sintaticamente como semanticamente, ou seja, se o tempo final é maior ou igual ao tempo inicial ou se o tempo final é nulo (ausência de valor). Significa que a tupla é válida, ou ainda que o tempo final é inicializado com uma estrutura auxiliar reconhecida por todo modelo, denominada *now*;
- **ConsisteRemoçãoTempo:** consiste e atualiza a finalização da tupla pelo tempo final de validade, ou seja, é atualizado o valor de tempo de validade; e

- **ConsisteAtualização:** esse método, além de finalizar a tupla utilizando o método **ConsisteRemoçãoTempo**, insere outra tupla através do método **ConsisteInserçãoTempo**, pois em BDTs a atualização de uma tupla significa finalizar o tempo de validade desta e inserir uma "cópia" da mesma com um outro valor de tempo de validade.

A classe **NodoTTVree** representa a estrutura do nodo. É composto por *(Bi, Ti)*, onde *Bi* é o ponteiro para o *bucket* das tuplas válidas no intervalo representado por *Ti*. Logo, *Ti* é representado pela herança da classe **ComposiçãoNodoTV**. Apresenta o seguinte método:

- **InicializaNodo:** opera a inicialização de uma entrada contendo validações dos tempos de validade e dos *buckets* existentes.

Classe **EstruturaAuxiliar** representa a "variável" *now*, a qual sustenta a idéia de um marcador temporal para o instante atual. Possui apenas um método:

- **ConsisteNow:** método pertinente consiste o valor *now* em termos sintáticos e não semânticos.

A classe cerne da estrutura é representada pela **TTVree**. É esta que contém os tempos de validade linearmente ordenados pela B^+-Tree. É composta por um vetor de nodos da estrutura *(Bi, Ti)*, herdada da classe **NodoTTVree**, ou seja, é uma estrutura formada por *buckets* e seus respectivos tempos de validades. Possui os seguintes métodos:

- **RealizaConsulta**: que deve ser sobrescrito ao método da B^+-Tree porque possui uma característica peculiar, devolve um vetor de tuplas válidas para o intervalo de tempos de validade indicado na consulta; e
- **InicializaTTVree:** esse métdo serve para inicializar a árvore de indexação.

A estrutura para indexar os valores regulares, que não são valores temporais, é chamada de **NodoTITree**. É uma classe e que serve para doar as chaves para a **TimeIndexTree** processar os *buckets* indicados pela **TTVree**, verificando se existem índices regulares. É importante salientar que esses índices só serão operados para as tuplas válidas quando no intervalo dos seus tempo de validade. Esses dados são indexados pela técnica B^+-Tree.

A classe **TimeIndexTree** apresenta o conjunto dos *buckets* de todas as tuplas, inclusive a de índices regulares. A classe **TTVree**, a qual representa o índice dos tempos de validade, aponta, para cada tempo desses, uma entrada em **TimeIndexTree**, que, depois de localizada, verifica a utilização ou não de índices regulares. Possui os seguintes métodos:

- **InicializaTimeIndexTree:** método para inicializa a estrutura de árvore. Inicaliza as variáveis e vetores para constituir a estrutura; e

- **UtilizaÍndicesRegulares:** função que retorna se a estrutura utiliza ou não índices regulares. Tem o intuito de direcionar a ativação dos métodos da classe **NodoTITree** para fazer uso de uma indexação para dados não

temporais.

5.1.2 Características principais do *Time Index*

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa somente Tempo de Validade;
- utiliza a estrutura auxiliar *no*w;
- cada tempo indexado por uma B^+-Tree indica outro índice de todas tuplas válidas para aquele tempo;
- utiliza técnica como base B^+-Tree; e
- podem ser agregados índices regulares.

5.2 NB^+-Tree e NR-Tree

Em [LÜ95], duas estratégias de indexação para acessar dados temporais são mostradas, NB^+-Tree e NR-Tree. São variações de B^+-Tree e R-Tree, respectivamente. Nesta seção primeiramente serão apresentados os conceitos das técnicas NB^+-Tree e NR-Tree, indicando as características principais de cada uma. Em seguida é apresentada uma modelagem UML de ambas as técnicas procurando já conseguir alguma padronização, com vistas a obter, no final, um modelo unificado de todas as técnicas.

Dados temporais de uma dimensão podem ser vistos como um caso especial de dados temporais multidimensionais. Por um lado, para responder eficientemente a consultas multidimensionais é preciso um índice explícito multidimensional. Este índice é essencial para agrupar dados n-dimensionais em áreas contínuas. Entretanto, não existe uma forma perfeita para mapear dados multidimensionais em uma forma linear física de armazenamento em disco. Por outro lado, um método de indexação de uma dimensão, por exemplo a B^+-Tree, é muito eficiente para responder consultas a uma região indexada. Entretanto, se os valores multidimensionais são envolvidos, cada intervalo de consulta precisa usar os índices que resolvem esse intervalo, e depois fazer a interseção das respostas de cada índice. Esta solução é altamente ineficiente. Lü apresenta uma solução em que, quando existe dado temporal de uma dimensão, como em muitas aplicações, um método de indexação de uma dimensão pode ser usado.

Os dois métodos propostos em [LÜ95], NB^+-Tree (*non-deletion* B^+-Tree) e NR-Tree (*non-deletion* R-Tree), suportam dados temporais de uma dimensão e dados temporais multidimensionais, respectivamente.

5.2.1 NB^+-Tree

NB^+-Tree é uma técnica de indexação para dados temporais de uma dimensão. Atua especificamente sobre tempos de validade e tem a técnica B^+-Tree como base de indexação, tal como sugere seu nome.

Duas alterações foram feitas na B^+-Tree. Primeiramente, a estrutura do índice

passou a contemplar o histórico, sendo adicionado, em cada entrada do índice, um intervalo de tempo. Quando uma nova entrada é criada, o ponto inicial é informado e o ponto final é representado por "*", símbolo que serve também para mostrar que esta entrada é a corrente. Quando esta versão (*) tornou-se história, o tempo de *update* ou de *delete* é informado para o ponto final. Em segundo lugar, como as versões antigas do mesmo objeto estão mantidas na NB$^+$-Tree, diferentes versões da mesma entrada podem existir em mais de uma subárvore. Então, quando a consulta procura toda história de um objeto, é feito acesso em mais de uma subárvore.

A técnica NB$^+$-Tree possui dois algoritmos diferentes de busca, o primeiro para a versão corrente e o segundo para retornar a versão histórica da informação.

O **algoritmo de busca da versão corrente** é usado para procurar entradas com valor de chave em uma NB$^+$-Tree. Para a versão corrente, o valor tempo do final do intervalo temporal é representado pela marca final (*). Em cada nodo existe um conjunto de chaves de busca e suas respectivas informações temporais, e ainda os ponteiros que apontam para o próximo nível formando a estrutura de árvore. O processamento começa na raiz. Se o tipo de nodo é folha, então verifica se o valor da chave procurada é menor que o valor de chave do nodo, se for o processamento segue para o próximo nível da árvore. Caso o valor procurado seja maior ou igual ao valor de chave feito na verificação, o processamento segue para os nodos apontados pelo ponteiro maior ou igual da chave de busca. Esse processo é repetido até encontrar o nodo em que a chave de busca é igual ao valor requerido. Por último é verificado, somente para os valores em que a igualdade seja verdadeira, se o marcador da versão corrente (*) se faz presente.

O **algoritmo de busca para toda história** de um objeto é usado para retornar a versão específica, existindo dentro de um intervalo temporal *(Tbegin, Tend)* de uma entrada com valor de chave. As principais diferenças entre o algoritmo da versão corrente e o algoritmo de busca para toda história são: (i) depois de achar o valor de chave encontrado na árvore, a procura não pára, continua até que o valor de chave seguinte não seja o procurado, e (ii) todas subárvores as quais contém o valor de chave e o intervalo temporal pertence ao requerido, precisam ser processadas por completo.

5.2.1.1 Proposta de modelo para NB$^+$-Tree

Na figura 5.3 é apresentado o modelo UML da técnica NB$^+$-Tree. A classe **ComposiçãoNodoTV** é a mesma apresentada no modelo do *Time Index*. A classe subseqüente **TimeInformation**, foi criada para sobrescrever algumas idéias dos elementos temporais. Isto se faz necessário em virtude do algoritmo de busca diferenciar a versão corrente da busca de toda história pelo caractere especial '*'. E para não utilizar em todo modelo esse caracatere, particular dessa técnica, resolveu-se isolá-lo em uma classe pertinente somente à modelagem NB$^+$-Tree. Possui os seguintes métodos:

- **CriaçãoNovaEntrada:** serve para criar uma nova entrada no índice e por conseqüência ativa os método **SetaTempoValidade**;
- **FinalizaTVEnd:** quando existe a necessidade de finalizar o tempo de validade esse método é ativado, tirando a marca de versão corrente e

colocando um valor para o tempo final; e

- **SetaTempoValidade:** corresponde à ação de atribuir o tempo para o rótulo temporal de validade.

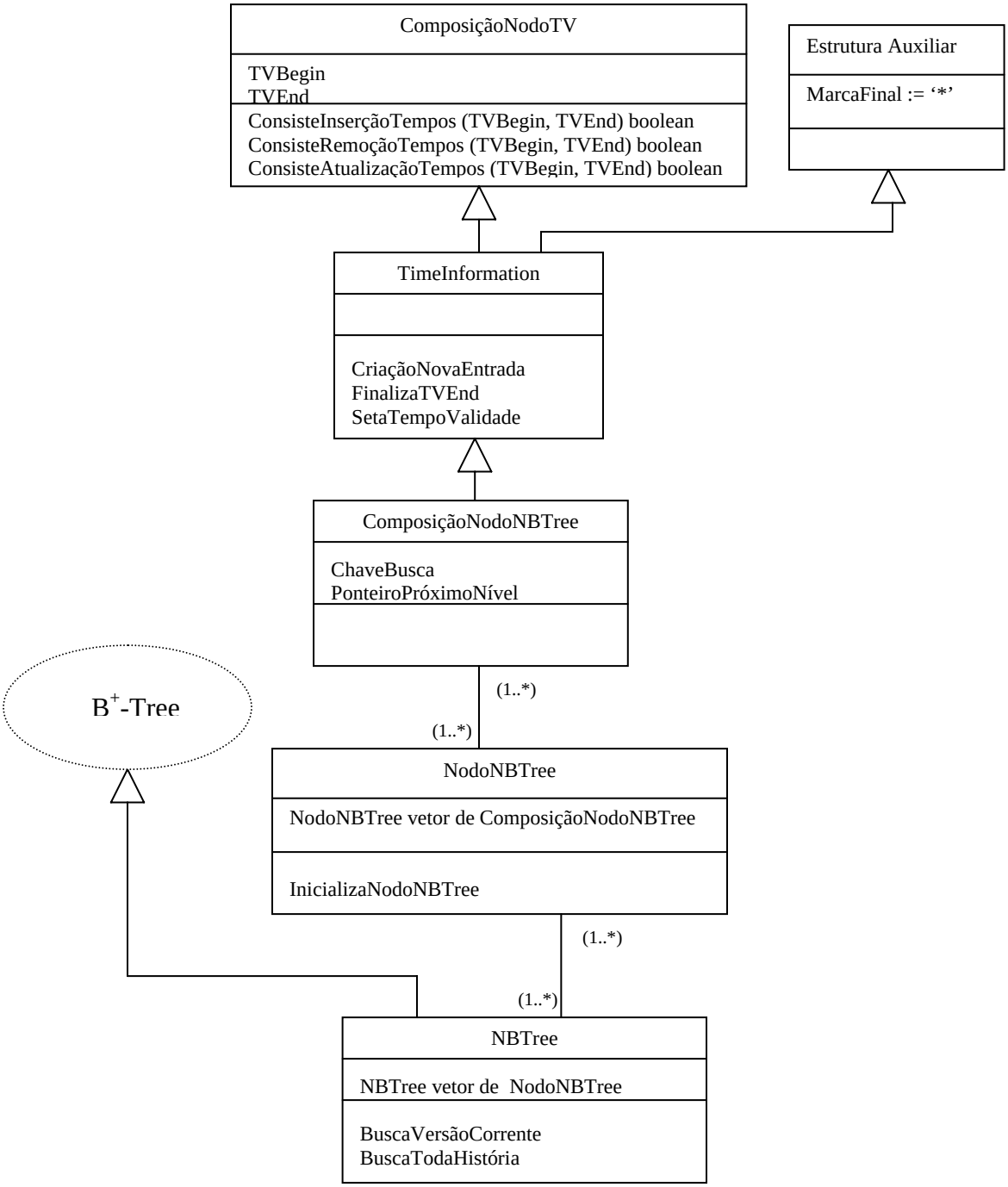


FIGURA 5.3 – Modelo UML da NB^{+}-Tree

Quando uma nova estrutura é criada, o ponto inicial do tempo de validade é informado e o tempo final é representado por um marcador (‘*’). Este passo é realizado pelo método **CriaçãoNovaEntrada**, que por sua vez, ativa o método **SetaTempoValidade**. O método **FinalizaTVEnd** corresponde a informar o tempo final dizendo que aquela tupla não é mais a versão corrente.

A classe **EstruturaAuxiliar** possui o atributo **MarcadorFinal** para facilitar a representação da versão corrente.

A classe **ComposiçãoNodoB+-Tree** tem o intuito de compor a entrada do nodo, formada pela chave de busca, o ponteiro para o próximo nível e os tempos de validades herdados da classe acima na hierarquia do modelo.

A classe **NodoNBTree** consiste no vetor que formará a árvore de indexação na classe subseqüente **(NBTree)**. Para se formar o vetor corretamente foi usado o relacionamento de associação entre as classes. O método pertinente a esta estrutura denominado de **InicializaNodoNBTree** corresponde ao método padrão de inicialização dos componentes.

A classe **NBTree** utiliza as mesmas características da B^+-Tree, como sugere o nome. A principal diferença está na formação do nodo de cada entrada. Por este fato, pode-se então fazer uma hierarquia da B^+-Tree e da NB^+-Tree para aproveitar as características já descritas na superclasse, apenas utilizando a sobreposição de métodos para adaptar as melhorias da técnica NB^+-Tree. Esta classe representa o cerne do modelo. É composta por um vetor dos nodos criados na classe **NodoNBTree**. A classe forma a estrutura em árvore, e como a estrutura é hierárquica, a composição desta árvore é composta por nodos os quais são formados pelos ponteiros pertinentes à estrutura e pelos tempos da classe **TimeInformation**. Os métodos dessa classe são:

- **BuscaVersãoCorrente:** corresponde ao algoritmo de busca da versão corrente explicado anteriormente (seção 5.2.1); e
- **BuscaVersãoTodaHistória:** corresponde ao algoritmo de busca da versão corrente explicado anteriormente (seção 5.2.1).

5.2.1.2 Características principais da NB^+-Tree

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa tempo de validade;
- utiliza B^+-Tree;
- difere busca pela versão corrente da busca de dados históricos; e
- possui caractere especial para definir o final da tupla de validade, identificando a versão corrente.

5.2.2 NR-Tree

Uma variação de R-Tree é proposta por Lü em [LÜ95], chamada de NR-Tree. Uma dimensão é dedicada para suportar a identificação da entidade. Na NR-Tree, cada

entrada é usada para representar o estado de uma entidade. Dentro de cada entrada existe um identificador único de entidade, um ponteiro para folhas, o qual é o endereço de um nodo menor, e um retângulo n-dimensional, o qual cobre todos os retângulos dos nodos menores das entradas. No nodo folha existe um retângulo n-dimensional o qual é denominado de *bounding box*, um identificador único e seus endereços para os índices secundários ou para a própria entrada quando uma NR-Tree é usada como índice primário. Um exemplo de NR-Tree é apresentado nas figuras 5.4 e 5.5.

