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

Extensão de um Modelo OO Formal com Aspectos Temporais

por

MELISSA MARCHIANI PALONE ZANATTA

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

Prof. Carlos Alberto Heuser
Orientador

Prof[a]. Nina Edelweiss
Co-orientadora

Porto Alegre, dezembro de 2000

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Zanatta, Melissa Marchiani Palone
Extensão de um Modelo OO Formal com Aspectos Temporais / por Melissa Marchiani Palone Zanatta. – Porto Alegre: PPGC da UFRGS, 2000.
79p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2000. Orientador: Heuser, Carlos Alberto. Co-orientadora: Edelweiss, Nina.
1. dimensão temporal 2. atributos temporais 3. eventos temporais 4. OASIS Temporal I. Heuser, Carlos Alberto. II. Edelweiss, Nina. III.Título.

Dedico esta dissertação

Ao Mauricio, meu marido,
que esteve ao meu lado em todos os momentos, com o seu carinho, incentivo e compreensão.

Aos meus pais,
pelo auxílio, amor e dedicação. Por acreditarem em mim desejando sempre o meu êxito.

Agradecimentos

A DEUS sobre todas as coisas.

A Dona Ana Cardoso Maia de Oliveira Lima em nome UNOESTE- Universidade do Oeste Paulista que, a partir de 1998 proporcionou-me ajuda de custo para as despesas das viagens, hospedagens e mensalidades, tornando possível a conclusão do curso de pós-graduação em ciência da computação da UFRGS.

Ao Moacir Del Trejo em nome da FIPP – Faculdade de Informática de Presidente Prudente, pelo incentivo. Valeu a força!

Aos professores da UPV- Universidade Politécnica de Valência, em especial Oscar Pastor, que me acolheu com carinho e atenção tornando minha viagem produtiva e agradável.

Ao professor Heuser. Sua maneira de orientar, seu jeito sereno e atencioso serviu-me de incentivo para o meu trabalho. Agradeço todas as oportunidades que proporcionou-me. Foi um grande orgulho estudar sob a sua orientação.

A professora Nina. Agradeço pela co-orientação e por ter tido a paciência de revisar vários textos.

Aos professores José Ricardo de R. Zeni e Eliana Maria Trevisan por terem revisado a dissertação e sugerido algumas correções.

Aos professores Álvaro Jose Periotto, Silvely Salomão e Darcy A. Alessi Delfim, pelas suas recomendações, que muito contribuíram para que eu fosse aceita como aluna do Mestrado.

Aos alunos do mestrado remoto de Campo Grande – M.S., especialmente aos amigos Claudio e Lincoln, pelo companheirismo, pelos trabalhos desenvolvidos em grupo e pelas interessantes conversas que mantivemos.

As bibliotecárias e funcionários da UFRGS, em especial à Ida e o Luis Otávio que sempre me atenderam com gentileza e eficiência.

Aos amigos da UFRGS Carina Dorneles, Daniel Notari e Marcos Winckler, pela amizade, atenção e espírito de colaboração.

Aos donos da pensão “Tiazinha Eliane”, Roberto e Eliane, pela hospitalidade, carinho e generosidade.

A Amábile, minha vó, pelas orações e palavras doces, que tornaram a minha caminhada mais tranquila.

A Dona Neide, minha sogra, e ao Marcelo, meu padrinho, pelo apoio.

Aos meus tios (as), primos (as), sobrinhos (as), cunhados (as) pelos momentos de carinho e distração.

Aos meus irmãos, Paula e Junior, pela torcida. Valeu !

Enfim, a todas as pessoas que de alguma maneira, contribuiram para o término desse trabalho.

Sumário

Lista de Abreviaturas

OO	**(*Oriented Object*)** **Orientação a Objetos**
OO-METHOD	**Modelo orientado a objetos**
OASIS	**(*Open and Active Specifications of Information System)*** **Especificação de um Sistema de Informação Ativo e Aberto.**
Mod. de Objetos	**Modelo de objetos**
Mod. Dinâmico	**Modelo dinâmico**
Mod. Funcional	**Modelo funcional**
SGBD	**Sistema Gerenciador de Banco de Dados**
UML	**(*Unified Model Language*)** **Linguagem de Modelos Unificados**
SI	**Sistema de Informação**

Lista de Figuras

Lista de Tabelas

Resumo

Na área de Engenharia de Software, há vários modelos formais de especificação orientado a objetos (OO). Um destes é o OO-Method / OASIS.

OO-Method se baseia nos seguintes princípios:

- dar suporte às noções do modelo conceitual orientado a objetos;
- integrar os modelos formais com metodologias de aceitação industrial;
- possibilitar a produção de software avançado que inclua a geração completa de código (estática e dinâmica) do desenvolvimento comercial.

O processo de desenvolvimento consiste em levantar as propriedades principais do sistema em desenvolvimento (modelo conceitual) por parte do engenheiro de software, e construir de forma automática, em qualquer momento (por um processo de conversão gráfico-textual) a especificação formal orientada a objetos em OASIS (*Open and Active Specifications of Information System*) que constituirá um repositório de alto nível do sistema.

O objetivo de OASIS é expressar os requisitos funcionais de um sistema de informação, em um marco formal, que facilite sua validação e geração automática de programas.

OASIS não inclui a especificação de aspectos temporais. A modelagem de aspectos temporais é um importante tópico da modelagem de sistemas de informação, porque através destes são representadas as características dinâmicas das aplicações e a interação temporal entre diferentes processos.

A especificação de requisitos de aplicações através de modelos orientados a objetos permite representar não só os seus estados, mas também, seu comportamento. Modelos temporais representam também a evolução de objetos com o tempo. Como o estado de um objeto pode ser alterado devido à ocorrência de um evento (fato ocorrido em um determinado instante no tempo), é importante que o modelo utilizado permita apresentar a história destes eventos.

O presente trabalho tem por finalidade propor uma extensão temporal a um modelo formal de especificação OO. Esta extensão inclui tanto aspectos estáticos quanto dinâmicos. A extensão de aspectos estáticos estende OASIS com atributos temporais. A extensão dos aspectos dinâmicos, contribuição central do trabalho, estende OASIS com *eventos temporais*.

Palavras-chave: dimensão temporal, atributos temporais, eventos temporais, OASIS Temporal.

Title: Extension of a Formal OO Model with Temporal Aspects

Abstract

In the Software Engineering area, several formal object-oriented (OO) specification are available. One of these is the OO-Method / OASIS.

OO-Method is based on the following principles:

- to give support to the object-oriented conceptual model concepts;
- to integrate formal models with methodologies of industrial acceptance;
- to allow the production of advanced software that includes the full generation of code (static and dynamic) in commercial development.

The development process consists of raising the main properties of the system in development (conceptual model) by the software engineer, and constructing in an automatic form, at any moment (by means of a graphical-textual conversion process) the formal object-oriented specification in OASIS (Open and Active Specifications of Information System) that it will constitute a high level repository of the system.

The objective of OASIS is to express the functional requirements of an information system, in a formal way, improving the automatic validation and generation of programs.

OASIS does not include the specification of temporal aspects. The modeling of temporal aspects is an important topic in information systems modeling, because through these the dynamic features of the applications and the temporal interaction between different processes are represented.