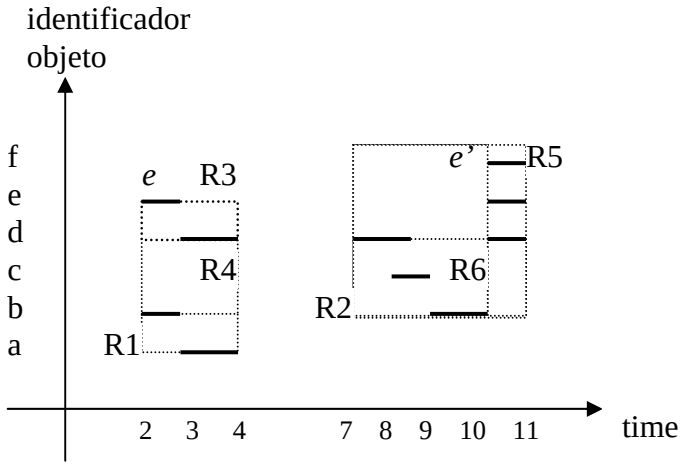


FIGURA 5.4 – Gráfico de Exemplo para NR-Tree [LÜ95]

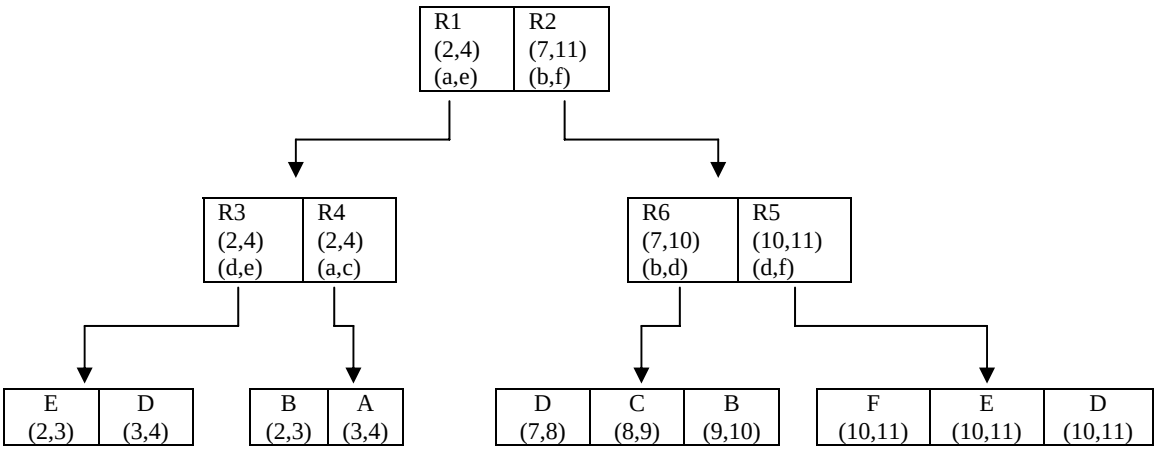


FIGURA 5.5 – Exemplo de NR-Tree [LÜ95]

O algoritmo de busca da NR-Tree é utilizado para retornar as versões específicas de objeto com especificação de tempo. Dado uma NR-Tree achar uma entrada *o*(*e*, *t*), onde *e* é um identificador de um objeto e *t* é o retângulo de tempo no qual *e* existe.

O algoritmo começa seu processamento pela raiz e verifica (1) se o nodo não é folha, caso seja verdadeiro (não seja folha), verifica para cada entrada pelo nodo não

folha se o objeto requerido é igual ao objeto representado pelo nodo ou se o objeto está contido no objeto representado pelo nodo, caso seja verdadeiro, o processamento é direcionado para o próximo nível e ativa o início da pesquisa, agora partindo da verificação (1) e assim recursivamente até encontrar os nodos folhas. Quando encontrar os nodos folhas, é verificado se as entradas do objeto requerido é igual ao objeto representado pelo nodo, caso seja verdadeiro essa é a entrada procurada.

O NR-Tree é uma estrutura eficiente para dados temporais de múltiplas dimensões. A principal razão pela qual NR-Tree apresenta melhor desempenho em relação à R-Tree é que a primeira não indexa sobreposição de valores e tempo ramificado, o que geralmente não acontece, enquanto a segunda (R-Tree) faz uso de algoritmos para indexar esta ocorrência.

5.2.2.1 Proposta de modelo para NR-Tree

Na figura 5.6 é apresentada a modelagem UML desta técnica.

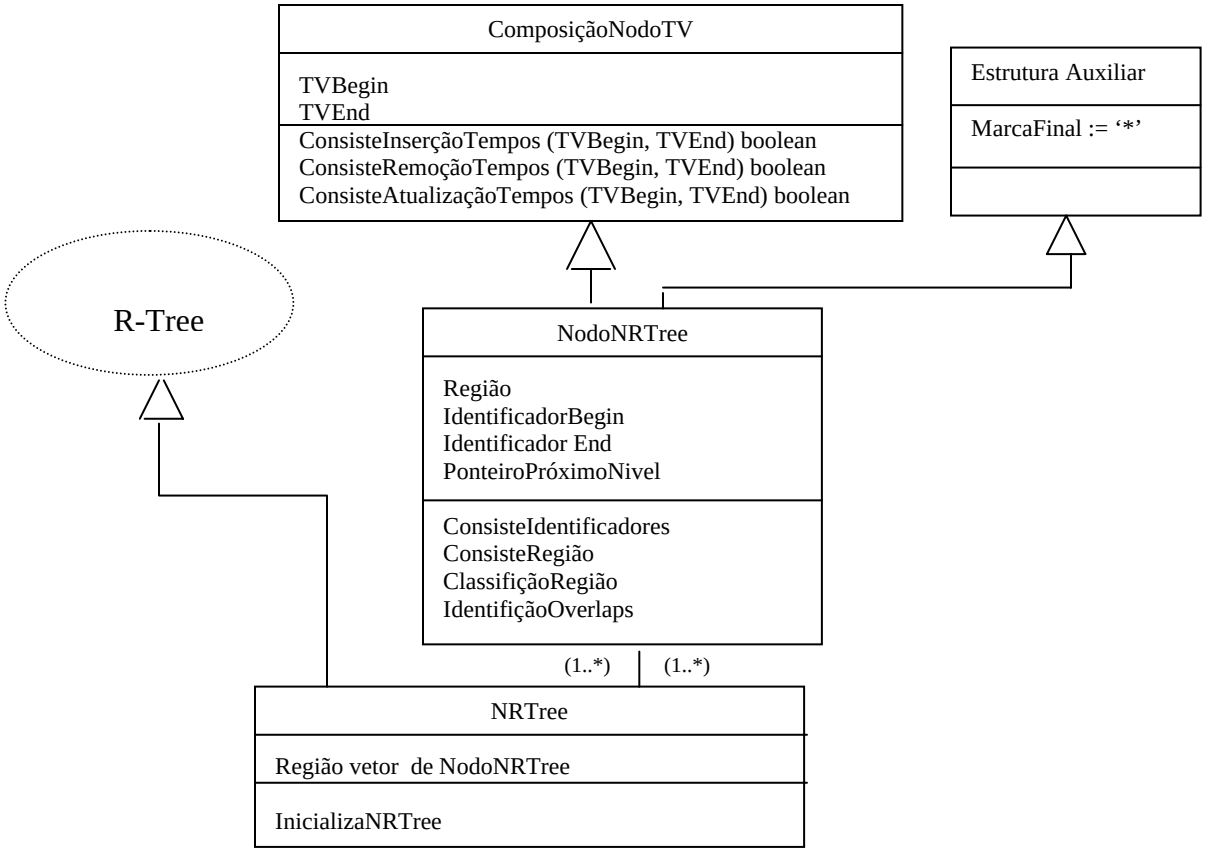


FIGURA 5.6 – Modelo UML da NR-Tree

A classe **NodoNRTree** implementa a região multidimensional formada pelos tempos iniciais e finais no eixo dos tempos, e dos identificadores das regiões no eixo das coordenadas. Desta forma são mapeadas regiões as quais representam dados temporais. Sendo assim, pode-se utilizar os algoritmos da técnica R-Tree. Possui os seguintes

métodos:

- **ConsisteIdentificadores:** verifica se os identificadores são únicos;
- **ConsisteRegião:** consiste as entradas no que tangente à semântica dos valores que irão formar a região dimensional;
- **ClassificaRegião:** organiza, segundo algoritmos de manipulação de regiões cartesianas, as regiões representantes dos dados temporais. Este método invoca o método **Identifiçãoverlaps**; e
- **Identifiçãoverlaps: i**dentifica sobreposição de valores, talvez um dos métodos mais críticos desta técnica, porque é responsável por não deixar acontecer sobreposição das regiões indexadas. Caso ocorram regiões sobrepostas, caracterizando dado com mais de uma validade, pode ocorrer a falha do índice por completo, uma vez não está preparado para este caso.

A classe **NRTree** codifica a estrutura em árvore a qual, por associação da classe **NodoNRTree**, herda da classe R-Tree as características e os métodos para processar dados espaciais que são representados por esta associação. Possui apenas o método padrão de inicialização denominado de **InicializaNRTree**.

5.2.2.2 Características da técnica da NR-Tree

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa tempo de validade;
- utiliza a R-Tree;
- controla regiões temporais como regiões espaciais, mas sem sobreposição de valores (não indexa tempo ramificado); e
- utiliza um marcador final para tempo de validade.

5.3 B^+-Tree para BDTs

Cheng Goh, em [GOH96], define uma estratégia para mapear dados temporais em uma forma linear. O intuito deste mapeamento é poder utilizar uma técnica de indexação tradicional, como a B^+-Tree, para indexar os dados temporais agora "lineares".

Nesta implementação, uma relação temporal é mapeada para pontos em espaços multidimensionais, com cada intervalo de tempo sendo transposto para uma coordenada de duas dimensões. Assim, a seleção temporal é constituída como uma operação de busca espacial.

Goh sugere uma estratégia de acesso para a indexação temporal, a qual é baseada na técnica de indexação espacial. Muitos estudos têm surgido para buscas envolvendo dados espaciais, porém é provado que somente estes índices não suportam dados temporais eficientemente. Isto porque, diferentemente dos objetos espaciais, os

quais aumentam a extensão de sua área, os dados temporais são linhas paralelas aos eixos de tempo e não têm aumento de extensão, apenas na sua forma linear. Portanto, somente índices como R-Tree e R*-Tree, que são destinados para manipular objetos espaciais, não podem manipular dados temporais; deve sim, existir um suporte para que, com o auxílio destas técnicas sejam manipulados dados temporais.

Este suporte pode ser resolvido como uma transformação de TV e/ou TT para um sistema de coordenadas de duas dimensões. Feito este mapeamento, qualquer seleção temporal pode ser recuperada através de uma operação de busca espacial em uma área de espaço multidimensional. Goh definiu este mapeamento, com IST – *Interval Spatial Transformation*.

A transformação IST de uma tupla é definida como: IST(tupla) = (*TVbegin*, *TVend – TVbegin*), onde *TVbegin* é o início do tempo de validade, *TVend* é o final do tempo de validade e *TVend – TVbegin* representa a duração do tempo de validade.

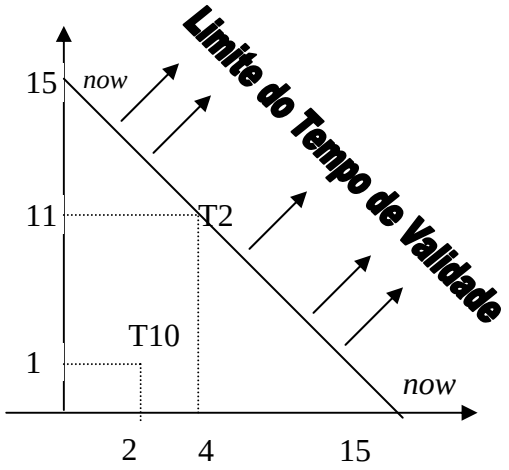


FIGURA 5.7 – Representação da Transformação IST de uma tupla [GOH96]

Na figura 5.7, para uma tupla "T10" com o tempo de validade de [2,3], a transformação IST corresponde ao ponto cartesiano (2,3-2), logo (2,1). Assim, pode-se fazer um mapeamento de tempos de validade para um espaço multidimensional. No caso das tuplas que possuem o tempo de validade final *now*, é definido um valor para esta variável como sendo o exato instante do *snapshot* mais recente, por exemplo 15, conforme tabela 5.2.

TABELA 5.2 - Determinação da Coordenada Y para Tuplas com *TVend "now"*

Tupla	Coordenada X	Função - x + y=b	Coordenada Y
T2	4	4 + y=15	11
T11	8	8 + y=15	7
T12	10	10 + y=15	5
T13	12	12 + y=15	3
T14	11	11 + y=15	4

Após a determinação dos pontos cartesianos do mapeamento, pode-se fazer algumas observações somente com a visualização do gráfico, como por exemplo:

- qualquer tupla com o *TVbegin* = *a* deve estar abaixo da linha *x* = *a*;
- qualquer tupla com o *TVend* = *b* deve estar abaixo de *x* + *y* = *b;* e
- qualquer tupla com o duração do tempo de validade = *c* deve estar abaixo da linha *y* = *c*.

Alguns exemplos de buscas seguindo a transformação IST, conforme figura 5.8, podem ser dados seguindo as informações de entrada e saída de turistas nos EUA:

(i) recuperar todas pessoas que entraram nos EUA no dia *Ta* ou antes;

(ii) recuperar todas pessoas que deixaram os EUA no dia *Tb* ou antes;

(iii) recuperar todas pessoas que permaneceram nos EUA por um total de *Tc* dias ou menos;

(iv) recuperar todas pessoas que entraram nos EUA no dia *Ta* ou antes e que deixaram no dia *Tb* ou antes;

(v) recuperar todas pessoas que entraram nos EUA no dia *Ta* ou antes ou que permaneceram nos EUA por um total de *Tc* dias ou menos; e

(vi) recuperar todas pessoas que foram aos EUA no dia *Tf*.

Estas perguntas podem ser mapeadas nas figuras 5.8a, 5.8b, 5.8c, 5.8d, 5.8e 5.8f, respectivamente.

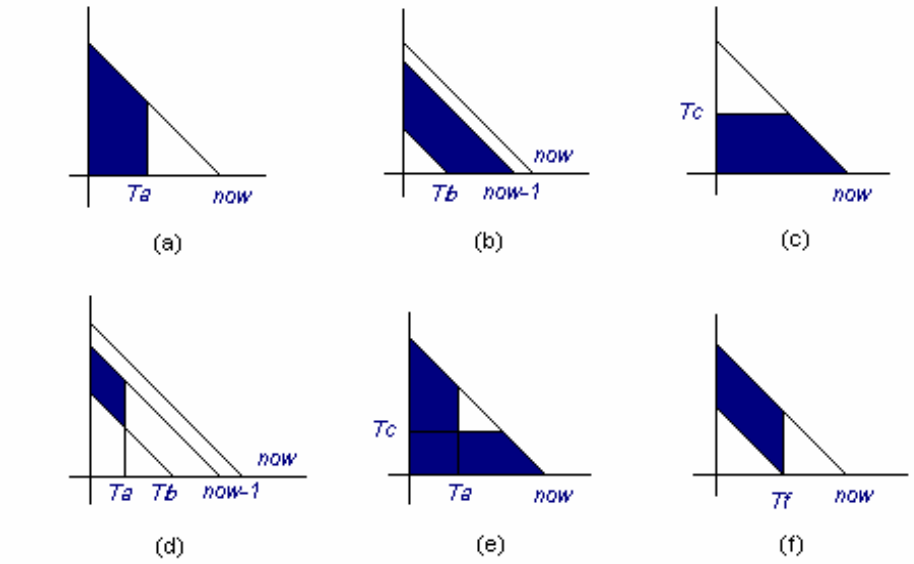


FIGURA 5.8 – Seleções Temporais Mapeadas para Sistema Bidimensional [GOH96]

Uma vez mapeadas as seleções temporais para uma forma multidimensional, pode-se, então, aplicar uma forma de indexação para dados espaciais. Apesar disto, Goh

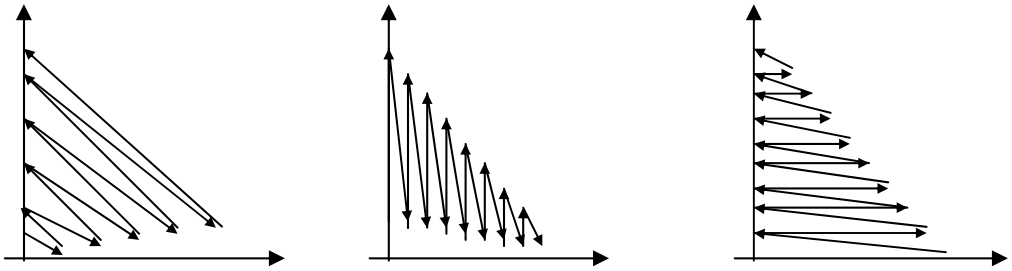
explora uma alternativa para indexar estes dados espaciais com uma característica peculiar de buscas temporais.

Especificamente, é demonstrado que, dependendo do tipo de seleção temporal mais freqüentemente requisitado, pontos em espaços multidimensionais podem ser mapeados para uma forma linear. Conseqüentemente, pode-se utilizar uma forma clássica de indexação, como por exemplo, a B^+-Tree.

Goh sugere três formas de ordenação, segundo a seleção temporal mais freqüentemente requisitada, tais como:

- (D)iagonal ;
- (V)ertical, e
- (H)orizontal.

Uma representação bidimensional, como na figura 5.9, pode ser disposta em uma forma linear, apenas especificando a sua ordem linear.



(a) – Diagonal (b) – Vertical (c) - Horizontal

FIGURA 5.9 – Formas de Ordenação Linear

Considerando a representação espacial do figura 5.9, pode-se adotar um mapeamento para uma forma linear seguindo um dos três tipos de ordenação. Por exemplo, se for adotada a (D)iagonal, tem-se a seguinte ordenação: t1(0,3) < t10(2,1), porque é a mesma linha x + y = b (0+3 = 3 e 2+1 = 3), porém 0 < 2; t8(8,1) < t2(4,11), porque 8+1 < 4+11, assim sucessivamente, conforme figura 5.10. A partir desta ordenação linear, pode-se então aplicar a técnica B^+-Tree.

Na verdade, segundo as consultas mais freqüentemente requisitadas, pode-se determinar a melhor ordenação para o mapeamento linear aumentando a eficiência da técnica.

Goh afirma ter duas vantagens básicas na sua técnica de indexação:

- o mapeamento de uma relação temporal para um espaço multidimensional promove um *framework* uniforme para o tratamento com consultas temporais envolvendo TV e TT, bem como atributos não-temporais; e

- estando sob uma forma linear, a pesquisa em espaços multidimensionais permite utilizar um modelo clássico de indexação, a B^+-Tree, que significa que o suporte para seleções temporais pode ser efetuado sem modificação na camada física dos componentes do DBMS.

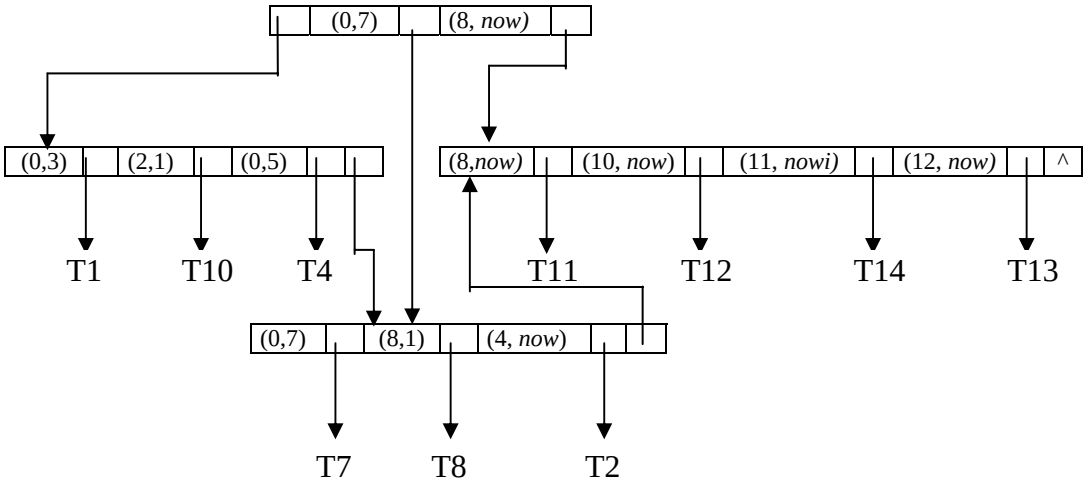


FIGURA 5.10 – Representação Espacial Usando B^+-Tree c/ Ordenação Diagonal

5.3.1 Proposta de modelo de B^+-Tree para banco de bados temporais

Na figura 5.11 é apresentada a modelagem UML desta técnica. A modelagem desta técnica é apresentada na figura 5.11. O fator primordial deste modelo UML é que está calçada na modelagem da B^+-Tree anteriormente apresentadas, e contempla as características básicas da técnica de Goh.

A modelagem apresentada tem como início a definição do tipo de BDT que será contemplado. Como demonstração, foi modelada a composição do nodo com tempo de validade e/ou tempo de transação, traduzidos nas classes **ComposiçãoNodoTV** e **ComposiçãoNodoTT**. As classes **ComposiçãoNodoTV** e **ComposiçãoNodoTT** possuem basicamente os tempos inicial e final de validade e de transação, respectivamente.

A classe **EstruturaAuxiliar** pode ser entendida como o local onde se encontram os atributos e métodos gerais da modelagem, como por exemplo, o conceito da variável *now*, que expressa o momento atual.

A classe **NodoGoh** define o tipo de nodo que esta técnica utiliza. Consiste em um vetor do tipo de **ComposiçãoNodoTV** e **ComposiçãoNodoTT**. Os métodos dessa classe são validações dos tempos de transação e validade, bem como a transformação IST da técnica de Goh:

- **ValidaTT:** método para consistir se os tempos de transação estão corretos para sofrer a transformação IST. Embora já exista alguma consistência nos

métodos da classe **ComposiçãoNodoTT**, no método **ValidaTT** pode ser feita alguma verificação mais específica da transformação IST;

- **ValidaTV:** igualmente ao método **ValidaTT**, serve para consistir se os tempos de validade estão corretos para sofrer a transformação IST;

- **TransformaçãoIST:** método que realiza a transformação dos tempos (TV e/ou TT) para uma coordenada cartesiana. Assim é possível fazer um mapeamento de regiões multidimensionais para que técnicas de indexação tradicionais possam ser utilizadas; e

- **VoltaTransformaçãoIST:** esse método deve reverter a fórmula de transformação IST, partindo do ponto cartesiano e calculando os tempos de validade e/ou transação.

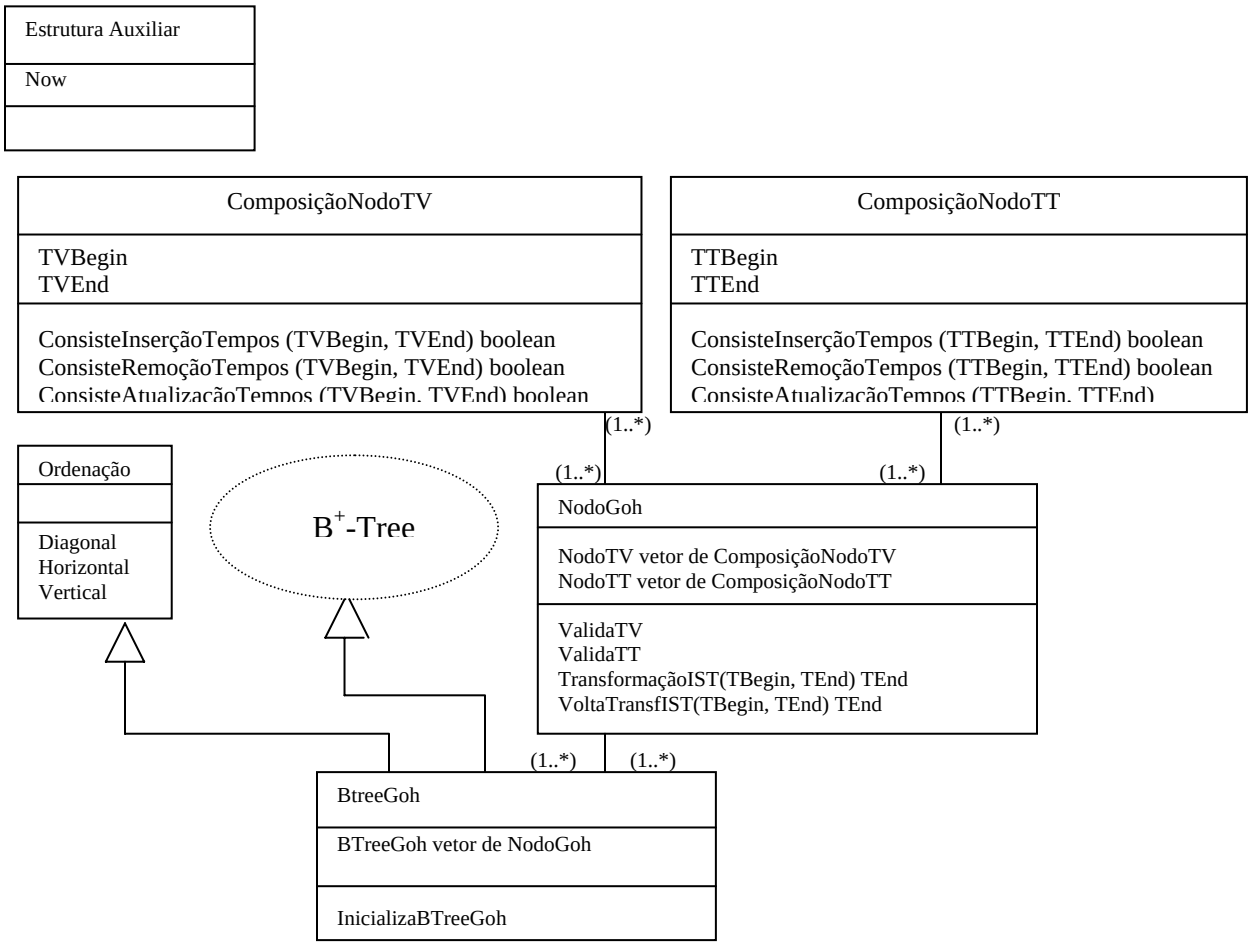


FIGURA 5.11 – Modelo UML da B^+-Tree para Dados Temporais

Quando o vetor dos tempos é inicializado, é feita a transformação IST dos

respectivos tempos. Existe também o método **VoltaTransformaçãoIST**, que serve tão somente para agilizar o processo de representação temporal dos vetores.

A classe **Ordenação** contempla os algoritmos de busca dos diferentes conceitos que Cheng Goh utiliza. É utilizado diretamente na classe **BtreeGoh**, descrita a seguir. Os três métodos desta classe, **Diagonal**, **Horizontal** e **Vertical** são pertinentes ao encaminhamento da árvore de pesquisa. De acordo com o tipo de encaminhamento pode se obter uma vantagem nas consultas.

O objetivo final desta modelagem é prover uma estrutura de indexação para contemplar a técnica apresentada por Cheng Goh. Isso se faz presente na classe **BTreeGoh**. Essa classe implementa a estrutura em árvore dos vetores de TV e TT, já instanciados e com valores transformados para uma forma linear. Possui o método padrão de inicialização da estrutura em árvore, **InicializaBTreeGoh**.

No que tangente às buscas de valores, a classe **BTreeGoh** herda da classe **Ordenação** os métodos e atributos para tal, bem como também herda da classe B^+-Tree os métodos de busca, podendo desta forma dar maior flexibilidade às buscas.

5.3.2 Características da B^+-Tree para banco de dados temporais

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa tempo de validade e tempo de transação;
- utiliza a B^+-Tree;
- utiliza a variável *now;*
- mapear dados temporais em dados lineares; e
- escolha do tipo de ordenação, segundo as características do sistema.

5.4 GR-Tree

Bliujüte [BLI98] propõe uma extensão de R*-Tree que permite a indexação de regiões de dados que crescem continuamente ao longo do tempo. Esta técnica foi denominada de GR-Tree. Estudos indicam que o melhor desempenho do índice estendido é tipicamente 3 a 5 vezes mais rápido que os índices baseados em R-Tree.

Bliujüte chama uma tupla de *now-relative* se suas informações são válidas até o tempo atual, ou se a tupla é parte do estado corrente do BD. Isto é representado pelo uso de variáveis especiais, *now* e *UC*, as quais demonstram o tempo atual para os atributos *TVend* e *TTend* [BLI98], conforme tabela 5.3.

TABELA 5.3 – Exemplo de *now* e UC para Relação de Empregado/Departamento

Tupla	Empregado	Departamento	*TTbegin*	*TTend*	*TVbegin*	*TVend*
1	João	Administrativo	4/97	*UC*	3/97	5/97
2	Tom	Gerência	3/97	7/97	6/97	8/97
3	Jane	Vendas	5/97	*UC*	5/97	*Now*
4	Júlia	Vendas	3/97	7/97	3/97	*Now*
5	Júlia	Vendas	8/97	*UC*	3/97	7/97
6	Ana	Gerência	5/97	*UC*	3/97	*Now*

Os dados *now-relatives* em TT produzem um retângulo que cresce na direção do TT, à medida que o tempo passa. Se existe TV e TT, sendo ambos *now-relative*, eles crescem em ambas direções, fazendo um crescimento em escada (figura 5.12). Dados *now-relative* são dados cujos finais de TV e TT não são fixos, mas são sempre crescentes, nunca decrescentes. Informações podem ser gravadas no BD depois delas se tornarem verdadeiras. Neste caso, tanto TV e TT são *now-relative*. É também possível gravar informações no BD antes de se tornarem verdadeiras.

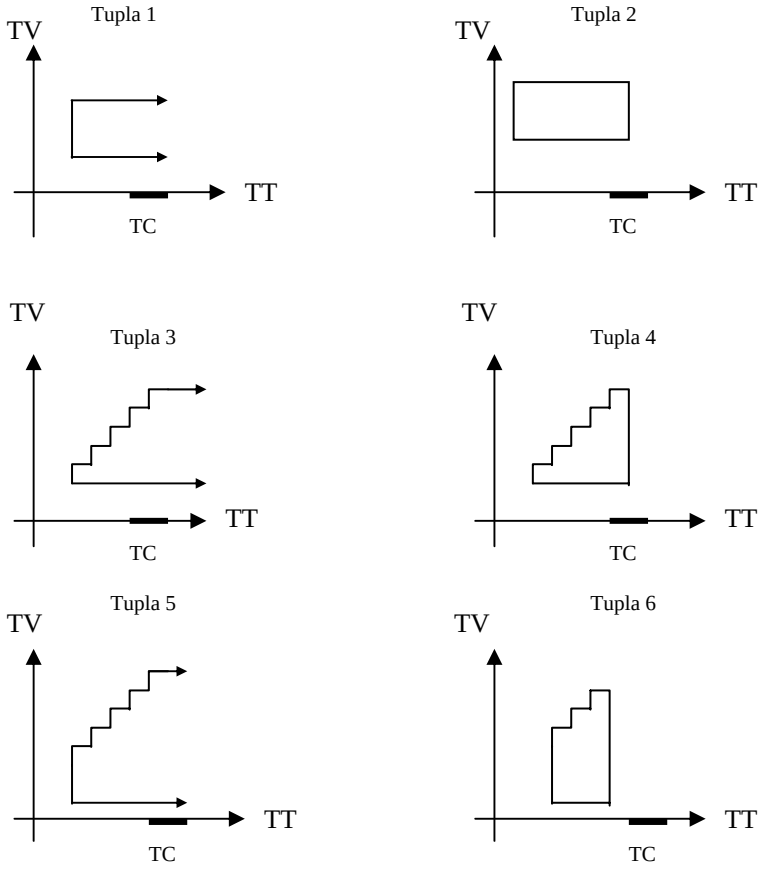


FIGURA 5.12 - Regiões Bitemporais

Com esses dados *now-relatives* representados bi-dimensionalmente, pode-se fazer uma analogia de retângulos espaciais, possibilitando o uso de técnicas de indexação específicas, em geral R-Tree e R*-Tree.

Bliujüte em [BLI98] enfoca a indexação para BDTs como uma forma de se construir índices para BDTs fazendo a indexação do tempo de validade parcialmente persistente [DRI89]. A árvore de intervalos de dados temporais representa esta forma [KUM95]. Outra forma é visualizar os dados temporais como um caso especial de dado espacial e adaptar índices destes dados para dados temporais. Este caminho é o mais discutido atualmente e é o escolhido por Bliujüte.

Todas as variantes da R-Tree tentam minimizar a sobreposição dos *bounding rectangle* (retângulos de limites das áreas indexadas) dos dados de cada nível da árvore e diminuir o espaço morto no *bounding rectangle*. Minimizar a sobreposição é reduzir a ramificação de I/O de pesquisa em diversas subárvores. Minimizar o espaço morto é reduzir a probabilidade de se pesquisar desnecessariamente os dados não qualificados para a pesquisa.

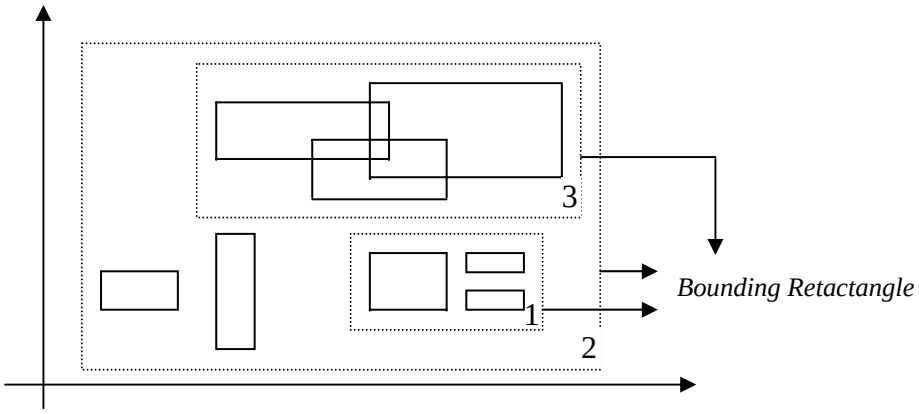


FIGURA 5.13 – Representação de *Bounding Rectangle*

R*-Tree é promissor para um índice de dados bitemporais, porém não é aplicável diretamente, porque representa apenas retângulos estáticos. É consenso que em dados temporais os retângulos crescem linearmente para TT e em escala para TV, precisando uma adaptação para o uso da R*-Tree.

Kumar em [KUM95] propõe um novo caminho para lidar com o TT *now-relative*, mas não endereça o TV *now-relative*. Em seus estudos denominados de 2-R, são usadas duas R-Tree. A primeira R-Tree indexa todos os retângulos crescentes, enquanto que a segunda indexa todos os retângulos estáticos. Observando que todos os retângulos crescentes estão na primeira R-Tree e que todos eles terminam no tempo corrente, Kumar mostra que o armazenamento de *TTbegin* apenas com intervalos fixos de TV na frente da árvore é adequado para suportar TT *now-relative*. Em [KUM95], Kumar contempla bem o TT *now-relative*, porém o TV *now-relative* permanece em aberto.

Para tentar indexar o TV *now-relative*, Bliujüte propõe o GR-Tree, através de duas variáveis especiais *NOW* e *UC* para representar TT e TV, respectivamente, e tentar diminuir o espaço morto e o crescimento da região temporal fora do *bounding rectangle*, conforme figura 5.13.