The specification of applications requirements through object-oriented models allows not only to represent its states, but also, its behavior. Temporal models also represent the object evolution with time. As the state of an object can be modified due to occurrence of an event (fact occurred in one specific instant of time), it is important that the used model allows the represntation of the history of these events.

The present work proposes a temporal extension to a formal OO specification model. This extension includes static and dynamic aspects. The extension of static aspects extends OASIS with temporal attributes. The extension of dynamic aspects, central contribution this work, extends OASIS with *temporal events*.

Keywords: temporal dimension, temporal attributes, temporal events, Temporal OASIS

1 Introdução

1.1 Especificação de um sistema de informação (SI) ao nível conceitual

A Universidade Federal do Rio Grande do Sul e a Universidade Politécnica de Valência (Espanha), desenvolvem um projeto conjunto com o objetivo de enriquecer a expressividade da produção automática de um modelo chamado OO-Method (figura 1.1) com características temporais.

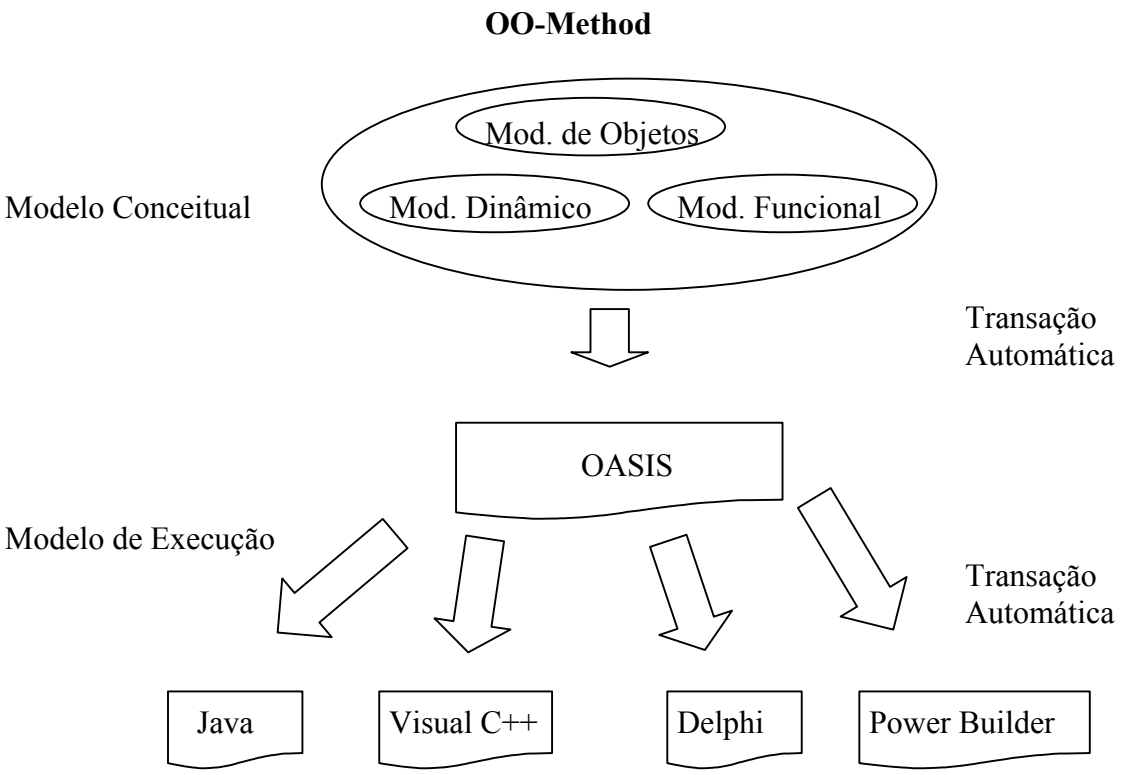

FIGURA 1.1- Fases do OO-Method

Em OO-Method a especificação é feita através de três modelos gráficos obtidos na fase de análise (modelo de objetos, modelo dinâmico e modelo funcional). A notação destes modelos baseia-se em representações gráficas conhecidas na literatura [LAR 97].

Os modelos gráficos são transfomados automaticamente em uma especificação formal OO, OASIS.

A partir de OASIS, é possível gerar completamente aplicações em várias linguagens de programação.

Entretanto, OASIS, como outros formalismos OO [JUN 95, SER 92] não possui primitivas para tratar o aspecto temporal.

A especificação de informações temporais é fundamental para que se possa capturar corretamente os aspectos estáticos e dinâmicos dos componentes do sistema estudado, assim como sua evolução temporal.

A noção de tempo como datas, períodos, duração de validade de informações, intervalos temporais, surge em diferentes níveis: (i) na modelagem de dados, (ii) na linguagem de recuperação e manipulação de dados e (iii) no nível de implementação do SGBD [EDE 94b].

Na literatura sobre linguagens formais para especificação de software, há várias propostas de extensão temporal das propriedades estáticas de objetos [ANT 97, BAT 92, CLI 95, EDE 94a, HEU 96, TAN 93]. Nestas propostas, é possível especificar que uma classe, um atributo, ou ainda, uma associação são temporais. Dizer que uma classe é temporal, significa dizer que a história de toda a existência de uma instância daquela classe pode ser utilizada. Dizer que um atributo é temporal, significa dizer que na linguagem de especificação é possível falar sobre a história dos valores assumidos pelo atributo em questão. Associações temporais, por sua vez, referem-se à variação das associações entre entidades com o tempo.

A limitação de tratar tão somente a temporalidade de aspectos estáticos vem do fato de os primeiros modelos terem base na abordagem relacional [CLI 87, GAD 93, LOR 88, SAR 90, SNO 87, TAN 86], ou na abordagem entidade-relacionamento [ELM 93, LOU 91, TAU 91].

Modelos orientados a objetos encapsulam estado e comportamento. O estado de um objeto pode ser alterado devido a ocorrência de algum evento. Um evento é um fato instantâneo, que ocorre em um determinado instante no tempo [ZAN 9?]. Assim, é natural considerar também, a temporalidade de aspectos dinâmicos. Neste contexto, isto significa que a linguagem de especificação deve permitir falar sobre a história das ocorrências de um determinado evento.

Em modelos temporais que consideram apenas aspectos temporais estáticos, é necessário usar atributos temporais especiais, para representar a história de um evento. Por exemplo, em uma vídeo-locadora o "empréstimo" de uma fita constitui um evento. Caso seja necessário falar sobre a história dos empréstimos de um usuário, é necessário criar um atributo temporal, como por exemplo, "data_de_empréstimo", para manter a história dos empréstimos.

Neste trabalho, propõe-se estender um modelo formal orientado a objetos de modo a permitir que sejam especificados *eventos temporais*. No exemplo acima, o evento "empréstimo" passaria a ser considerado temporal. Isto significa que o sistema, internamente, mantém um registro de todas ocorrências deste evento. Desta forma, o usuário pode, na linguagem de especificação, referenciar o histórico das ocorrências deste evento. Com isto é possível especificar situações, tais como, o fato de um usuário da vídeo-locadora poder receber um cartão de cliente especial, caso tenha alugado um determinado número de fitas em um período, ou de um filme poder ter seu preço de aluguel reduzido, caso ocorram poucos empréstimos em um período.

1.2 Estrutura do texto

Este trabalho está organizado da seguinte maneira.