Utilizando as variáveis *NOW* e *UC* em todos os níveis da árvore, torna-se possível determinar a exata geometria das regiões bitemporais nos nodos folhas e gerar os

minimum bounding rectangles, que crescem quando a região interna também cresce. Com esta extensão, o conteúdo da árvore não difere significativamente da original R*-Tree. Um nodo folha contém quatro rótulos temporais, referindo-se à região bitemporal, e um ponteiro para o atual dado bitemporal carregado no BD. No nodo folha são acrescidos um *flag* chamado de *Hidden* e um ponteiro para o nodo filho. Os rótulos temporais representam um *bounding rectangle* mínimo que engloba todas entradas dos nodos filhos. O rótulos temporais (TT1, *UC*, TV1, *NOW*) representam uma escada nos nodos folhas.

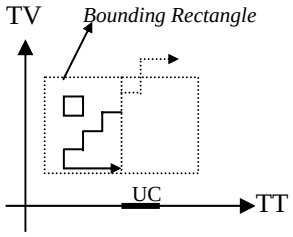


FIGURA 5.14 – Crescimento em Escada fora do *Bounding Rectangle*

Um pequeno crescimento em escada pode ser colocado junto com outras regiões no maior *bounding rectangle*, conforme figura 5.14, tendo um *TVend* fixado. Por vezes, a escada pode crescer para fora do seu *bounding rectangle*, fazendo este retângulo ficar inválido; para manipular esta situação é usado o *flag Hidden*.

Para indicar se os quatros rótulos temporais em um nodo folha referem-se a um *bounding rectangle* mínimo ou a um mínimo *bounding* em forma de escada, foi incluído outro *flag* chamado de *rectangle*, no nodo folha, conforme figura 5.15. Este *flag* é necessário para separar as situações onde se quer representar *TVend* e *TTend*, significando um crescimento escada versus crescimento retângulo.

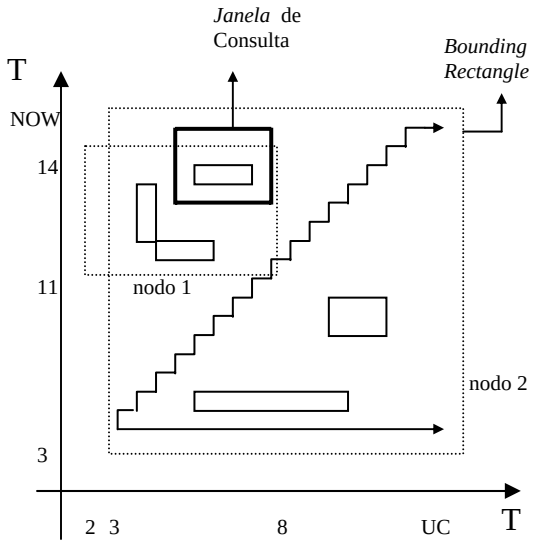


FIGURA 5.15 – Representação do *Bounding Rectangle* [GOH96]

Bliujüte aperfeiçoa o método de pesquisa espacial diminuindo a região a ser

pesquisada. Assim o espaço morto é reduzido significantemente. Como conseqüência direta, tem-se uma região menor como probabilidade de se pesquisar desnecessariamente os dados não qualificados para a pesquisa.

Uma representação da extensão do R*-Tree relativa à figura 5.15 (a) é vista na figura 5.16.

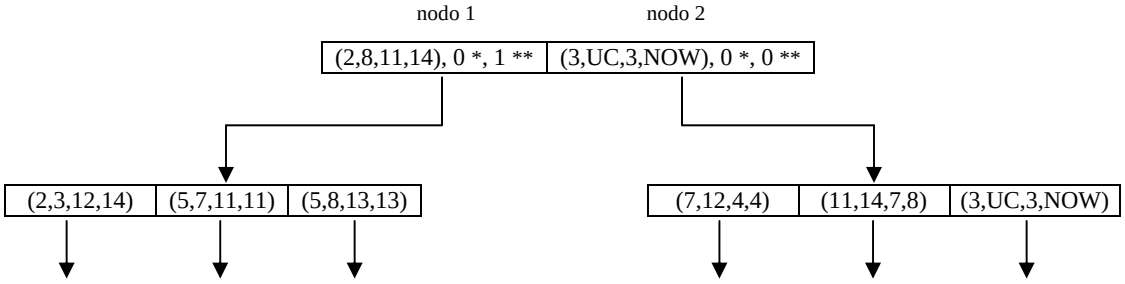


FIGURA 5.16 – Representação GR-Tree da Figura 5.16 [GOH96]

No nodo da raiz da figura 5.16, após as coordenadas da região indexada, não são representados os indicadores de controle da GR-Tree. O primeiro número (*) indica o *hidden* (0 nos dois nodos) e o segundo (**) indica se o crescimento é em escada ou retângulo (1 no primeiro e 0 no último nodo).

5.4.1 Proposta de modelo de GR-Tree

Na figura 5.17 é apresentada a modelagem UML da técnica de Bliujüte. É mais uma contribuição com vistas ao *framework* conceitual apresentado posteriormente. Essa modelagem tem uma importância fundamental por ter como técnica base a R*-Tree, necessitando da sua presença no modelo.

As classes de composição dos nodos que implementam a semântica temporal são modeladas separadamente por ter como primícia básica a ortogonalidade, ou seja, são independentes entre si.

A classe **ComposiçãoNodoGRTree** tem como objetivo formar a composição do nodo da árvore. É constituída pelos tempos da semântica temporal e pelo ponteiro que indica o atual dado no BD.

A classe **NodoGRTree** consiste na formação do nodo que será a estrutura do índice. Herda da classe **ComposiçãoNodoGRTree** os atributos temporais e o ponteiro para o atual dado no BD, sendo agregados os indicadores *hidden* e *rectangle*. São indicados aqui porque são pertinentes ao nodo inteiro e não à composição do nodo. Os atributos **FlagHidden** e **FlagRectangle** servem, respectivamente, para informar se o nodo correspondente fica "escondido", ou seja, se o crescimento ultrapassou o limite da janela de pesquisa, enquanto o outro indica se o crescimento é da forma linear em retângulo, ou seja, se o crescimento dos rótulos temporais (*uc* e *now*) deve ser proporcional. Estes dois *flags* são fundamentais para o bom funcionamento do índice, uma vez que os algoritmos pegam como base esses indicadores para melhor processar os

dados. Na verdade, melhorar o desempenho dessa técnica significa diminuir a região de procura onde provavelmente não serão encontrados valores que satisfaçam a seleção pretendida. Os métodos **SetaHidden** e **SetaRectangle** são rotinas que isolam o nodo para não serem processados e para atualizar os atributos pertinentes.

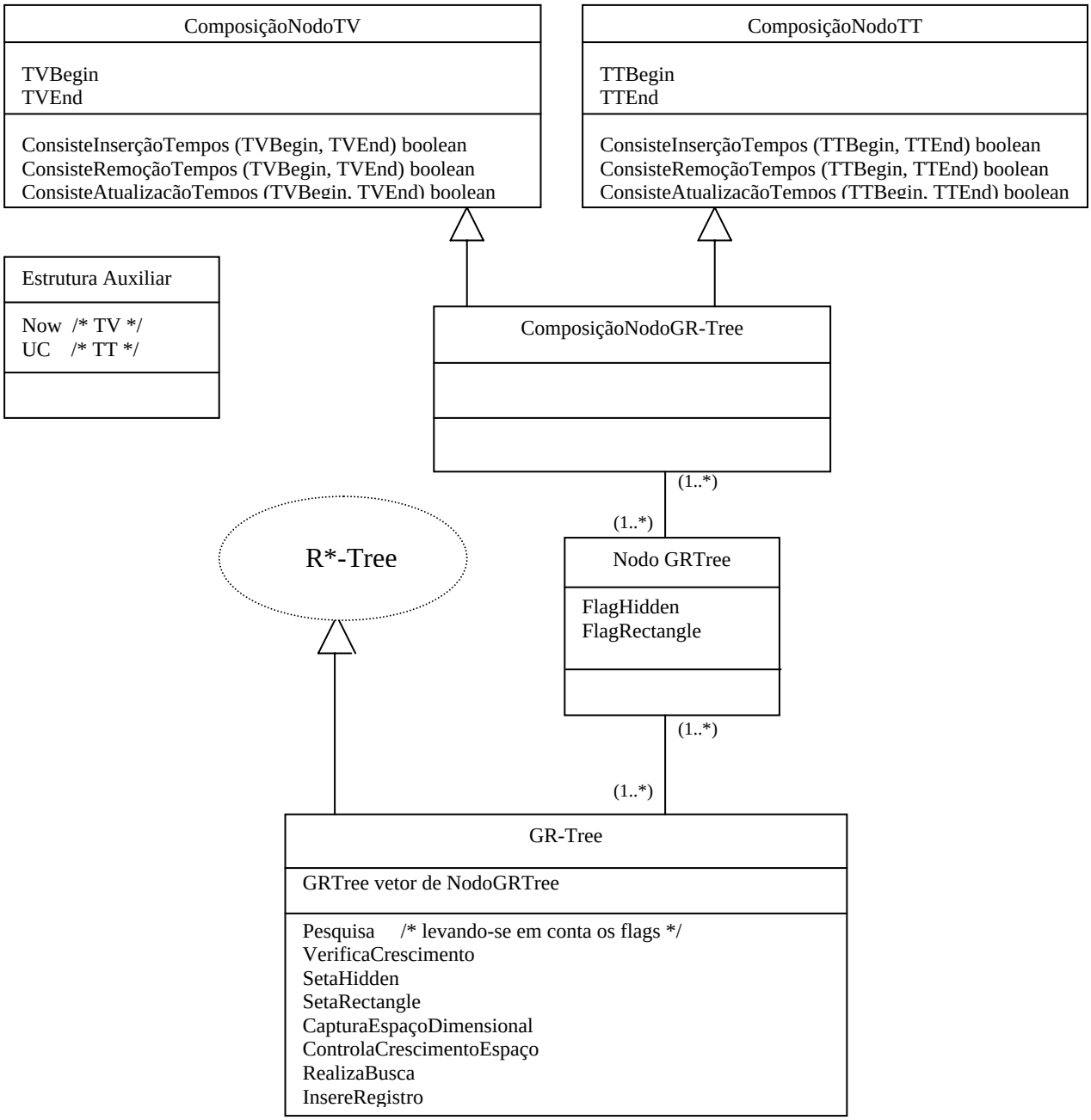


FIGURA 5.17 – Representação GR-Tree no padrão UML

A classe **GRTree** implementa a estrutura em árvore da técnica GR-Tree. É criada a partir da associação da classe NodoGRTree. Os métodos desta classe são os

seguintes:

- **Pesquisa:** esse método contém o algoritmo de consulta e retorna os nodos requeridos pela consulta. É ativa pelo método **RealizaConsulta;**
- **VerificaCrescimento**: é responsável por avaliar o crescimento e identificar a forma na qual os rótulos temporais podem variar, por exemplo, escada, retângulo, entre outras formas. Caso seja detectado algum tipo pertinente ao desempenho da estrutura, é chamado outro método para "marcar" a situação, como acontece se detectado um crescimento retângulo, o qual é determinante quando é invocado o método SetaRectangle. Em analogia, se um crescimento é do tipo retângulo e deixa de ser desse tipo, também devem ser chamados os métodos pertinentes;
- **SetaHidden:** método para marcar a região de consulta como sendo *hidden*, ou seja, essa região não será processada no método de consulta, invocado pelo método **RealizaConsulta**. O método **SetaHidden** é ativado pelo método **VerificaCrescimento**;
- **SetaRectangle:** métod que marca o tipo de crescimento da região como sendo em uniforme em retângulo e não em escada. O método **SetaHidden** é ativado pelo método **VerificaCrescimento**;
- **CapturaEspaçoDimensional**: é responsável por avaliar e capturar com exatidão o *bounding box*, para realizar a consulta dentro da janela mais apropriada sem desperdiçar recursos;
- **ControlaCrescimentoEspaço:** esse método é responsável pela crescimento do espaço dimensional. Ativa o método **VerificaCrescimento** para controlar de forma eficiente o crescimento do espaço;
- **RealizaBusca**: esse método tem a particularidade de utilizar os indicadores de *Hidden* e *Rectangle* para direcionar seu algoritmo de forma a ser mais eficiente do que os algoritmos da R*-Tree. Entretanto, é preciso herdar as estruturas auxiliares que governam esta técnica. Em geral, os métodos mais pretendidos na herança são *split* e o rebalanceamento, pois apesar de serem técnicas totalmente balanceadas dinamicamente, existe a preocupação de obter uma forma de balancear a estrutura novamente; e
- **InsereRegistro**: método que opera a inserção de um dado. É relevante sua diferenciação, uma vez que é preciso atentar às duas principais características da técnica, o indicador *Hidden* e o indicador *Rectangle*.

5.4.2 Características da técnica da GR-Tree

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa tempo de validade e tempo de transação;
- utiliza estrutura auxiliar para determinar os finais dos tempos de transação e de validade;

- utiliza a R*-Tree como base;
- avalia crescimento da região temporal;
- melhorar índice significa diminuir o *bounding box* para não pesquisar em regiões inoperantes; e
- utiliza dois *flags* para determinar o tipo de algoritmo para a pesquisa. Um é relativo ao tipo de crescimento e o outro determina se o crescimento ultrapassou o tamanho da janela para pesquisa.

5.5 M-ITVT

Nascimento, Dunham e Elmasri, em [NAS96], sugerem uma estrutura que se baseia em árvores incrementais de indexação sobre regiões de TV, as quais por sua vez, são apontadas por uma árvore de indexação para os TT. O TV é organizado sobre uma B^+-Tree que indexa eficientemente regiões via um mapeamento de função.

A técnica M-ITVT é oriunda de um aperfeiçoamento do ITVT [NAS95]. Nascimento define TTV como sendo uma árvore balanceada de indexação – B^+-Tree – para indexar o TV. Analogamente, TTT representa uma árvore para indexar o tempo de transação, conforme figura 5.18.

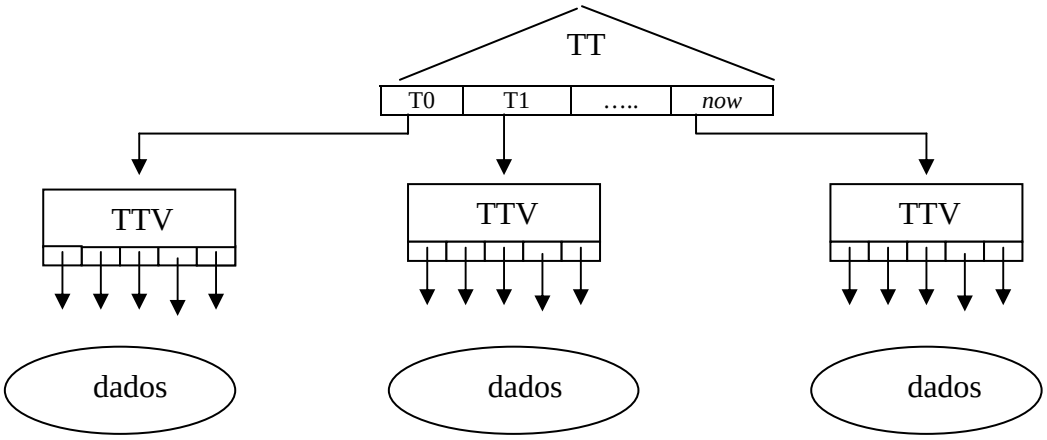


FIGURA 5.18 – Um Índice para Dados Bitemporais [NAS96]

Na figura 5.18 é estipulado um caminho direto para indexar dados bitemporais. Neste exemplo, todos os tempos onde a transação ocorre são indexados pela árvore de tempo de transação (TTT). Assim qualquer estado do BD pode ser acessado em um tempo logarítmico dado pela fórmula Log_d N [HOR84] (como explicado no capítulo quatro) em função do tempo de transação, isto porque a TT (primeiro tempo indexado) tem sua indexação baseada na B^+-Tree. Este fato garante a eficiência do índice.

Uma vez que o estado requerido é recuperado, pode-se então recuperar a validade das tuplas, através dos pontos de TV. Entretanto, para cada passo do TT, é mais provável que a maioria das TTV não sejam modificadas, causando um índice de

redundância muito grande entre as TTV.

Para tentar solucionar esta redundância, Nascimento propõe o ITVT, ou seja, *Incremental Valid Time Tree*. Esta técnica objetiva reduzir o espaço utilizado do índice (figura 5.18). O fato de que em um sistema, o número de vezes que se faz requisições dos estados atuais do BD é mais significativo do que consultas que requerem os estados do passado do BD, faz com o processamento sobre as tuplas mais visíveis seja muito eficiente. Portanto, a idéia é disponibilizar a TTV corrente de maneira mais eficiente. Para os outros casos, é usado um conjunto de *patches*, conforme figura 5.19. Com isto, não é preciso manter a TTT, visto que a TTV corrente é sempre acessada primeiro. O índice ITVT é mostrado na figura 5.19. Todos os *patches* anteriores podem ser acessados através de uma lista simplesmente encadeada, denominada de TTL (figura 5.19).

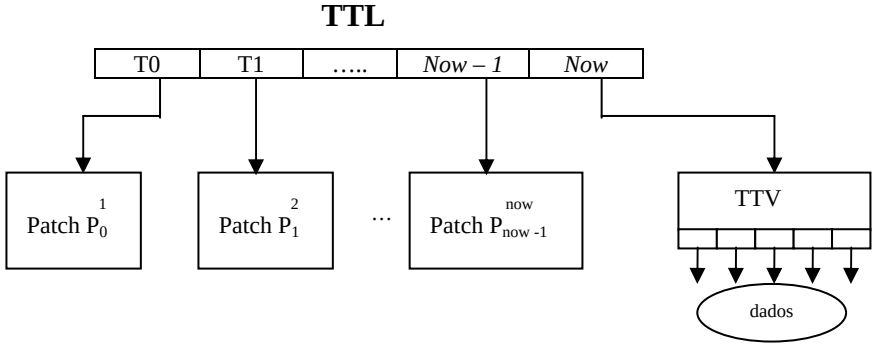


FIGURA 5.19 – ITVT – *Incremental Valid Time Tree*

O M-ITVT – *Multiple Incremental Valid Time Tree* – propõe uma solução para o comportamento ineficiente do ITVT quando tratada uma consulta que envolva estados distantes no passado do BD. Esta técnica sugere um múltiplo ITVT, ou seja, é permitido que muitos TTVs sejam disponíveis para cada tempo de transação, resolvendo a problemática de manter somente o estado mais atual disponível. O M-ITVT reduz de forma significativa o índice de conjuntos de *patches* para ser aplicado no número de estados (TT) entre um estado requerido e o TTV mais próximo.

O M-ITVT é um aperfeiçoamento do ITVT. Ele provê um ganho no processamento de consultas porque não causa tanto *overhead*, devido ao armazenamento do índice ocupar menor espaço, e ainda, porque a recuperação de um estado no passado é mais eficiente.

5.5.1 Proposta de modelo de M-ITVT

A figura 5.20 apresenta a modelagem UML da técnica M-ITVT. A classe **ComposiçãoNodoB+-TRee** é oriunda da modelagem da técnica base B^+-Tree (figura 4.14), utilizada para reaproveitar os seus atributos e agregar à classe **NodoMITTV** o indicador da árvore de indexação dos tempos de validade para cada tempo de transação. Ou seja, esta técnica trabalha com uma árvore para tempos de transação e uma para tempos de validade, para cada tempo de transação indexado.

A classe **NodoMITTV** é responsável por implementar a característica mais importante da M-ITTV. Essa característica consiste em informar, para cada árvore de tempo de transação, qual estrutura no tempo de validade é a corrente. Esse atributo é fundamental, uma vez que para se ter acesso aos dados históricos, o desempenho cai significantemente quando processadas duas árvores de indexação encadeadas.

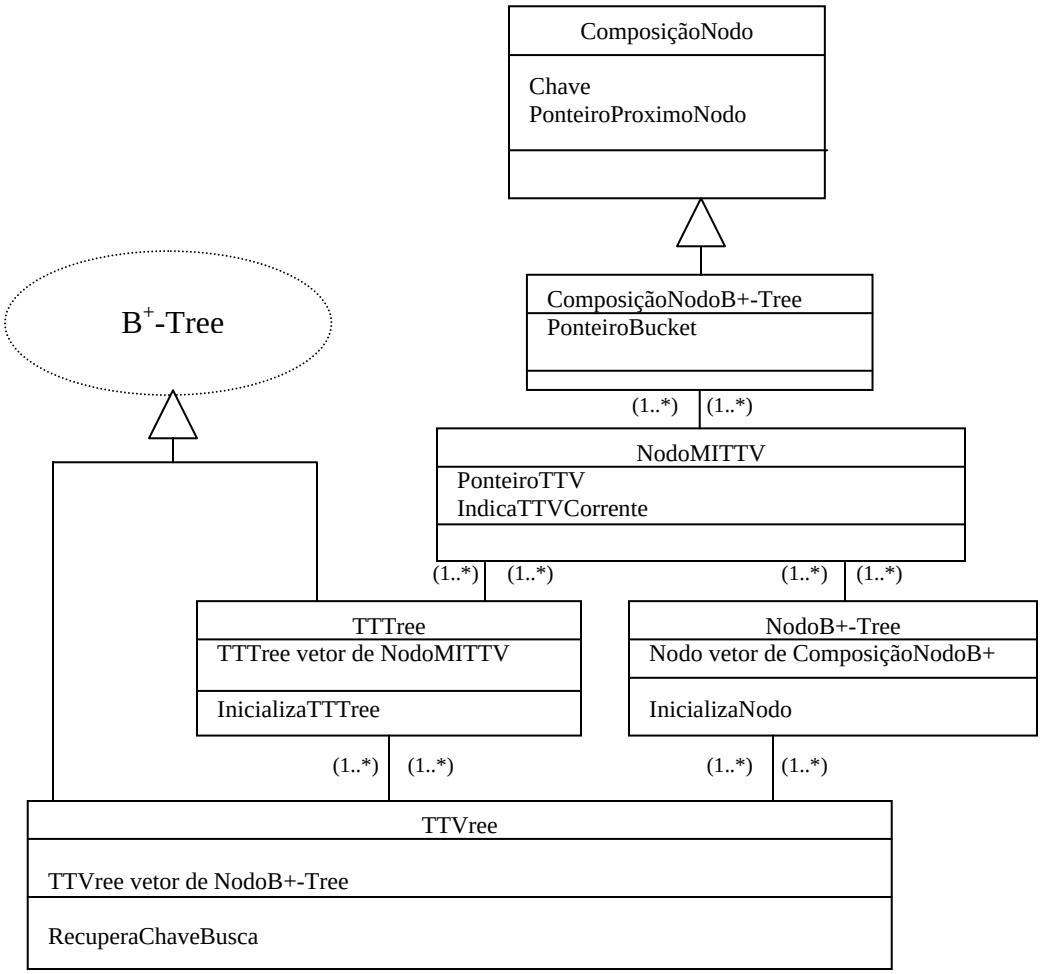


FIGURA 5.20 – Modelo UML do M-ITTV

A classe **TTTree** é responsável por implementar a árvore de indexação dos tempos de transação. Herda de B^+-Tree os métodos operacionais de manipulação da estrutura. Tem a associação da classe **NodoMITTV** para garantir que cada tempo de transação indexado tenha uma árvore válida de indexação dos tempos de validades. Possui o método padrão de inicialização da árvore denominado de **InicializaTTTree.**

A classe **NodoTTVree** é associada à classe **TTTree** porque representa a indexação dos tempos de validade para cada tempo de transação. O nodo é formado por uma especialização da classe **NodoB+-Tree** (figura 4.14) e pelo atributo de associação da árvore do tempo de transação.

A classe **TTVree** implementa o vetor correspondente aos tempos de validades. Garante o desempenho pela função logarítmica (Log_d N) que governa a técnica B^+-Tree. Possui o seguinte método:

- **RecuperaChaveBusca:** método que possui uma integração com os métodos herdados pelo método B^+-Tree. Esse método é redefinido com o intuito de verificar primeiramente na árvore do tempo de validade corrente, para se não encontrar o dados desejado, invocar o método para pesquisa das outras árvores da estrutura.

Como é indexado TV em função do TT, é bem provável que para cada tempo de transação os tempos de validades correspondentes não sejam alterados, o que gera uma ineficiência na lógica porque vai existir buscas em regiões inoperantes. A estrutura da árvore de indexação dos tempos de validade é implementada nesta classe. Tem como base as operações da B^+-Tree e possui como estrutura do nodo os mesmos da B^+-Tree e mais um agregado da associação da estrutura dos tempos de transação.

5.5.2 Características da técnica da M-ITVT

Resumindo, esta técnica de indexação apresenta as seguintes características:

- indexa Tempo de Transação e Tempos de Validade;
- uma árvore para TT e outra para TV;
- utiliza a B^+-Tree como base nas duas estruturas;
- utiliza um marcador para indicar o TV corrente; e
- algoritmo de busca verifica primeiro a versão corrente para depois o resto da estrutura.

5.6 Considerações finais

Neste capítulo foram apresentadas seis técnicas de indexação para dados temporais. Cada técnica possui uma ou mais particularidades que serão relevantes para o modelo final. Sendo assim, foi constituída uma gama de conceitos e atingida uma abrangência de tipos de dados temporais que se julga suficiente para a proposta do *framework* conceitual.

Diante da proposta de modelo de cada uma das técnica de indexação apresentadas, o próximo passo é unir os modelos e refinar um pouco mais as classes para chegar a um modelo único final com um grau de hierarquia de classes satisfatório. Esse modelo é apresentado no capítulo a seguir.

6 Proposto do *framework* conceitual

O grande objetivo do trabalho é, com base nos modelos elaborados para as técnicas de indexação apresentadas, propor um modelo único – *framework* conceitual - que possa ser utilizado como base para a construção de novas técnicas de indexação para BDTs. Esse modelo é apresentado neste capítulo, além das vantagens e desvantagens em utilizá-lo.

O *framework* conceitual proposto nesse trabalho é baseado nas regras e formalismos de orientação a objetos e representado na notação do diagrama de classes da Métodologia UML. O objetivo de um *framework* conceitual é o de fornecer um diagrama de classes que pode ser usado como base para a modelagem das classes do domínio da aplicação (índices para BDTs). Um *framework* conceitual [JOH92, JOH97, GAM94] não implica necessariamente em um produto acabado e executável, mas sim em um esquema conceitual de dados que, posteriormente, deverá ser traduzido para um esquema de dados específico para índices temporais.

O *framework* captura as características mais relevantes das técnicas vistas e pode ser utilizado como suporte a novas técnicas temporais de indexação. Características da semântica temporal apresentadas no capítulo dois, como por exemplo, intervalo temporal, ponto no tempo, tempo de validade e tempo de transação. Também estão presentes no modelo para direcionar as novas especificações de índices, bem como governar algumas características de alguns métodos do modelo.

A unificação dos modelos no *framework* proposto pode dar a idéia de que as soluções são isoladas para cada tipo de dado. Pode-se ter grandes vantagens com essa unificação, tais como padronização e reutilização de código, e principalmente a possibilidade de variação na construção dos índices específicos para dados temporais.

Quanto maior for o grau de granularidade hierárquica do *framework*, maior será a possibilidade de reuso das classes, dando maiores características de uma real estrutura de apoio para a construção de índices para dados temporais. Entretanto, isto acarreta um aumento na complexidade.

6.1 Características das técnicas de indexação temporal

Com o objetivo de incluir no modelo proposto uma gama relevante de características necessárias para indexar dados temporais, foram identificadas as principais características das técnicas de indexação apresentadas no capítulo cinco. Estas estão resumidas na tabela 6.1.

Estas características são a base do modelo unificado que será apresentado a seguir e estão contempladas na sua totalidade, representadas pelas modelagens individuais das técnicas estudas.

TABELA 6.1 – Características das Técnicas Específicas para Dados Temporais

Time Index	• Indexa tempo de validade • Utiliza B^+-Tree • Utiliza a estrutura auxiliar *no*w • Cada tempo indexado por uma B^+-Tree indica outro índice associado para todas tuplas válidas para o primeiro tempo • Pode agregar índices regulares
NB^+-Tree	• Indexa tempo de validade • Utiliza B^+-Tree • Utiliza um marcador final para tempo de validade • Difere busca pela versão corrente da busca de dados históricos
NR-Tree	• Indexa tempo de validade • Utiliza a R-Tree • Utiliza um marcador final para tempo de validade • Controla regiões temporais como regiões espaciais mas sem sobreposição
B+-Tree para Dados Temporais	• Indexa tempo de validade • Utiliza a B^+-Tree • Utiliza a variável *now* • Mapeia dados temporais em dados lineares • Escolha do tipo de ordenação, segundo as características do sistema
GR-Tree	• Indexa tempo de validade e tempo de transação • Utiliza a R*-Tree como base • Utiliza dois marcadores para determinar o tipo de algoritmo para a pesquisa. Um é relativo ao tipo de crescimento e o outro determina se o crescimento ultrapassou o tamanho da janela para pesquisa. • Utiliza estrutura auxiliar para determinar os finais dos tempos de transação e de validade, *UC* e *now*, respectivamente • Avalia crescimento da região temporal • Melhorar índice significa diminuir a região de pesquisa para não procurar em regiões inoperantes
M-ITVT	• Indexa tempos de validade e tempo de transação • Utiliza a B^+-Tree como base nas duas estruturas • Utiliza um marcador para indicar o TV corrente • Uma árvore para TT e outra para TV • Algoritmo de busca verifica primeiro a versão corrente para depois verificar o resto da estrutura

6.2 *Framework* conceitual para indexação temporal

O objetivo principal do trabalho é viabilizar um modelo capaz de auxiliar quando da criação de novas técnicas de indexação para dados temporais. Sendo assim, características como portabilidade, capacidade de agregação de novos conceitos e funcionalidades, clareza, visão global, entre outras, foram contempladas da melhor forma

possível.

Uma característica do *framework* conceitual proposto é que são apresentados apenas os métodos mais relevantes de cada classe. Métodos como inicializações, manipulações de persistência, entre outros, não são apresentados por não serem relevantes ao "estudo das técnicas de indexação para dados temporais". Além disso, para diminuir a apresentação do modelo, tornando-o menos complexo, são apresentados somente os nomes das classes e seus relacionamentos.

O modelo é apresentado na figura 6.4 e possui basicamente dois tipos de relacionamentos: (i) associações e (ii) especializações/generalizações. Quando foi necessário consistir que a classe que formará a estrutura de dados das árvores possua relacionamentos com seus nodos foram utilizadas associações. O relacionamento de especialização e generalização foi utilizado para os relacionamentos que necessitam herdar características (atributos e métodos) de uma subclasse da superclasse.

A estrutura do nodo pode ser vista como um vetor do tipo de composição de cada técnica de indexação. Na figura 6.1 os elementos que compõe as informações básicas dos nodos são P (ponteiro para o próximo nodo da árvore), X (chave de busca) e & (informações do endereço físico). Esse conjunto de três elementos foi denominado de **ComposiçãoNodo**. O nodo propriamente dito é visto como um vetor desses elementos. Na concepção dos índices, o tamanho do vetor é igual ao tamanho máximo de entradas por nodo. O número mínimo de entradas é definido pela implementação de cada técnica.

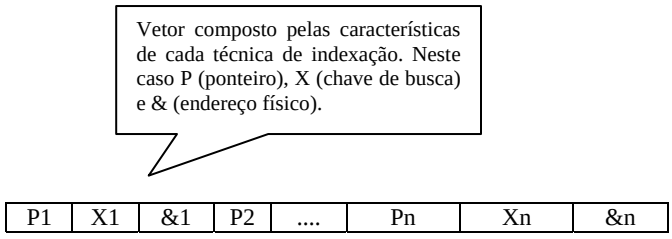


FIGURA 6.1 – Representação dos Nodos por Vetor

Um exemplo prático dessa representação no *framework* conceitual (figura 6.4) pode ser apresentado na classe **NodoB+-Tree** e **B+-Tree**, onde o relacionamento de associação garante que o vetor que corresponde à árvore seja do tipo **NodoB+-Tree**. Sendo assim, é assegurado que o vetor terá nodos do tipo especificado na classe **NodoB+-Tree**.

A segunda característica do modelo se refere às especializações e generalizações. Por exemplo, entre as classes **ComposiçãoNodo** e **ComposiçãoNodoB** foi utilizada a especialização/generalização porque existem atributos os quais, semanticamente, pertencem à classe **ComposiçãoNodo**, uma vez que outras classes fazem uso da mesma semântica. Os atributos específicos da classe **ComposiçãoNodoB** são representados na própria classe. Assim, as características (atributos e métodos) da classe **ComposiçãoNodoB** são formadas pelas características da **ComposiçãoNodo** que são herdadas através da especialização, mais as características específicas da **ComposiçãoNodoB**, conforme exemplo da figura 6.2.

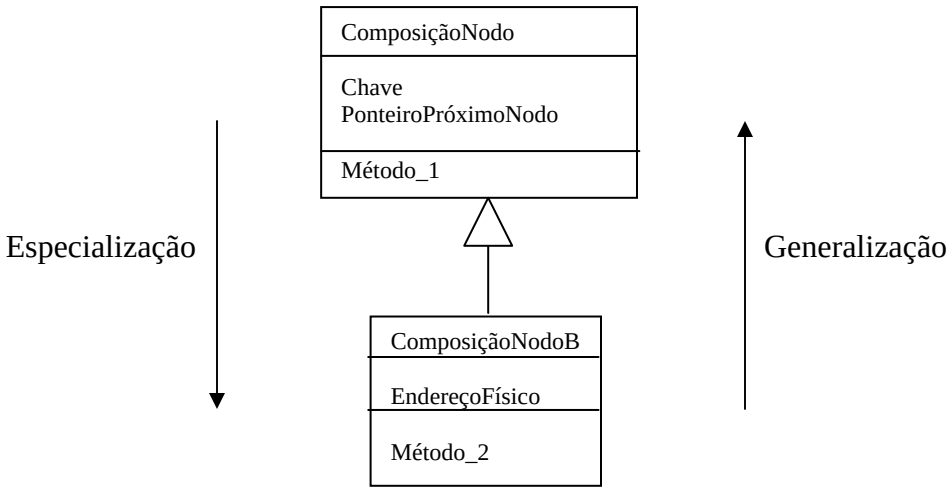


FIGURA 6.2 – Exemplo de Especialização e Generalização

Na figura 6.2, a classe **ComposiçãoNodoB** possui os atributos **Chave**, **PonteiroPróximoNodo** e **EndereçoFísico**, e pode executar os métodos **Método_1** e **Método_2**.