O segundo capítulo apresenta o modelo OASIS, como uma linguagem de especificação com uma sintaxe textual bem definida, que será estendida com aspectos temporais.

O terceiro capítulo mostra um estudo de caso de uma vídeo-locadora, como um exemplo de aplicação do modelo OASIS em um modelo conceitual de sistema de informação.

O quarto capítulo descreve as várias formas de acrescentar a dimensão temporal a um modelo dados, para que este possa representar a variação das informações ao longo do tempo e, apresenta a representação temporal escolhida para estender OASIS. No final do capítulo, após a descrição de alguns modelos OO, é apresentada uma tabela comparando os modelos com base nos principais aspectos para utilização de um modelo temporal.

O quinto capítulo é o ponto principal da presente dissertação, onde a extensão temporal dos aspectos estáticos e dinâmicos de OASIS é descrita (OASIS Temporal), através da definição de domínios temporais e classes temporais. A ênfase deste capítulo é a modelagem das propriedades dinâmicas de um sistema, integradas ao modelo de dados temporal descrito no quarto capítulo.

O sexto capítulo apresenta o mesmo estudo de caso do terceiro capítulo, com a extensão temporal, isto é, com a especificação em OASIS Temporal.

Na conclusão da dissertação apresentada no oitavo capítulo, estão expostas as contribuições proporcionadas pelo OASIS Temporal, bem como são apontados os possíveis estudos futuros visando aperfeiçoar a linguagem.

Adicionalmente, é apresentado um glossário de conceitos utilizados ao longo da dissertação.

Por último é apresentada a bibliografia utilizada para a realização desse trabalho.

2 Linguagem OASIS

Este capítulo contém uma introdução a OASIS. Esta introdução apresenta apenas os conceitos necessários a compreensão do trabalho. Para uma visão detalhada da linguagem, ver o livro que a documenta [LET 98b].

2.1 Notação usada na apresentação de OASIS

A seguir é apresentada a notação utilizada para apresentar OASIS. Essa notação é extraída do livros de OASIS [PAS 95a, LET 98b].

- símbolo terminal ou caracteres entre ‘ ’;
- símbolo não terminal entre < > ;
- símbolos opcionais entre [ ];
- símbolos alternativos separados por | .

As simplificações utilizadas para apresentar OASIS são apresentadas abaixo.

- sendo x um elemento em particular, para descrever um bloco de elementos x, usa-se <bloco_x>;
- para uma lista de elementos x, usa-se <lista_x>;
- para uma sequência de elementos x , usa-se <sec_x>.

Estas simplificações são definidas conforme apresentação abaixo.

```
<bloco_x>  := <bloco_x> <x> | <x>
<lista_x>    := <lista_x> ‘,’ <x> | <x>
<sec_x>     := <sec_x> <x> ‘;’ | <x> ‘;’
```

São listados abaixo, alguns símbolos não terminais usados freqüentemente:

- não terminal <fórmula>: é uma fórmula bem formada em lógica de predicados de primeira ordem, baseada em valores constantes, atributos e/ou parâmetros de ações;
- não terminal <expressão>: é uma expressão booleana ou aritmética baseada em valores constantes, atributos e/ou parâmetros de ações;
- não terminal <valor>: é um elemento de algum valor dos domínios permitidos.

2.2 Modelo conceitual

Em OASIS, um modelo conceitual corresponde a um conjunto de elementos de especificação, incluindo domínios, classes simples, classes complexas e interfaces [LET 98b, PAS 95a, PAS 95b].

O modelo conceitual de um sistema de informação corresponde em OASIS à especificação de um esquema conceitual, que está definido abaixo.

```
<def_esquema_conceitual> ::= conceptual_schema <id_esquema>
                             [ <bloco_def_domínio> ]
                             <bloco_def_classe>
                             [<bloco_def_classe_complexa>]
                             <bloco_def_interface>
                             end_conceptual_schema
```

Em seguida, serão apresentadas em detalhes as especificações de cada um dos blocos mencionados acima [LET 98b].

2.3 Tipos de dados (domínios)

Em OASIS há vários tipos de dados: *nat, int, real, bool, char, string, date, time e blob* ("*binary large objects*"). O especificador tem a possibilidade de declarar outros tipos abstratos de dados e utilizá-los.

Por exemplo, neste trabalho, são definidos tipos para representar os preços que uma fita de vídeo obteve na vídeo-locadora desde que esta foi cadastrada até o dia de hoje (dia atual do sistema). Para isto, é definido um atributo "preço" cujo domínio são os números reais com dimensão temporal por intervalo de tempo.

A seguir é apresentado a sintaxe para especificação de um domínio.

```
<def_domínio>   ::= domain <id_domínio>
                    '(' <lista_id_domínio> ')'
                    [import <lista_id_domínio>] ';'
                    [var <lista_def_variáveis>] ';'
                    functions <sec_def_funções>
                    equations <sec_def_equações>
                    end domain
<def_variáveis>  ::= <id_variáveis> ':' <domínio>
<def_funções>    ::= <id_funções>
   ['('lista_def_parâmetros')']':'<domínio>
<def_equações>   ::= <fórmula_equacional>
```

Os tipos de dados genéricos podem ser especificados através de uma lista de domínios entre parênteses, indicando que tais tipos de dados estão parametrizados por outros tipos base.

A seção de ***import*** permite utilizar tipos abstratos de dados previamente definidos.

A seção ***equations*** é uma seqüência de fórmulas bem formadas da lógica equacional, do tipo ter1 = ter2, onde ter1 e ter2, são terminações que descrevem como trabalham as funções.

2.4 Especificação da Classe

Uma classe é composta por um nome da classe e um *template*. O *template* da classe está dividido em seções e parágrafos.

A sintaxe para definir uma classe é apresentada a seguir.

```
<def_classe>      ::=   class <id_classe>
                  <mecanismos_identificação>
                  <atributos_constantes>
                  [ <atributos_variáveis> ]
                  [ <atributos_derivados> ]
                  [ <derivações> ]
                  [ <restrições_integridade> ]
                  <eventos>
                  [ <operações> ]
                  [ <gatilhos> ]
                  [ <avaliações> ]
                  [ <pré-condições>]
                  [ <protocolos> ]
```

```
                    end class
```

Além das seções opcionais mostradas acima, se uma classe for definida por herança a partir de outra classe, então todas as seções são opcionais.

Se <id_classe> coincide com o identificador do esquema conceitual assume-se que se trata da descrição de propriedades globais associadas ao da classe implicitamente estabelecida por agregação de todas as classes definidas no esquema conceitual.

As próximas sub-seções descrevem detalhadamente os *template*s da classe. As operações e os protocolos não são descritos neste trabalho pelo fato de não serem utilizados.

2.4.1 Mecanismos de identificação

É uma função que estabelece correspondência entre valores de atributos de um objeto e seu *oid* (objeto identificador*)*. Uma mesma classe pode ter distintos mecanismos de identificação.

Os mecanismos de identificação apresentam a seguinte sintaxe:

```
<mecanismos_identificação>    ::= identification
                                  <sec_def_identificador>
<def_identificador>           ::= <id_identificador>
                                  ':' '('<lista_id_atributo>')'
```

Um mecanismo de identificação é composto por uma lista de atributos constantes. A lista de valores associada a estes atributos constantes, deve determinar unicamente um objeto da classe.