A composição de árvore é formada por uma estrutura de vetor do tipo de nodo, própria de cada técnica. Por exemplo, na técnica B-Tree a composição do nodo é formada pelos elementos necessário a essas técnica (ponteiro, chave de busca e informações do endereço físico). O nodo é formado por um vetor do tipo de composição do nodo (figura 6.1) e a árvore por sua vez, é formada por um vetor do tipo de nodo, conforme figura 6.3.

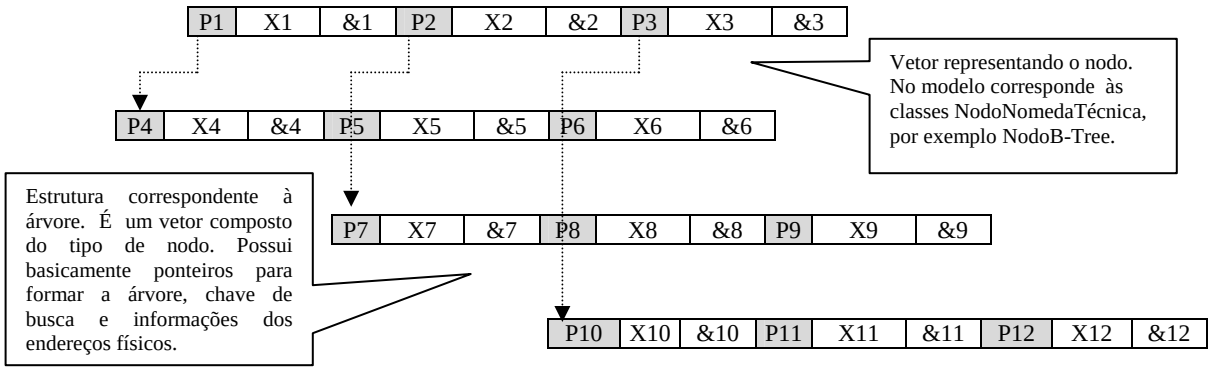


FIGURA 6.3 – Estrutura de Árvore no *Framework* Conceitual

A classe **TipoTempo** representa os tipos de tempo utilizados nos rótulos temporais, (i) intervalo de tempo ou (ii) ponto no tempo. É uma generalização das classes **ComposiçãoTV** e **ComposiçãoTT** porque nestas classes existe a necessidade de se diferenciar os atributos início e fim. Por exemplo, se for caracterizado intervalo de tempo, os atributos início e fim possuem um comportamento; se for caracterizado ponto no

tempo, os algoritmos devem tratar os atributos início e fim como o mesmo valor, identificando ponto no tempo. Algumas implementações de BDTs fazem uso desse artifício para representar ponto no tempo, como Hübler definiu em [HÜB2000, HÜB 2000a].

Tempo de validade e tempo de transação estão contemplados nas classes **ComposiçãoTV** e **ComposiçãoTT**, respectivamente. Possuem basicamente a semântica das características temporais dos BDTs, por exemplo: tipo de tempo (herdada da classe **TipoTempo**), tempo inicial e tempo final. Possuem métodos de consistência desses tempos em cada classe (**ComposiçãoTV** e **ComposiçãoTT**). De acordo com o tipo de tempo, os métodos de consistência podem assumir um determinado comportamento. Por exemplo, se for intervalo de tempo, existe a necessidade de consistir se o tempo final é maior ou igual ao tempo inicial; se for ponto no tempo, o valor final assume o mesmo que o valor inicial e não existe a necessidade de consistência.

Outra característica do modelo é que este possui uma estrutura auxiliar, definida na classe **EstruturaAuxiliar**, a qual é constituída pelos atributos utilizados em técnicas diferentes e que possuem a mesma semântica. Por exemplo, na técnica *Time Index* existe a variável *now*, que traduz o instante atual do banco de dados, utilizado para representar o tempo final do tempo de validade. Na técnica GR-Tree, dois atributos são utilizados para representar o mesmo tipo de informação, *UC* e *now*, o primeiro para tempo de transação e o segundo para tempo de validade. Com essa classe o *framework* todo pode usufruir da mesma informação e da mesma nomenclatura, já existindo aqui uma padronização das técnicas.

Segundo características da especificação de um índice, podem ser feitos diferentes caminhos de pesquisa nas árvores de indexação. De acordo com o método de Goh em [GOH96], existe uma possibilidade de variação na leitura dos nodos na árvore. Essa característica é contemplada na classe **Ordenação**, onde existem algoritmos diferenciados segundo um critério pré-estabelecido, tal como ordenação diagonal, horizontal e vertical. Estas especificações de consulta (tipo de ordenação) governam de maneira mais eficiente a busca de dados naquela estrutura.

Além da variabilidade de ordenação dentro da mesma técnica, uma idéia semelhante pode ser utilizada quando existem sistemas com características mais acentuadas. Por exemplo, é de conhecimento notório que a técnica B-Tree é mais eficiente na consulta do que a B^+-Tree devido à composição de sua estrutura. Em contrapartida, a B^+-Tree tem melhor desempenho que a B-Tree para operações de inserção e/ou alteração de dados. Com base nesta afirmação e analogamente à idéia de Goh em escolher o melhor algoritmo de busca dentro da mesma técnica, pode-ser fazer especializações de técnicas que possuem melhor desempenho para cada tipo de caso e segundo características da aplicação, usufruir de um ou outro elemento de manipulação de dado. Por exemplo, pode-se construir uma estrutura que possua uma hierarquia composta tanto de B^+-Tree como de B-Tree, conforme o exemplo da colaboração de B^+-Tree e B-Tree na técnica *Time Index* (figura 5.3). Assim, para aplicações que exijam um alto grau de desempenho nas consultas, deve-se utilizar os métodos da B-Tree, e para aquelas com grande incidência de inserções e/ou deve-se alterações, utilizar a B^+-Tree.

Uma característica presente no *framework* conceitual proposto é a possibilidade

de prover informações de desempenho, desde que fornecida a fórmula para o cálculo da mesma. Segundo Lü em [LÜ95], deve ser contemplada a possibilidade de prover informações de desempenho dos índices. Essa idéia foi implementada no método **InformaçãoDesempenho** na classe **Tree** (figura 4.14 e figura 6.4). Essa informação pode ser fornecida de duas maneiras: fazer a implementação no próprio método **InformaçãoDesempenho** da classe **Tree** através do tipo de árvore fornecer essa informação, ou por redefinição desse método em cada técnica, implementando a fórmula para fornecer essa informação.

Essa característica pode ser útil para determinar qual a melhor técnica a ser utilizada, segundo as características de uma aplicação. Um exemplo pode ser dado analisando a técnica B^+-Tree, onde o número de acessos pode ser calculado de acordo com algumas características da estrutura da árvore, como ordem e número de entradas por nodo. A fórmula que provê essa informação nas B-Trees é dada por $Log_d N$. Sendo assim, pode-se, antecipadamente, simular o desempenho e tomar as melhores medidas.

Para melhorar a visualização do modelo na figura 6.4, os atributos e métodos foram omitidos, existindo uma legenda que indica a qual técnica pertence cada classe. Assim, para um melhor detalhamento, pode-se identificar estas informações nos diagramas de classes específicos de cada técnica apresentados no capítulo cinco.

6.3 Análise do *framework* conceitual

As técnicas que serviram de base para a construção do *framework* conceitual podem ser classificadas em dois tipos: (i) as técnicas tradicionais como B-Tree, B^+-Tree e R-Tree, e (ii) as técnicas temporais de indexação como *TimeIndex*, NB^+-Tree, NR-Tree, GR-Tree, B^+-Tree para BDs e M-ITTV. Em particular, a R-Tree possui uma técnica evolutiva denominada R*-Tree ou RStar-Tree, que também serve de base para algumas estruturas de indexação temporal. Entretanto, por apresentar grande semelhança com a R-Tree, diferenciando-se desta apenas em um dos algoritmos (*SplitMode*), não foi apresentada neste trabalho.

As técnicas B-Tree, B^+-Tree e R-Tree são de fundamental importância no estudo das técnicas de indexação temporal, por isso sua funcionalidade é detalhada no *framework*. A técnica B^+-Tree, embora seja muito parecida com a B-Tree, é um aperfeiçoamento desta última, porém como afirmado em [HOR84], nenhuma técnica consegue ser melhor em todas operações de manipulação de dados. Sendo assim, a B^+-Tree e a B-Tree possuem características distintas. A técnica que melhor se adapta para aplicações onde a incidência de consultas é maior que a de inserções e/ou alterações é a B-Tree, por apresentar características na sua construção que possibilita encontrar o dados na árvore antes de processar os nodos-folhas. Em contrapartida, em aplicações que exijam um melhor desempenho na entrada de dados e/ou alterações é recomendável o uso da técnica B^+-Tree. Embora nenhuma das técnicas apresentadas se baseie diretamente na B-Tree, esta foi incluída no modelo para permitir que uma nova técnica, definida a partir do modelo e que tenha maior incidência de operações de consultas do que inserções e/ou alterações, possa se basear nela.

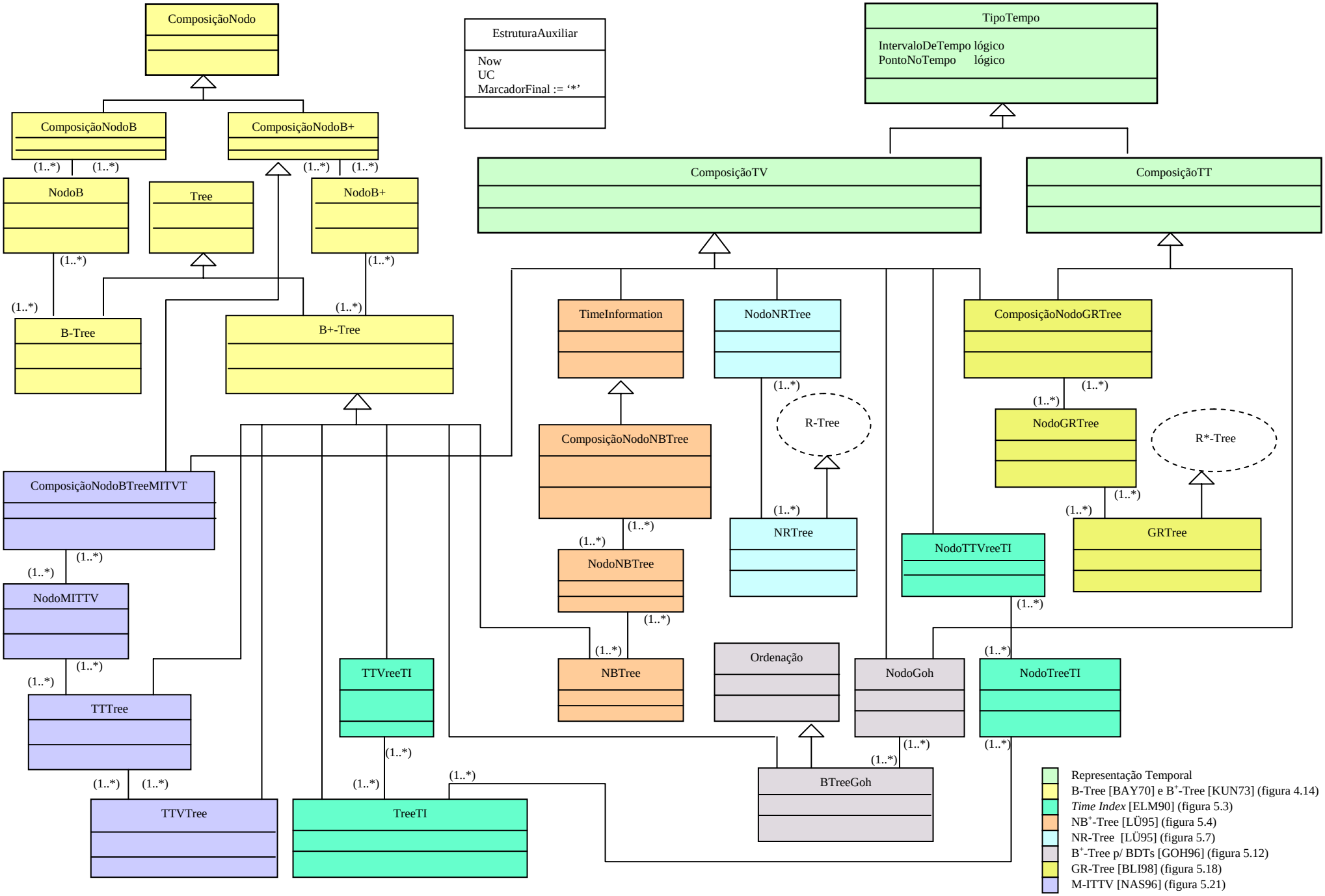


FIGURA 6.4 – Proposta de um *Framework* Conceitual para Indexação Temporal

Outra característica do modelo que deve ser ressaltada se refere a situações em que se deseja indexar dados temporais com tempo de validade e tempo de transação ao mesmo tempo. Em geral, as técnicas estudadas utilizam uma primeira árvore para indexar alguns dos tempos e uma segunda, encadeada, para indexar a parte da semântica temporal relativa ao outro tempo em questão. Por exemplo, uma primeira árvore para tempo de validade e uma outra para tempo de transação, ou vice-versa.

O *framework* conceitual proposto neste trabalho deixa inteiramente livre a possibilidade de variação para a construção das árvores. Por exemplo, pode ser utilizada uma B^+-Tree com identificador da tupla corrente (tipo estrutura auxiliar) para processar os tempos de validade em uma primeira instância, e uma segunda árvore com características espaciais (R-Tree ou R*-Tree) para representar os tempos de transação. Essa composição foi apenas um exemplo de quão pode ser variável a construção do índice.

Um *Framework* conceitual não implica necessariamente em um produto acabado e executável. O seu objetivo é de fornecer um diagrama de classes que pode ser usado como base para a modelagem das classes do domínio da aplicação (índices para BDTs). Assim, o modelo funciona basicamente como base a criação de índices temporais. As especificações de novas estruturas podem ser especializações do modelo apresentado, bem como redefinições de atributos e métodos.

6.4 Considerações finais

Neste capítulo foi apresentado um modelo caracterizado como um *framework* conceitual, que tem o objetivo de dar suporte à criação de técnicas de indexação para dados temporais.

Uma das principais vantagens é a manutenção simplificada das técnicas expressadas neste modelo. Isto porque, com a orientação a objetos, as operações e a estrutura de dados estão descritas em um mesmo nível conceitual. Com isto, é garantida maior compreensão, legibilidade e visualização dos atributos e seus métodos, embora na figura 6.4, os atributos e métodos não estejam representados.

Outra vantagem do modelo, e uma das principais, é o princípio da reusabilidade de código. Com o mecanismo de especialização e generalização, é possível criar outras estruturas de classes cada vez mais especializadas, mais complexas, apenas adicionando o comportamento especializado e aproveitando as características das superclasses.

A portabilidade constitui uma característica importante do *framework* conceitual. Uma vez que foi utilizada a Metodologia UML, isto isenta qualquer relacionamento direto com algum tipo de linguagem de programação, possibilitando uma maior portabilidade do modelo. Desta forma, projetistas podem utilizar o modelo para diferentes plataformas e podem agregar novas funcionalidades que possam ser úteis para outros índices. Por exemplo, uma das técnicas de indexação [GOH96] utiliza uma função de mapeamento de dados multidimensionais para dados lineares, denominada de IST. Um projetista pode desenvolver outra forma de mapeamento e agregá-la ao modelo.

Embora existam inúmeras vantagens para utilizar um conjunto de conceitos agregados em um modelo único definido por um *framework* conceitual, existe alguma desvantagem. A maior desvantagem do modelo consiste na complexidade do seu

entendimento por parte do projetista, uma vez que apresenta conceitos variados em uma mesma estrutura, o que pode gerar dificuldade quanto à abstração do modelo.

É importante lembrar que esta proposta é o primeiro passo para se construir uma estrutura sólida para criação de índices para dados temporais. Sendo assim, além da complexidade natural do entendimento da estrutura, pode acontecer que alguns conceitos sejam ainda insipientes e seu significado esteja em parte distorcido.

Como este trabalho é um estudo inicial no tema de *framework* conceitual para indexação temporal, algumas características ainda podem e devem ser melhoradas. Porém, isso só será feito corretamente e com precisão no momento em que forem projetados índices utilizando essa estrutura. Com o tempo, necessidades de alterações e um melhor refinamento vão surgir, aperfeiçoando o *framework* conceitual.

7 Estudo de caso

Neste capítulo serão abordados os conceitos básicos do TF-ORM [EDE94], bem como uma especificação de índices para contemplar seus conceitos utilizando o *framework* proposto. É preciso fazer uma análise bem criteriosa das características dos dados temporais a serem amparados por alguma arquitetura de índices, uma vez que estas características podem e provavelmente devem determinar a melhor forma de estrutura.

No caso do TF-ORM, serão explanados os principais conceitos temporais e algumas características relevantes para a especificação proposta. Para uma determinação mais precisa de qual estrutura de indexação se adapta melhor para o TF-ORM seria preciso algo a mais do que é demonstrado, principalmente no comportamento funcional das operações sobre este modelo. O intuito deste estudo de caso é tão somente demonstrar a viabilidade de criação de estruturas para uma determinada situação. Não é comprovada a eficiência do índice, uma vez que isto só seria possível quando fosse implementado e testes exaustivos fossem feitos. Para uma demonstração de como pode ser útil e o quanto se torna simples projetar uma estrutura de índices utilizando o *framework* conceitual, as características explanadas são o suficiente.