Para exemplificar, suponha que em uma vídeo-locadora exista a classe usuário. Nesta classe, o número do usuário é baseado no atributo constante "n_usuário", conforme apresentado abaixo.

```
class usuario
identification
        número_usuario: (n_usuario);
constant attributes
        n_usuario: string;
        …
```

2.4.2 Atributos

O estado dos objetos é observado através dos valores de seus atributos. Os atributos são propriedades estruturais das instâncias de uma classe e possuem um tipo associado.

Em OASIS distinguem-se três tipos de atributos:

- atributos constantes: aqueles que recebem seu valor quando se cria o objeto e não mudam com o passar do tempo;
- atributos variáveis: aqueles cujos valores podem alterar quando ocorre uma ação relevante;
- atributos derivados: que são derivados a partir de outros atributos.

Para cada atributo definido existem funções e operações associadas, disponíveis para serem usadas em especificações do *template,* e que são úteis para manipular atributos cuja cardinalidade é maior que um.

Atributos constantes e variáveis utilizam uma mesma sintaxe definida abaixo.

```
<atributos_constantes>  ::=   constant attributes
                              <sec_def_atributo_cons_ou_var>
<atributos_variáveis>   ::=   variable attributes
                              <sec_def_atrib_cons_ou_var>
<def_atrib_cons_ou_var> ::= <id_atributo> ':' <domínio>
                            ['('<lista_valor_por_default>')']
                            [towards <cardinalidade>
                               [<def_representação>]]
<valor_por_default>     ::= <valor>
<cardinalidade>         ::= '('<card_min>','<card_max>')'
<def_representação>     ::= list '['<def_índice>']'| set | bag
<def_índice>            ::= <intervalo> | <lista_corrente>
```

Os atributos podem ter valor por *default* quando se cria o objeto. O especificador pode introduzir entre parênteses este valor por *default* na declaração do atributo.

A cláusula ***towards*** <cardinalidade> apresenta as cardinalidades vistas desde que o objeto tenha o domínio primitivo do atributo. Se não especificar assume-se ***towards*** (0,1).

Deve-se garantir que <card_min> <= <card_max>.

Os valores dos atributos constantes e variáveis são parâmetros implícitos do evento ***new*** do objeto. Se para algum atributo tem-se <card_mim> >=1, então é obrigatório que a partir da criação do objeto deste atributo a condição seja satisfeita, isto é, tenha pelo menos essa quantidade de valores. Se o parâmetro associado a um atributo constante ou variável não está instanciado no evento ***new*** e existe um valor por *default* declarado, então o atributo receberá este valor.

Quando o atributo é multivalorado (<card_max> >1), ***list***, ***set*** e ***bag*** permitem indicar qual é a representação escolhida. Em ***list*** e ***bag*** permitem elementos duplicados, ***set*** não. Em ***list*** a declaração <def_índice> permite definir a forma de referência usada para acessar os elementos da lista. Se não se especifica ***list***, ***set*** e ***bag***, assume-se que é do tipo ***set***.

Por exemplo, é definida abaixo, a declaração do atributo "telefone" com domínio *string*, multivalorado (*towards* (0,*)) e com representação *list,* na classe usuário da vídeo-locadora.

```
Variable attributes
        telefone:string towards (0,*) list [1..*];
        …
```

O asterisco (*) indica que não existe restrição a respeito da cardinalidade máxima (em ***towards***), e no máximo valor do número (em ***list***).

A sintaxe para especificação dos atributos derivados é a seguinte:

```
<atributos_derivados>   ::= derived attributes
                            <sec_def_atrib_deriv>
<def_atrib_deriv>       ::= <id_atributo> ':' <domínio>
```

Para atributos derivados, o domínio declarado deve ser consistente com sua especificação na seção ***derivations***, na qual determina o tipo de atributo (constante ou variável) e sua cardinalidade e representação.

Suponha que exista um atributo derivado na classe multa, que irá guardar o cálculo do valor da multa conforme especificação a seguir:

```
derived attributes
        valor_multa: nat;
        …
```

2.4.3 Derivações

São usadas para especificar um atributo derivado.

A sintaxe para especificação das derivações é apresentada a seguir.

```
<derivações>            ::= derivations
                              <sec_def_atrib_deriv>
<sec_def_atrib_deriv>   ::=[‘{‘<fórmula>‘}’]< atribuições>
<atribuições>           ::= <id_atributo> ‘:=’ <expressão>
```

Para exemplificar, considere a definição abaixo, de uma derivação na classe multa para definir o valor do atributo derivado “valor_multa”, que depende do número de dias que o usuário ficou em atraso na vídeo-locadora.

```
{dias_atraso = 2} valor_multa = fita.preço + 2 * valor;
```

O atributo “valor” armazena o valor da multa diária. Esse exemplo expressa que se a quantidade de dias em atraso na devolução da fita for igual a dois ({dias_atraso = 2}), o valor da multa (“valor_multa”) será o preço da fita (“fita.preço”) multiplicado pelo valor da multa diária (“valor”).

Esse mesmo exemplo pode ser definido da seguinte forma:

```
valor_multa:={{day(currenttime)-day(data_mul) * valor};
```

Dessa forma, a quantidade de dias em atraso é calculada pelo próprio sistema.

2.4.4 Restrições de Integridade

São fórmulas baseadas no estado do objeto e que devem ser satisfeitas para cada um de seus estados. Podem ser classificadas em estáticas (referem-se a somente um estado) e dinâmicas (relacionam diferentes estados). Operadores temporais tradicionais são utilizados para especificar restrições de integridade dinâmicas.

A sintaxe para especificação de restrição de integridade é apresentada abaixo.

```
<restrições_integridade>::=   constraints <sec_def_restrições>
<def_restrições>        ::=
                  [always ] ‘{‘ <fórmula> ‘}’
                  | sometimes ‘{‘ <fórmula> ‘}’
                  | next ‘{‘ <fórmula> ‘}’
                  | <fórmula_depois> since <fórmula_antes>
                  | <fórmula_antes> until <fórmula_depois>
                  | sometimes [ ‘(‘<timeout> ‘)’ ] <fórmula_depois>
                              since <fórmula_antes>
                  | always [ ‘(‘<delay> ‘)’ ] <fórmula_depois>
                              since <fórmula_antes>
```

```
<fórmula_antes>   ::=         '{' <fórmula> '}'
<fórmula_depois>  ::='{' <fórmula> '}'
<timeout>         ::= <op_rel_timeout> <natural> <unidade_tempo>
<delay>           ::= <op_rel_delay> <natural> <unidade_tempo>
<op_rel_timeout>  ::= '<' | '<='
<op_rel_delay>    ::= '>' | '>='
<unidade_tempo>   ::= seconds | minutes | hours | days
                        weeks | months | years
```

Se na verificação das restrições de integridade algumas delas não forem satisfeitas, o estado do objeto deve ser o último estado no qual se cumpriu a restrição de integridade. Os estados posteriores a última situação de integridade são ignorados.

Uma restrição de integridade que utiliza o operador ***sometimes*** sem especificar o limite de tempo, tem um sentido equivalente a de uma pré-condição para o evento ***destroy*** do objeto.

Suponha o exemplo da vídeo-locadora. A seguir é apresentada uma restrição de integridade pode ser definida na classe fita para restringir que o número de fitas tem que ser diferente de vazio (para garantir a existência de fitas vídeo-locadora).

```
Constraints
        {n_fita <> '' }
```

Pela extensão temporal proposta no quinto capítulo, não é necessário o uso dos operadores temporais nas restrições de integridade. A sintaxe fica definida como foi proposta na primeira versão de OASIS [PAS 92b].

2.4.5 Eventos

Um evento ocorre em um instante de tempo. Sua ocorrência resulta em algumas ações, que podem causar mudanças de estado em alguns objetos.

Abaixo é apresentada a sintaxe para especificação de eventos.

```
<eventos>    ::= events
                   <sec_def_evento>
<def_evento>::=
             <id_evento> ['('lista_def_parâmetro')']
             [new | destroy ]
             [alias for <operações_implícitas> | <coordenação>]
<coordenação>::=  calling to members
                  <lista_serviço_componente>
                  | sharing with members
                  <lista_serviço_componente>
```

As cláusulas ***new*** e ***destroy*** indicam respectivamente do evento de criação e destruição de instâncias. Ambos eventos são únicos para cada classe. O evento ***new*** tem implicitamente como parâmetros, os valores para os atributos constantes e variáveis do objeto.

O evento ***new*** cria o objeto e deve ser o primeiro evento da vida do objeto. O evento ***destroy*** é a causa da destruição de um objeto. Quando uma classe não tem o evento ***destroy*** declarado assume-se que esta classe é "imortal".

Há uma distinção entre eventos privados (quando não tem a cláusula ***calling to members*** *ou* ***sharing with members***), eventos implicados (com a cláusula ***calling to members***) e eventos compartilhados (com a cláusula ***sharing with members***). Os

eventos privados são serviços oferecidos ou requeridos por um objeto e cuja ocorrência depende somente do comportamento do objeto. Nos eventos implicados a ocorrência do evento no objeto agregado implica necessariamente na ocorrência do correspondente serviço em seus objetos componentes. Quando se trata de eventos compartilhados, esta implicação se dá em ambos os sentidos.

Os eventos implicados e compartilhados somente se definem em uma classe agregada. Em ambos os casos, deve-se estabelecer uma correspondência entre o evento do agregado e o serviço do componente, mediante a definição da lista <lista_serviço_componente>.

Considere a definição abaixo da classe usuário da vídeo-locadora onde existem entre outros o evento para cadastrar um usuário (“insere_usu”) e para remove-lo (“remove_usu”) da vídeo-locadora.

```
class usuario
....
events
        insere_usu new;
        remove_usu destroy;
```

2.4.6 Avaliações

São fórmulas na lógica dinâmica que estabelecem como as ações afetam no estado do objeto. Os atributos variáveis modificam seus valores segundo o estabelecido nas avaliações.

A seguir é apresentada a sintaxe para especificação das avaliações.

```
<avaliações>       ::= valuations
                       <sec_def_avaliação>
<def_avaliação>    ::= [‘{‘<fórmula>‘}’]‘[‘<ações>‘]’
                       <lista_atribuição>
<atribuição>       ::=   <id_atributo> ‘:=’ <expressão>
```

Os atributos constantes e derivados não devem ser um atributo de atribuição. Por outro lado, todo atributo variável deve ter associado ao menos uma atribuição em alguma avaliação.

O domínio resultante da <expressão> deve ser consistente com o domínio do atributo.

Um exemplo especifica que o atributo “status_pagto” pertencente a classe multa da vídeo-locadora, estabelece se o usuário pagou a multa ou não. Esse atributo é afetado pelo evento “altera_status_pagto”. Esta especificação é a seguinte:

```
class multa
        …
        events
                altera_status_pagto;
        valuations
                [altera_status_pagto] {status_pagto=true};
```

2.4.7 Pré-condições

Através de pré-condições é possível limitar a ocorrência de determinados eventos.

A seguir é apresentada a sintaxe para especificação das pré-condições.

```
<pré-condições>          ::= preconditions
                             <sec_def_pré-condições>
<def_pré-condições>      ::= <ações> if '{'<fórmula>'}'
```

Os eventos de criação e destruição (***new*** e ***destroy***) têm como pré-condição (*default*) a não existência e existência da instância que vai ser criada ou destruída respectivamente.

Para exemplificar, considere o evento "emprestar" na classe fita da vídeo-locadora. A pré-condição definida abaixo especifica que para esse evento ocorrer é importante que a fita esteja disponível.

```
class fita;
....
events
        emprestar;
preconditions
        emprestar if {disponível:=true}
```

2.4.8 Gatilhos

Um gatilho especifica um evento que deve ocorrer quando um objeto atinge um determinado estado.

A sintaxe é apresentada abaixo.

```
<gatilhos>     ::= triggers
                   <sec_def_gatilho>
<def_gatilho>  ::= <ações> when '{'<fórmula>'}'
```

O objeto que tem a definição de gatilho deve ser cliente, servidor, ou ambos na ação.

Os gatilhos, por representar uma obrigação, devem ser consistentes com as permissões definidas mediante pré-condições e protocolos.

Considere na classe multa o gatilho abaixo, para zerar o valor da multa quando o atributo "status_pagto" for verdadeiro.

```
class fita;
....
events
        altera_valor_multa(0);
triggers
        :: altera_valor_multa (0)  when {status_pagto =true}
```

2.5 Classes Complexas

Uma classe complexa utiliza em sua definição outras classes (simples ou complexas). Os operadores para construção de classes complexas são agregação e herança.

A seguir é apresentado a sintaxe para especificação das classes complexas.

```
<def_classe_complexa> ::= <def_agregação>
                          |<def_herença>
```

2.5.1 Agregação

Uma agregação modela a noção de relações estruturais entre objetos. Uma associação é representada mediante uma agregação multivalorada definida com somente um componente.

A sintaxe para especificação de agregação é a seguinte:

```
<def_agregação>   ::= <id_classe> aggregation of <partes>
<partes>          ::= <sec_def_componente> | <def_associação>
<def_componente>  ::=
                  [static | dynamic]
                  [inclusive | relational]
                  [<id_alias> alias for]
                  <id_classe_componente>
                  [towards <cardinalidade>[<def_representação>]]
                  [from <cardinalidade>[<def_representação>]]
```

A seguir a sintaxe para especificação de associação é apresentada.

```
<def_associação>  ::=
                  <id_classe_componente>
                  grouping by <lista_id_atributo>
                  [ where '{' <fórmula> '}' ]
                  [towards <cardinalidade> [ <def_representação> ]]
                  [ from <cardinalidade> [ <def_representação> ]]
<cardinalidade>   ::='(' <card_min> ',' <card_max> ')'
<def_representação>     ::= list '['<def_índice>']' | set | bag
<def_índice>            ::= <intervalo> | <lista_corrente>
```

A cláusula ***static*** implica que os objetos do componente não podem ser eliminados, nem inseridos. Já ***dynamic***, os objetos do componente podem modificar-se. Se não especificar se é ***static*** ou ***dynamic***, se assume ***dynamic***. Uma associação sempre será ***dynamic***.