Uma vez feita, criteriosamente, a análise de requisitos e o estudo das características relevantes para a definição de uma estrutura, pode-se então especializar e agregar funcionalidades junto ao modelo hierárquico apresentado no capítulo seis, para a definição de um índice.

7.1 TF-ORM – conceitos básicos

O TF-ORM (*Temporal Functionality in Objects With Roles Model*) [EDE 93, 94] é um modelo de dados orientado a objetos que utiliza o conceito de papéis para representar os diferentes comportamentos de um objeto. O modelo permite a modelagem dos aspectos estáticos e dinâmicos da aplicação, pois considera todos os estados do objeto, associando informações temporais às propriedades que podem mudar de valor ao longo do tempo. É um modelo bitemporal, ou seja, suporta tanto tempo de transação quanto tempo de validade.

O conceito de papéis tem por objetivo separar a representação dos aspectos dinâmicos de um objeto, dos seus aspectos estáticos. O objeto poderá assumir ao longo de sua existência um ou mais papéis, em diferentes tempos, continuando a ser uma instância de uma única classe. O objeto, com o conceito de papéis, pode apresentar, simultaneamente, diversas instâncias de um mesmo papel, o que possibilita a representação de variações do seu comportamento.

Muitas vezes o conceito de papéis é confundido com o de subclasses, cuja diferença fundamental está na identidade dos objetos [EDE 94]. Quando representações através de papéis são utilizadas, o objeto existe como instância da classe independentemente do fato de estar ou não desempenhando o papel. Considere o exemplo: se um determinado objeto empregado fosse representado como uma subclasse de pessoa,

este objeto apenas poderia ser criado no momento em que esta pessoa tivesse um emprego. Utilizando a representação de empregado através de um papel, o objeto empregado existe independentemente desta pessoa possuir um emprego.

7.1.1 Definição de classes e de papéis

Uma classe é definida por um nome único em toda a especificação de classes e por um conjunto de papéis. Cada papel representa um comportamento diferente do objeto.

Cada papel de uma classe consiste de um nome (Np_i), um conjunto de propriedades (Pr_i) do papel, um conjunto de estados abstratos S_i que o objeto pode apresentar neste papel, um conjunto de mensagens M_i que o objeto pode receber e enviar, e de um conjunto de regras R_i (regras de transição de estado e regras de integridade):

$$P_i = < Np_i, Pr_i, S_i, M_i, R_i >$$

As descrições abstratas das características de um objeto são chamadas de *propriedades*. Existem dois tipos de propriedades que podem ser identificadas em aplicações que evoluem com o passar do tempo:

- **propriedades estáticas**: são aquelas que nunca mudam de valor. Somente um valor pode ser definido para uma propriedade estática, permanecendo o mesmo durante toda a existência do objeto; e

- **propriedades dinâmicas**: podem representar diversos valores durante a existência do objeto. Todos os valores definidos para uma propriedade dinâmica devem ficar armazenados permanentemente, pressupondo a implementação de um banco de dados temporal.

7.1.2 Identificador de instâncias

Diversas instâncias de uma mesma classe podem estar presentes simultaneamente ao se tratar de um modelo orientado a objetos. A manipulação das diferentes instâncias, tanto de classes quanto de papéis, requer uma identificação de instâncias.

Um identificador interno, único no sistema, é associado ao objeto, pelo sistema gerenciador do banco de dados, quando da criação do mesmo. Cada instância de um papel deste objeto será associada a um identificador, que também apresentará valores únicos no sistema.

Foram definidas duas propriedades especiais para representar este identificador único:

- *old*: presente no papel básico de todas as classes, armazena o valor do identificador de cada objeto da classe; e

- *rld*: presente em todos os papéis armazena os identificadores de cada papel

instanciado.

7.1.3 Representação temporal e elemento temporal primitivo

No modelo de dados TF-ORM é considerado que o tempo está linearmente ordenado, variando de forma discreta.

A variação de valores armazenados em um banco de dados que implemente este modelo é considerada na forma de escada - uma vez definido um valor, este permanece válido até que outro valor seja definido.

O elemento temporal primitivo adotado para a representação de aspectos temporais em TF-ORM foi o ponto no tempo. Todos os pontos pertencem a uma ordenação total. Intervalos serão representados através de um par de pontos no tempo, os quais definem seus limites. Em implementações como as definidas em [HÜB2000] e [EDE2000] foram utilizados quatro pontos no tempo, representando intervalo de TV e TT para facilitar as consultas.

Uma análise do domínio de sistemas de informação identificou o minuto como a menor granularidade temporal adequada para atividades humanas. Este foi, portanto, escolhido como *chronon* do modelo de dados - menor intervalo de tempo entre quaisquer dois pontos no tempo. A definição completa de um ponto no tempo é dada, portanto, pela definição de uma data (ano, mês e dia) e por um horário dentro desta data (hora e minuto). É um modelo bitemporal, acrescentando tempo de transação e tempo de validade às propriedades dinâmicas.

7.2 Proposta do índice

As características do TF-ORM devem ser bem salientadas para que se possa projetar a melhor técnica de indexação para este modelo. Estas características serão enfocadas segundos os conceitos básicos acima descritos. Uma vez analisado o TF-ORM bem como suas principais características, pode-se então esboçar uma técnica que possa contemplar o seu propósito.

O modelo TF-ORM foi criado para modelar atividades de escritório, então, pela sua definição a granularidade adotada foi o minuto, uma vez que não foi encontrada uma atividade que exigisse menor granularidade relevante. Essa informação pode ser considerada importante porque reflete o tipo de dado temporal que será armazenado e indexado.

A representação temporal no modelo TF-ORM é feita através de rótulos temporais para as propriedades dinâmicas. A cada propriedade dinâmica, é acrescentada uma informação bitemporal. Desta forma, a especificação de um índice deve contemplar o acesso a essas propriedades analisando os rótulos temporais associados às mesmas. As propriedades estáticas, que funcionam como dados lineares, podem ser indexadas com técnicas bases (B-Tree e B^+-Tree) e não constituem um problema para a especificação do índice para o TF-ORM.

Enfocando melhor o problema, deve ser especificada uma estrutura de índices para contemplar TV e TT para as propriedades dinâmicas com elemento temporal intervalo no

tempo para ficar de acordo com as implementações feitas.

Estudando somente os tempos envolvidos, pode ser analisado um comportamento em que foi descartada a hipótese de utilizar a técnica GR-Tree [BLI98] e seus algoritmos pela forma como o tempo de validade se comporta. Essa técnica de indexação possui grande eficiência para dados temporais, porém tem como principal requisito que o comportamento da semântica temporal seja do tipo *now-relative* [BLI98], ou seja, os tempos devem sempre crescer continuamente, não importando a escala, desde que aumentem linearmente. Pela definição do modelo TF-ORM não existe uma garantia de que isso ocorra, uma vez que o tempo de validade pode ser definido como "válido" do futuro para o passado. Isto descarta o conceito de dados *now-relatives*, e por conseqüência a técnica de Bliujüte.

7.3 Especificação

Uma primeira idéia de especificação de um índice é a seguinte:

- duas árvores serão utilizadas, a primeira para processar os tempos de validades e a segunda para representar os tempos de transação;
- a primeira será utilizada sob o prisma da B-Tree, como técnica base;
- para ter melhor desempenho nas consultas que envolvem a versão corrente, será utilizado o artifício de um marcador de versão corrente, representado pelo caractere '*';
- a segunda árvore também será baseada na B-Tree; e
- o relacionamento entre **TFORMIndexTV** e **TFORMIndexTT** serve para garantir que cada TT está relacionado com um TV existente.

A utilização do *framework* proposto está ilustrada na figura 7.1. A semântica temporal do modelo está representada pela especialização das classes **TipoTempo**, referente a intervalo de tempo; **ComposiçãoTV**, referente aos tempos de validades; e **ComposiçãoTT**, referente aos tempos de transação.

Algumas características desse modelo, tais como marcador de versão corrente e o instante atual representado pelo atributo *now*, estão contidas em uma classe única denominada de **EstruturaAuxiliar**. Tanto as classes de representação temporal, **TipoTempo**, **ComposiçãoTV** e **ComposiçãoTT**, bem como a classe **EstrturaAuxiliar,** são provenientes do *framework* conceitual.

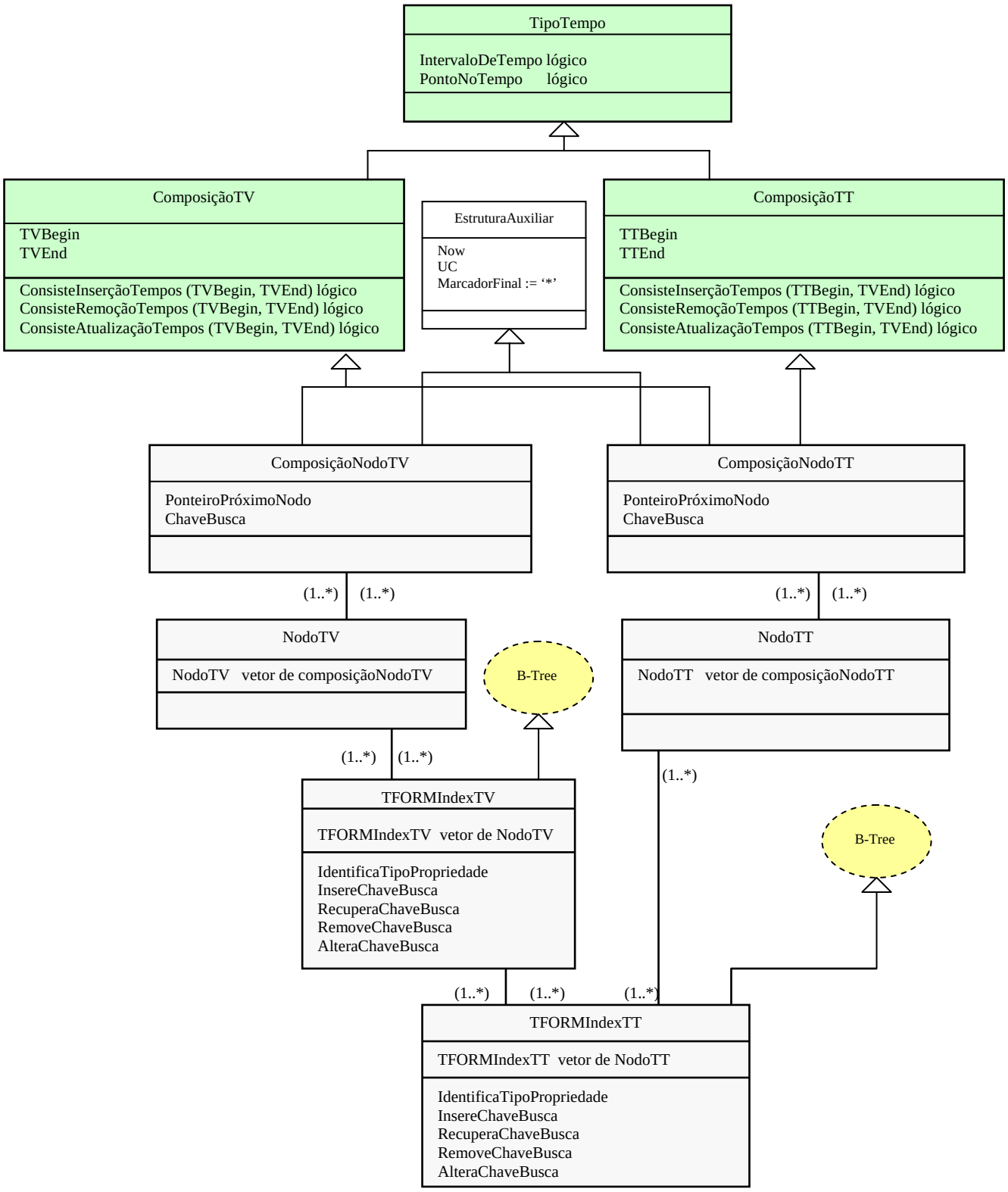


FIGURA 7.1 – Especificação do TFORMIndex

As classes de composição dos nodos referentes ao índice do TF-ORM estão representadas em **ComposiçãoNodoTV** e **ComposiçãoNodoTT**. São especializações das classes de representação temporal, logo possuem os atributos de tipo de tempo, tempo inicial e final de validade e transação, respectivamente. Os métodos referentes à manipulação destes atributos também estão disponíveis. São acrescentadas a esses atributos, as características para compor a estrutura de árvore, como ponteiro para o próximo nodo e a chave de busca. Assim fica contemplada totalmente a composição do nodo para formar os vetores que representarão a árvores de índices, contendo (i) tempo inicial, (ii) tempo final, (iii) ponteiro para o próximo nodo e a (iv) chave de busca. No caso da classe **ComposiçãoNodoTT**, existe uma especialização da classe **ComposiçãoNodoTV** para que os atributos de tempo de validade estejam disponíveis na composição dos tempos de transação, sendo assim feita a ligação das árvores, conforme figura 7.2.

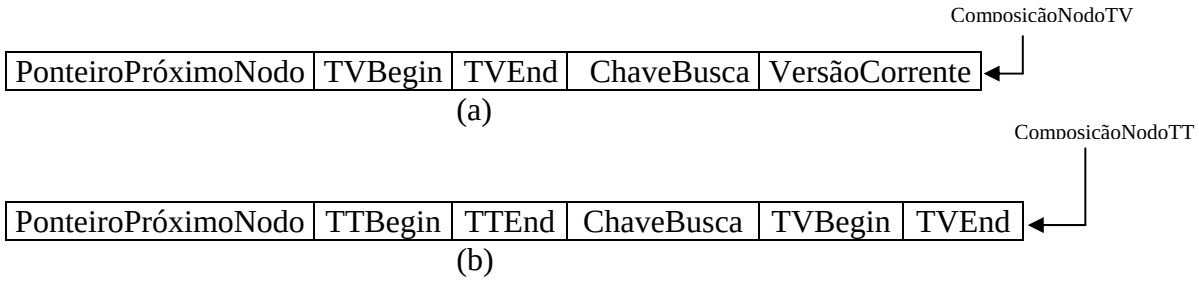


FIGURA 7.2 – Composição dos Nodos do TFORMIndex

O funcionamento do índice pode ser exemplificado na tabela 7.1. É importante lembrar que para as propriedades estáticas não existe a relação dos rótulos temporais, podendo assim, ser indexadas por técnicas convencionais. É o caso do nome do funcionário (tabela 7.1). Entretanto, as propriedades dinâmicas necessitam dos rótulos temporais (TV e TT) para dar sentido temporal ao modelo TF-ORM, como é o caso do salário (tabela 7.1).

As classes que suportam o conceito de nodo são denominadas **NodoTV** e **NodoTT**, respectivamente para tempos de validades e tempos de transação. Sua funcionalidade é compor os vetores referentes aos nodos da árvore. Por isso são do tipo da estrutura que comporta as características dos nodos, conforme figura 7.3.

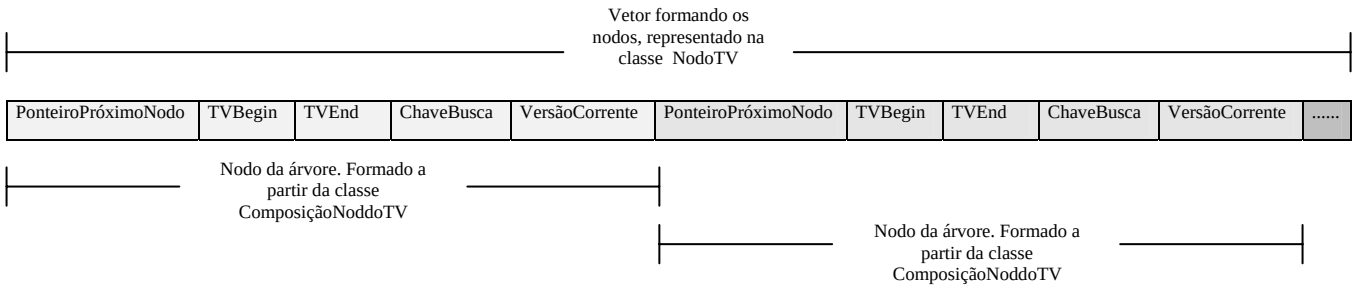


FIGURA 7.3 – Estrutura de Nodos do TFORMIndex

A estrutura da árvore é formada a partir de um vetor do tipo vetor de nodos,

representado na figura 7.3. Essa construção é representada na classe **TFORMIndexTV**, no caso dos tempos de validades. Essa estrutura de dados que representa a árvore é indexada pela técnica B-Tree. Entretanto, para surtir melhor efeito, alguns algoritmos devem ser redefinidos, principalmente quando tratada a versão corrente. Por esta razão estes métodos são definidos na classe **TFORMIndexTV**. Nesta classe existe um método denominado de **IdentificaTipoPropriedade** responsável pela identificação do tipo de propriedade, classificando-a como estática ou dinâmica. Segundo essa classificação, o método vai ativar a técnica de indexação para propriedades dinâmicas, que possui os rótulos temporais, ou invocar a indexação para dados lineares, diretamente pela B-Tree, uma vez que propriedades estáticas não possuem rótulos temporais.