A cláusula ***inclusive*** implica que os objetos do componente somente podem interagir com outros comunicando-se através do objeto agregado. A cláusula ***relational*** significa que os objetos do componente podem comunicar-se diretamente com outros, podendo não existir coordenação com o objeto agregado. Se não especificar se é ***relational*** ou ***inclusive***, se assume ***relational***. Uma associação sempre será ***relational***.

Para diferenciar em uma agregação entre os componentes definidos sobre a mesma classe usa-se ***alias for***.

Para representar as cardinalidades vistas desde o objeto agregado até o objeto do componente usa-se ***towards*** <cardinalidade>. Se não especifica ***towards*** <cardinalidade>, se assume que é ***towards*** (0,*).

A cláusula ***from*** <cardinalidade> representa as cardinalidades vistas desde um objeto do componente ao agregado. Se não especifica ***from*** <cardinalidade>, se assume que é ***from*** (0,*). Deve cumprir que <card_min> <= <card_max>.

Para indicar qual é a representação escolhida, de forma opcional e no caso de ser multivalorada (<card_max> >1), é utilizado as cláusulas ***list***, ***set*** e ***bag***. Em ***bag*** é permitido elementos duplicados; em ***list*** a declaração <def_índice> permite definir a

forma de referência usada para acessar os elementos da lista. Se não é especificado, assume-se que é do tipo ***set***.

A notação "<def_associação>" permite definir uma associação como uma agregação na qual somente existe um componente.

A cláusula ***grouping by*** <lista_id_atributo> indica que cada instância do componente (grupo de objetos) se cria (ou destroí) automaticamente agrupando os objetos da classe do componente, segundo os valores destes atributos nos objetos da classe do componente.

Em uma associação, ao realizar-se o agrupamento de instâncias da especificação ***where*** permite selecionar os objetos da classe do componente que satisfazem a fórmula especificada.

Um exemplo de agregação na vídeo-locadora é a classe locação. Nesta classe fica definido qual fita foi emprestada (evento "emprestar" da classe fita) e quem a emprestou (evento "emprestar" da classe usuário). A especificação em OASIS está apresentado a seguir:

```
locação aggregation of
        static relational usuario towards (1,1) from (0,*)
        static relational fita towards (1,1) from (0,*)
```

2.5.2 Visibilidade desde o Agregado até o Componente

Independentemente do tipo de agregação, existe visibilidade de atributos do componente desde o agregado.

A sintaxe é apresentada a seguir.

```
<atributo_componente> ::= <ref_objeto> '.' <id_atributo>
                          |<ref_objeto> '.'<atributo_componente>
<serviço_componente>  ::= <ref_objeto> '.'<serviço>
                          |<ref_objeto> '.'<serviço_componente>
```

2.5.3 Assinatura Implícita

Para cada componente e atributo definido existem funções e operações associadas disponíveis para serem utilizadas nas fórmulas do *template*. Estas funções e operações são úteis para manipular componentes (ou atributos) cuja cardinalidade é maior que um. A tabela 2.1 apresenta as funções e operações implícitas para atributos e componentes.

TABELA 2.1- Funções e operações implícitas para atributos e componentes

FUNÇÕES
position (atributo/componente,<referência>)
count (atributo/componente) [where '{'<fórmula>'}']
min (atributo/componente) [where '{' <fórmula>'}']
max (atributo/componente) [where '{' <fórmula>'}']
sum (atributo/componente) [where '{' <fórmula>'}']
avg (atributo/componente) [where '{' <fórmula>'}']
first (atributo/componente [,mecanismo])
last (atributo/componente [,mecanismo])
OPERAÇÕES
insert (atributo/componente,<referência>)
insert (atributo/componente [índice], <referência>)

```
remove (atributo/componente [índice])
remove (atributo/componente, <referência>)
remove_all (atributo/componente )
```

A notação "(atributo/componente)" é o atributo ou componente ao que se aplica a função ou operação.

Quando trata-se de um objeto componente, a notação "<referência>" é uma referência a objeto. No caso de atributos "<referência>" coincide com o valor do *sort* do atributo.

"Mecanismo" é o nome de um mecanismo de identificação da classe do componente. Para o caso de um atributo não é necessário.

A função ***position*** devolve a posição de um valor ou de um objeto na lista atributo/componente.

A função ***count*** devolve o número de valores ou instâncias no atributo/componente. Quando o atributo está representado por ***bag***, contam-se também os valores duplicados.

As funções ***min***, ***max***, ***sum*** e ***avg***, permitem calcular respectivamente o mínimo e máximo, a soma e a média dos valores de um atributo ou valores de um componente.

As funções ***count***, ***min***, ***max***, ***sum*** e ***avg*** podem restringir os valores ou instâncias sobre os que se aplicam, usando a especificação ***where***. Quando a função é aplicada a um atributo, a fórmula associada a ***where*** utiliza a palavra ***value*** para determinar o valor do atributo.

As funções ***first*** e ***last*** devolvem respectivamente o primeiro e último valor ou mecanismo de identificação no atributo/componente.

No caso de uma lista de operações ***insert*** e ***remove*** produzem a reorganização da lista.

"Índice" é o valor permitido de acordo com a definição do índice dado pela lista.

Para exemplificar, suponha que o proprietário da vídeo-locadora quer saber a quantidade de fitas existentes. Para isto ele utiliza a função ***count***. Esse exemplo é expresso a seguir:

```
total_fitas_vídeo = count (fita)
```

2.5.4 Considerações Adicionais para Agregação

As diferentes possibilidades de agregação tem conseqüências importantes, considerando as associações:

a) ***inclusive***: só pode ceder aos serviços do componente mediante aos serviços do objeto agregado;
b) ***static***: não se dispõe das operações implícitas ***insert*** e ***remove*** depois da criação do objeto agregado;
c) ***towards*** (min, max):
   c1) se min >= 1, junto ao evento ***new*** do agregado devem ocorrer ao menos min ocorrências do evento ***insert*** associadas ao componente;
   c2) cada inserção do componente tem com pré-condição implícita agregada: *evento_insert if count (componente) < max;*
   c3) cada vez que se remove um componente tem como pré-condição implícita: *evento_remove if count (componente) > min*
d) ***from*** (min, max):

regras iguais ao item c, apresentadas acima.

2.5.5 Herança

Mediante a herança podemos especializar (ou generalizar) propriedades definidas nas classes.

A sintaxe para especificação de herança é apresentada a seguir.

```
<def_herança> ::= <def_partição_estática>
                  | <def_partição_dinâmica>
                  | <def_regras>
```

As especificações de uma classe são tratadas separadamente de suas relações de heranças existentes. Com isto, tanto as classes simples como as classes complexas têm o mesmo *template* de classe.

Não se permitem definições circulares porque as partições devem estar formadas por classes que não tenham sido particionadas previamente.

Considerando a compatibilidade de comportamento das partições estáticas e dinâmicas, deve cumprir-se o seguinte:

- as permissões (pré-condições e protocolos) nas subclasses devem implicar logicamente as permissões das superclasses;
- as avaliações nas subclasses somente podem modificar atributos emergentes;
- as obrigações (gatilhos e operações) na subclasse devem implicar logicamente as obrigações da superclasse correspondente;
- as condições associadas às restrições de integridade nas subclasses devem implicar logicamente nas condições associadas a estas restrições nas superclasses.

2.5.6 Partições Estáticas

As instâncias das subclasses definidas por partições estáticas estão associadas a uma partição a partir de sua criação e se mantêm nela durante toda sua vida.

A sintaxe é apresentada abaixo.

```
<def_partição_estática> ::=   <lista_id_subclasse>
                          static specialization of
                          <id_classe>
```

A <lista_id_subclasse> são subclasses da mesma partição estática sobre a classe determinada por <id_classe>

Para exemplificar, considere a definição abaixo das classes “fita_lançamento” e “fita_promoção” na vídeo-locadora. Ambas são especialização da classe fita.

```
fita_lançamento, fita_promocao
        static specialization of fita;
```

2.5.7 Partições Dinâmicas

Dispõem-se de duas formas alternativas para especificar o processo de migração: pela ocorrência de certas ações em determinados estados do objeto e por partição dos possíveis estados do objeto em função dos valores que apresentem seus atributos.

A seguir é apresentada a sintaxe para definir uma partição estática.

```
<def_partição_dinâmica>::= <def_dinâmica_migração>
                           |<def_dinâmica_atributo>
```

Partições dinâmicas podem ser definidas pela ocorrência de eventos específicos. A sintaxe para este tipo de partição é a seguinte:

```
<def_dinâmica_migração> ::= <lista_id_subclasses>
                            dynamic specialization of
                              <id_superclasse>
                            migration relation is
                              <expressão_migração>
<expressão_migração>    ::= <sec_def_estado>
```

Uma especialização dinâmica relativo ao estado civil do usuário na vídeo-locadora pode ser: solteiro, casado, divorciado, viúvo, conforme especificação a seguir:

```
dynamic specialization of usuario migration relation is
        usuario = nascer.solteiro;
        solteiro = casar-se.casado;
        casado = divorciar-se.divorciado + enviuvar-se.viuvo;
        viuvo= casar-se.casado;
        divorciado = divorciar-se.divorciado;
```

Abaixo é apresentada as partições dinâmicas em função de valores de atributos.

Sintaxe:

```
<def_dinâmica_atributo>       ::= <lista_def_subclasse_dinâmica>
                                  dynamic specialization of
                                    <id_superclasse>
<def_subclasse_dinâmica>      ::= <id_subclasse>
                                  where '{' <fórmula> '}'
```

Uma partição dinâmica da classe usuário na vídeo-locadora em função dos atributos pode ser definida abaixo para determinar a classificação de um usuário (criança, adolescente ou adulto) conforme sua idade.

```
criança where {idade<14},
adolescente where {idade>=14 and idade<=18},
adulto where {idade >18}
        dynamic specialization of usuario
```

2.6 Elementos sintáticos comuns

Nesta secão, são apresentados vários elementos sintáticos referenciados nas seções anteriores, com referência a objeto, fórmula, expressão e outros. A semântica destes elementos é a usual em linguagens de especificação. Por este motivo, somente sua sintaxe é apresentada. Detalhes encontram-se no livro de OASIS [LET 98b].

2.6.1 Referência a objeto

```
<ref_objeto>      ::= self [<espécie> [<identificação>]
<espécie>         ::= <id_classe> | <id_classe> '*' <espécie>
<identificação>   ::= '(someone)'| '(everyone)'
                      | '('<id_identificação> ','
                        '[' <lista_valor_atributo> ']' ')'
<valor_atributo>  ::= <expressão> | '_'
```

2.6.2 Fórmulas

As fórmulas são utilizadas na especificação das derivações, pré-condições, gatilhos, restrições de integridade e avaliações. Estas fórmulas serão estendidas neste trabalho para referenciar a história da ocorrência de eventos.

```
<fórmula>   ::=
            '(' <fórmula> ')' | not '(' <fórmula> ')'
            | <fórmula> and <fórmula>
            | <fórmula> or <fórmula>
            | <expressão_aritmética> <op_rel> <expressão_aritmética>
            | true | false
<op_rel>    ::= '=' | '<>'| '<' | '<=' | '>' | '>='
```

2.6.3 Atributos e serviços componentes

```
<atributo_componente>   ::= <ref_objeto>'.'<id_atributo>
                            | <ref_objeto>'.' <atributo_componente>
<serviço_componente>    ::= <ref_objeto>'.'<serviço >
                            | <ref_objeto>'.' <serviço_componente>
```

2.6.4 Expressões

As expressões em OASIS também serão estendidas com aspectos temporais.

```
<expressões>            ::= <expressão_aritmética> | '{'<fórmula>'}'
<expressão_aritmética>  ::=
                        '('<expressão_aritmética>')' | <operando>
                        |<operando> <operador> <expressão_aritmética>
<operando>              ::=<valor>|<atributo>|<atributo_componente>
                           |<parâmetro> | <função>
<operador>              ::= '+' | '-' | '*' | '/'
<valor>                 ::= <natural>|<inteiro>|<real>|<lógico>
                            |<letra>|<caracter>|<cadeia>|<data>
                            |<tempo>|<cadeia de bytes (blob)>
```

2.6.5 Domínios

Em OASIS é possível que outros domínios sejam definidos.

```
<domínio>   ::= <id_domínio> | nat | int | real | bool
                | char | string | date | time | blob
```

2.6.6 Parâmetros

```
<def_parâmetros>  ::= <id_parâmetro> ':' <domínio>
```

2.6.7 Cardinalidade

```
<cardinalidade>   ::= '('<card_min> ',' <card_max> ')'
<card_min>        ::= <natural>
<card_max>        ::= <natural> | '*'
```

2.6.8 Representação do atributo ou componente multivalorado

```
<def_representação>     ::= list '[' <def_índice> ']' | set | bag
<def_índice>            ::= <intervalo> | <lista_cadeia>
<intervalo>             ::= <int_min> '…' <int_max>
<int_min>               ::= <natural> | <letra>
<int_max>               ::= <natural> | <letra> | '*'
```

3 Estudo de caso em OASIS

A primeira seção deste capítulo apresenta a descrição de um sistema de vídeo-locadora, relatando o seu funcionamento interno em relação a seus usuários (sócios). Na segunda seção, a especificação do sistema em OASIS é definida, apresentando todas classes e os seus mecanismos de identificação.

3.1 Descrição de um sistema de vídeo-locadora

Basicamente a vídeo-locadora administra o empréstimo de fitas, a um grupo de pessoas cadastradas (sócios).
Os usuários podem emprestar no máximo 3 (três) fitas por dia. A vídeo-locadora fica aberta todos os dias.
Um empréstimo possui prazo de devolução de 1 (um) dia e o mesmo fica registrado na locadora para controle interno. Em caso de atraso na devolução, o usuário tem que pagar uma multa que dependerá do número de dias de atraso na devolução.
Existem dois tipos de fitas na vídeo-locadora, a fita comum e a fita lançamento. Para que a fita seja classificada como lançamento, esta deve ter sido adquirida na locadora dentro do prazo de seis meses. Após seis meses de cadastro, a fita deixa de ser lançamento, e tem seu preço reduzido em 20% (vinte por cento).

3.2 Especificação do sistema em OASIS

É necessário definir as seguintes classes para o sistema da vídeo-locadora:

- fita;
- usuario;
- multa;
- locação.

As duas últimas são classes complexas, isto é, são classes agregadas da classe fita e usuário.