Analogamente à estrutura dos tempos de validades, os tempos de transação também são declarados como uma estrutura de vetor composto por outro vetor. O primeiro representa a árvore e o segundo representa os nodos. Essa representação é atendida na classe **TFORMIndexTT** e **NodoTT**, respectivamente. A principal diferença entre a estrutura dos TVs e TTs é que a segunda possui dois atributos a mais na composição dos seus nodos. Esses atributos se referem aos tempos de validade. Assim, é feita a ligação entre as duas árvores. Após o processamento dos tempos de validade, é processada a árvore dos tempos de transações, mas somente para os tempos de validade encontrados na primeira pesquisa. Para surtir melhor efeito, principalmente por conter uma chave de busca importante, o TV, os métodos da classe **TFORMIndexTT** também são redefinidos, assim como na classe **TFORMIndexTV**.

Um exemplo dessa especificação de índices para o TF-ORM pode ser analisada na tabela 7.1.

TABELA 7.1 – Tabela de Funcionários e Salários para Exemplificar o TFORMIndex.

Tupla	Nome do Funcionário	Salário (R$)	TV (Dia)	TT (Dia)
T1	Fábio	1.100,00	[1,3]	[1,1]
T2	Evelise	1.500,00	[10,now]	[2,2]
T3	Brenner	200,00	[11,11]	[11,11]
T4	Simone	1.800,00	[11,15]	[11,11]
T5	Valéria	500,00	[7,9]	[2,2]
T6	Jeferson	1.250,00	[20,now]	[20,20]
T7	Evandro	500,00	[16,now]	[22,22]

No exemplo da tabela 7.1, somente o atributo salário possui rótulo temporal, por se tratar de uma propriedade dinâmica. O atributo nome do funcionário é uma propriedade estática, logo não possui TV nem TT, podendo ser indexado diretamente por técnicas convencionais como a B-Tree ou a B^+-Tree. A tabela 7.1 pode ser estruturada de acordo com a figura 7.4.

Uma pesquisa temporal do tipo “Selecione os funcionários e seus salários que tiveram este salário acima de R$1.000,00 desde o dia 11” (simplificando nesse exemplo a representação da data), pode ser traduzida na linguagem de consulta definida para o TF-ORM conforme tabela 7.2.

TABELA 7.2 – Consulta do TF-ORM

Select nome_funcionario and salario From funcionarios When (salario > 1000 or sometimes past salario > 1000)   and period in [11,now]

Ao processar este comando a estrutura de índices deve procurar as tuplas com os tempos de validades associados ao salário (propriedade dinâmica). Assim, o processamento segue conforme figura 7.4.

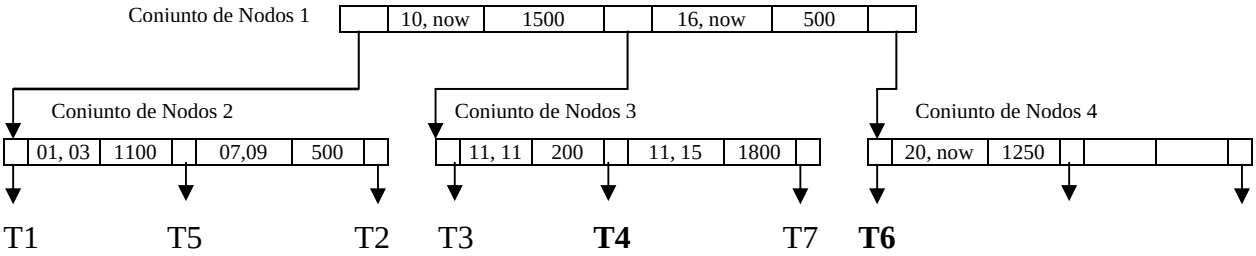


FIGURA 7.4 – Árvore TFORMIndex referente à Tabela 7.1.

O processamento desta consulta começa na pesquisa dos TV´s maiores ou iguais a 11. Depois de encontrados os nodos, é verificado se o salário correspondente àquele TV´s possui o salário maior que R$1000,00 (condições da consulta). O primeiro passo deve ser o processamento do primeiro conjunto de nodos. O primeiro ponteiro indica os tempos de validade menores ou iguais a 10, o que não satisfaz os requisitos da consulta, logo o conjunto de nodos 2 não é processado. O ponteiro central do primeiro conjunto de nodos aponta para a entrada de nodos cujos TV´s são maiores que 10 e menores ou igual a 16. Neste caso o algoritmo processará o conjunto número 3, uma vez que a consulta requer TV´s maiores que 11. Assim, o conjunto de nodos 3 será pesquisado retornando as tuplas com salários maiores que R$1000,00, resultando a tupla T4. O processamento continua no conjunto de nodos número 1. O último ponteiro dessa entrada leva para os nodos com TV´s maiores que 16, logo o conjunto número 4 também será processado, retornando a tupla T6 por ter o salário maior do que R$1000,00. Assim é satisfeita por inteiro a consulta original da tabela 7.2, tendo como resultado total as tuplas T4 e T6. Seria prudente lembrar que a eficiência ou não do índice só pode ser determinada quando testes exaustivos forem feitos e o resultado final for analisado.

7.4 Considerações finais

A conclusão mais importante deste capítulo não é a especificação em si, e sim a grande variedade de conceitos e possibilidades de estruturas que estão disponíveis no *framework* conceitual. Talvez esse ponto seja o mais importante agregado ao estudo.

As especificações denotam algumas particularidades. Por exemplo, em geral quando dois tempos são tratados, serão utilizadas duas árvores de indexação, uma para cada tipo de tempo. No caso dos tempos de validades, a analogia dos dados espaciais se torna mais visível, por se tratar do intervalo mais naturalmente representado, tendo um *início* e um *fim*, enquanto o tempo de transação possui uma postura mais linear.

Para BDTs em que são tratados pontos no tempo, é possível indexá-los por intervalo de tempos, visto que existem implementações que utilizaram este artifício para facilitar as consultas. Assim, é necessário que exista uma técnica de indexação para contemplar este tipo de BDT.

Enfim, a idéia de utilizar uma estrutura pronta e pré-definida em alguns conceitos pode tornar a arquitetura de índices com grande possibilidade de variação, trazendo conseqüentemente, grandes benefícios para este ramo de estudo.

8 Conclusão

Neste trabalho foi apresentado um estudo de algumas técnicas de indexação para dados, dando ênfase àquelas que têm por objetivo indexar dados temporais. Um dos pontos relevantes desse estudo foi a identificação da analogia existente entre dados temporais e dados espaciais, o que permite a utilização de técnicas como R-Tree e R*-Tree, definidas para dados espaciais, como base para a definição de novas técnicas para dados temporais. Outro aspecto importante é a possibilidade de fazer um mapeamento de dados multidimensionais (por exemplo, bitemporais) para dados lineares, o que permite a utilização de técnicas consagradas como B-Tree e B^+-Tree também para este tipo de dados, como exemplificado na técnica de Goh [GOH96].

Com base nas características das técnicas apresentadas foi proposto um *framework* conceitual com o intuito de prover um apoio para a criação de novas técnicas de indexação para dados temporais.

Para representar o *framework* conceitual foi utilizado o padrão UML. Foi escolhida essa notação por apresentar um conjunto de características que se encaixam perfeitamente nesse tipo de estudo. Existem algumas vantagens em se modelar uma estrutura apoiada na orientação a objetos. Por exemplo: (i) esta abordagem incentiva os desenvolvedores a trabalharem e pensarem em termos do domínio da aplicação durante a maior parte do desenvolvimento, proporcionando dedicação maior à fase de análise; (ii) no desenvolvimento baseado em objetos há uma redução no tempo de manutenção, pois as revisões são mais fáceis e mais rápidas já que o problema é melhor localizado; (iii) favorece a reutilização, já que ocorre a construção de componentes mais gerais, estáveis e independentes; (iv) facilidade de extensão, visto que objetos têm sua interface bem definida.

O *framework* foi construído basicamente sobre um diagrama de classes, que agrega classes, especializações/generalizações e associações representando o relacionamento semântico entre estas classes.

O *framework* proposto possui algumas vantagens, tais como agregar novos conceitos sem ferir as características originais e particulares das técnicas estudadas, modularidade, reutilização, portabilidade, entre outras vantagens. Por outro lado, apresenta também uma desvantagem, que é a complexidade de entendimento do domínio do problema: para buscar no *framework* conceitual as melhores características do modelo é preciso um estudo e um entendimento profundo do trabalho.

A grande contribuição desse trabalho se encontra em unificar conceitos, características, detalhes de uma ou outra técnica em um modelo único, que possa servir de base para a criação de outras técnicas de indexação para dados temporais, adequadas a modelos de dados temporais específicos.

Fica de incentivo para trabalhos futuros a continuação do estudo e a possível realização de algumas ferramentas, por exemplo. Como esse trabalho constitui uma primeira versão de proposta de *framework* conceitual, o principal trabalho para o futuro seria aprofundar e agregar mais funcionalidade a esse modelo. Existem outras propostas de continuação por exemplo:

- agregar novas funcionalidade ao modelo para dar maior variabilidade na criação de novas técnicas de indexação;
- agregar a implementação de alguns métodos ou de técnicas completas como, a B-Tree e B+-Tree;
- desenvolver técnicas de indexação para dados temporais com efetiva implementação e exaustivos testes para validar o índice construído. Da mesma forma, isto validaria o *framework* conceitual; e
- desenvolvimento de uma ferramenta para auxiliar na especificação de técnicas de indexação para dados temporais. Essa ferramenta permitiria uma entrada de parâmetros equivalente às características dos dados a serem contemplados. O resultado seria uma pré-especificação da nova técnica de indexação temporal.

Bibliografia

[BAY 70] BAYER, R.; McCREIGHT, E. Organization and maintence of large ordered indexes, In: WORKSHOP ON DATA DESCRIPTION AND ACCESS, 1970, Houston, Texas. **Proceddings…** Houston: ACM-SIGFIEDT, 1970.

[BEC 90] BACKMANN, N. The R*-Tree: An Efficient and Robust Acess Method for Points and Rectangules. **ACM Sigmod**, Atlantic City, v.19, p. 322-331, 1990.

[BLI 98] BLIUJUTE, R. et al. R-Tree Based Indexing of Now-Relative Bitemporal Data, In: INTERNATION CONFERENCE ON VERY LARGE DATA BASES, 24., 1998, New York, USA. **Proceedings….** [S.l.:s.n.], 1998.

[BOO 94] BOOCH, G. **Object-Oriented Analysis and Desing with Aplication.** Redwood: Benjamin/Cumminngs, 1993.

[BOO 96] BOOCH, G. **Object Solutions**: Managing the Object-Oriented Project. Menlo Park: Addison-Wesley, 1996.

[BOO 99] BOOCH, G.; JACBSON, I.; RUMBAGH, J. **The Unified Modeling Language User Guide.** Menlo Park: Addison – Wesley, 1999.

[CLI 97] CLIFFORD, J. On the Semantics of "NOW" in Databases. **ACM TODS**, New York, v. 22, n.2, p. 171-214, 1997.

[DAT 91] DATE, C.J. **Introdução a Sistemas de Banco de Dados.** Rio de Janeiro: Campus, 1991. Tradução da 4a. edição americana.

[DRI 89] DRISCOLL, J. R. Making Data Structures Persistent. **Journal of Computer and System Sciences**, [S.l.], v.38, n.1, p. 86-124, 1989.

[EDE93] EDELWEISS, N.; OLIVEIRA, J. P. M. de; PERNICI, B. An Object-Oriented Temporal Model. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING - CAISE, 5., 1993, Paris. **Proceedings...** Heidelberg: Springer-Verlag, 1993. p.397-415.

[EDE 94] EDELWEISS, N. Modelagem de Aspectos Temporais de Sistemas de Informação. In: ESCOLA DE COMPUTAÇÃO, 9.,1994, Recife. **Anais...** Recife: UFPE, 1994.

[EDE 2000] EDELWEISS, N. et al. A Temporal Database Management System Implemented on Top of a Conventional Database. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY - SCCC, 20., 2000, Santiago, Chile. **Proceedings…** Los Alamitos: IEEE Computer Society, 2000. p.58-67.

[ELM 90] ELMASRI, R.; WUU, G.T.J.; KIM, Y. J. The Time Index: an access structure for temporal data. In: INTERNATION CONFERENCE ON VERY LARGE DATA BASES, 16., 1990, Brisbane, Australia. **Proceedings...** [S.l.:s.n.], 1990. p.1-12.

[GAM 94] GAMMA, E. et al. **Design Patterns**: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1994.

[GOH 96] GOH, C. H. et al. Indexing temporal data using existing B+-Trees. **Data & Knowledge Engineering,** [S.l.]**,** v.18, n.2, p.147-165, 1996.

[GUT 84] GUTTMAN, A. R-Tree: A Dynamic Index Structure for Spatial Searching. **ACM Sigmod**, Boston, v.12, p. 47-57, 1984.

[HOR 84] HOROWITZ, E.; SAHNI, S. **Fundamentos de Estrtura de Dados.** Rio de Janeiro: Campus, 1984.

[HÜB 2000] HÜBLER, P. N. **Implementação de um Sistema de Banco de Dados Temporal para o Modelo TF-ORM**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[HÜB 2000a] HÜBLER, P.N.; EDELWEISS, N. Implementing a Temporal Database on Top of a Conventional Database. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 2000, João Pessoa. **Anais...** João Pessoa: SBBD, 2000. p.259-272.

[JAC 92] JACOBSON, I. et al. **Object-Oriented Software Engineering**: a Use Case Driven Approuch. Wokwgam: Addison – Wesley, 1992.

[JEN 98] JENSEN, C.S. et al. The Consensus Glossary of Temporal Databases Concepts. Temporal Databases Research and Pratice. Version In: ETZION, O.; JAJODIA, S.; SRIPADA, S. **Temporal Databases**: research and pratice. Berlin: Springer-Verlag, 1998. p.367-405. (Lectures Notes in Computer Science, v.1399).

[JOH 92] JOHSON, R. E. Documenting frameworks using patterns. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERE - OOPSLA, 1992, Vancouver, Canada. **Proccedings...** New York: ACM, 1992.

[JOH 97] JOHNSON, R. E. **Components, Frameworks, Patterns.** [S.1.: s.n.], 1997. White paper.

[KNU 73] KNUTH, D. **The art of computer programming.** Reading, MA: Addison-Wesley, 1973. v.3.

[KOB 99] KOBRYN, C. UML 2001: a Standardization**. Communications of the ACM,** New York, v.42, n.10, Oct 1999.

[KUM 95] KUMAR, A.; TSOTRAS, V. J.; FALOUTSOS, C. Access methods for bi-temporal databases. In: INTERNATIONAL WORKSHOP ON TEMPORAL DATABASES, Zurick, Switzerland. **Proccedings…** Zurick: Spring and Britsj Computer Society, 1995. p235-254.

[LÜ 95] LÜ, J. et al. Indexing techniques for temporal data. INTERNATION CONFERENCE ON VERY LARGE DATA BASES – DEXA, 1994, London, UK. **Proccedings...** [S.l.:s.n.], 1995.

[NAS 95] NASCIMENTO, M. A.; DUNHAM, M. H.; ELMASRI, R. Using

Incremental Trees for Space Efficient Indexing of Bitemporal Databases. In: THE INTERNATIONAL CONFERENCE ON APPLICATION OF DATABASES, 2., 1995, Santa Clara, California. **Proceedings...** [S.l.:s.n.], 1995.

[NAS 96] NASCIMENTO, M. A.; DUNHAM, M. H.; KOURAMAJIAN, V. A Multiple Tree Mapping-Based Approach for Valid-Time Tanges Indexing. **Journal of the Brazilian Computer Society,** São Paulo, v.2, n.3, Apr. 1996.

[RUM91] RUMBAUGH, J. et al. **Object-Oriented Modeling and Design.** Englewood Cliffs:Pretice Hall, 1991.

[SAL 97] SALZBERG, B.; TSOTRAS, V. J. A Comparation of Access Method for Temporal Data. TimeCenter TR-18. **ACM Computing Surveys**, New York, 1997.

[SNO 85] SNODGRASS, R. T.; AHN, I. A Taxonomy of Time in Databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1985, Texas. **Proceedings...** New York: ACM, 1985.

[SNO 86] SNODGRASS, R. T. Temporal Databases. **IEEE Computer**, New York, v.19, n.9, p.35-42, 1986.

[SZW 94] SZWARCFITER, J. L. MARKEZAN, L. **Estrutura de Dados e Seus Algoritmos.** Rio de Janeiro: Ed. Afiliada, 1994.

[TAN 93] TANSEL, A.J. A Generalized Relational Framework for Modeling Temporal Data. In: TANSEL, A.J. et al. **Temporal Databases: t**heory, design, and implementation**. Redwood City: The Benjamin/Cummings, 1993.

[UML99] UML REVISION TASKFORCE. **OMG Unified Modeling Language Specification**, v.1.3. [S.1.]: Object Management Group, 1999. (Document ad99-06-08). Disponível em: http://www.rational.com. Acesso em: out. 2001.

[UML2000] RUMBAGH, J.; BOOCH, G.; JACBSON, I. **UML – Guia do Usuário**. Rio de Janeiro: Campus, 2000.