Em OASIS isto é expresso da seguinte forma:

```
multa aggregation of
        static relational usuario towards (1,1)
        static relational fita towards (1,1)

locação aggregation of
        static relational usuario towards (1,1)
        static relational fita towards (1,1)
```

A classe multa é uma agregação das classes usuário e fita. O usuário pode ter uma multa para cada fita ou não ter multa nenhuma caso entregue a fita no prazo determinado.

A classe locação, é uma agregação das classes usuário e fita. Isto significa que quando um usuário empresta uma fita na locadora, é registrado o empréstimo na classe fita (para lançar a saída da fita) e o empréstimo na classe usuário (para determinar qual usúario a retirou da vídeo-locadora).

A seguir é apresentado o esquema conceitual da vídeo-locadora, onde são definidas as classes necessárias para especificação do sistema em OASIS.

3.2.1 Classe usuário

Esta classe é definida para cadastrar os usuários (sócios) da vídeo-locadora.

```
conceptual schema locadora_video
-------------------------------------------------
class usuario

identification
        n_usuario : (n_usuario);

constant attributes
        n_usuario : string;
        nome_usu : string towards (1,1);

variable attributes
        endereço : string;
        telefone: string towards (0,*) list[0..*];
        qtde_fitas: int (0);

events
        insere_usu new;
        remove_usu destroy;
        modifica (endereço: string , telefone:string);
        emprestar ;
        devolver ;

valuations
        [modifica (ende,fone)] endereço:= ende, telefone:= fone,
        [emprestar] qtde_fitas:= qtde_fitas +1;
        [devolver] qtde_fitas:= qtde_fitas -1;

preconditions
        remove_usu if {qtde_fitas = 0};
        emprestar if {qtde_fitas <=3}

end class
```

Os atributos constantes estão declaradas abaixo:

- “n_usuário”: o número de identificação do usuário;
- “nome_usu”: o nome do usuário. Esse atributo poderia ser declarado como atributo varíavel para representar por exemplo, o caso de uma mulher solteira que ao se casar tem adicionado ao seu nome o sobrenome do seu marido.

Os atributos variáveis são os que seguem:

- “endereço”: endereço do usuário;
- “telefone”: telefone do usuário. O usuário pode não ter telefone ou ter mais que um;
- “qtde_fitas”: a quantidade de fitas que o usuário emprestou. Quando o usuário é cadastrado a “qtde_fitas” é zero.

Na especificação da classe estão declarados eventos que podem ocorrer sobre um usuário da vídeo-locadora. São eles:

- “insere_usu” é o evento de cadastro de um novo usúario na vídeo-locadora;
- “remove_usu” é o evento de exclusão de um usuário na vídeo-locadora;

- “modifica (endereço:string, telefone: string)” é o evento que permite alterar o endereço e o telefone de um determinado usuário;
- “emprestar” e “devolver” são eventos que permitem que o usuário empreste ou devolva uma ou mais fitas na vídeo-locadora.

Os atributos variáveis modificam seus valores de acordo com o estabelecido nas avaliações (*valuations*). Na classe usuário as avaliações são as que seguem:

- na ocorrência do evento “modifica”, o atributo “endereço” recebe um novo endereço (“ende”) e o atributo “telefone” recebe um novo número de telefone (“fone”);
- quando o usuário da vídeo-locadora empresta ou devolve uma fita (ocorrência do evento “emprestar” ou “devolver”), é executada uma avaliação que aumenta ou diminui respectivamente em uma unidade, o atributo “qtde_fitas”.

Para a ocorrência dos eventos “emprestar” e “remove_usu”, algumas pré-condições (*preconditions*) devem ser satisfeitas. São elas:

- para que o usuário empreste uma fita (evento “emprestar”), a quantidade de fitas em seu poder não pode ser maior que três;
- para que um usuário deixe de ser sócio da vídeo-locadora (evento “remove_usu”) ele não pode ter nenhuma fita em seu poder.

3.2.2 Classe fita

Esta classe é definida para cadastrar as fitas na vídeo-locadora.

```
class fita
identification
        n_fita : (n_fita);

constant attributes
        n_fita : string;
        nome_autor:  string;
        titulo: string;
        assunto: string;
        descrição: string;
        data_compra: date;

variable attributes
        localização : string;
        disponível: bool (false) towards (1,1);
        lançamento: bool (true);
        preço: nat;

derived attributes
        total: nat (0);

derivations
        total:= preço + multa.valor_multa;

constraints
        {n_fita <> ´´}

events
        insere_fita new;
        remove_fita destroy;
        modifica_lançamento;
        alterar_preço(npreço);
        emprestar;
```

```
        devolver;

trigger
        ::modifica_lançamento when
                {month(currenttime) – month (data_compra) > 6 and lançamento:= true}

valuations
        [modifica_lançamento] lançamento:= false, npreço:=preço * 0,80;
        [alterar_preço(npreço)] preço:=npreço;
        [emprestar] disponível:= false;
        [devolver] disponível:= true; total:=0;

preconditions
        remove_fita if {disponível :=true};
        emprestar if {disponível :=true};
        devolver if {disponível :=false};

end class
```

A fita é identificada por um número (“n_fita”). Os atributos constantes caracterizam a fita na vídeo-locadora. São eles:

- “nome_autor”: nome do autor ;
- “titulo”: título do filme;
- “assunto”: assunto a que se refere o filme;
- “descrição”: descrição breve do filme e;
- “data_compra”: data de aquisição da fita pela vídeo-locadora.

Os atributos variáveis são os seguintes:

- “localização”: localização da fita na prateleira;
- “disponível”: define se a fita está disponível para empréstimo na vídeo-locadora (o resultado é do tipo lógico);
- “lançamento”: verifica se a fita é do tipo lançamento na vídeo-locadora (o resultado é do tipo lógico);
- “preço”: determina o valor da fita para empréstimo.

Esta classe apresenta o atributo derivado “total” para calcular o valor do empréstimo da fita na vídeo-locadora.

A derivação especifica que “total” é obtido pelo valor da fita (“preço”) mais o valor da multa (“valor_multa”) caso o usuário entregue a fita com atraso.

A restrição de integridade especifica que

Os eventos que podem ocorrer sobre uma fita na vídeo-locadora são declarados abaixo:

- “insere_fita” e “remove_fita” são os eventos para cadastro e exclusão de uma fita na vídeo-locadora;
- “modifica_lançamento” é o evento que faz com que a fita deixe de ser lançamento na vídeo-locadora;
- “altera_preço(nv)” é o evento que altera o preço do empréstimo da fita;
- “emprestar” e “devolver” são eventos que registram o empréstimo ou devolução de uma ou mais fitas na vídeo-locadora.

Um gatilho (*trigger*) é uma ação que ocorre quando um objeto atinge um estado que é especificado por uma condição. Na classe fita, o gatilho especifica que o evento

“modifica_lançamento” irá ocorrer quando a data da compra da fita for maior que 6 (seis) meses e atributo “lançamento” for verdadeiro.

As avaliações para a classe fita são as seguintes:

- na ocorrência do evento “modifica_lançamento”, o atributo “lançamento” passa a ter o valor falso e o atributo “npreço” passa a ter o valor da fita reduzido em 20% (vinte por cento);
- na ocorrência do evento “altera_preço”, o atributo “preço” recebe um novo valor (“npreço”). Quando a fita deixa de ser lançamento o valor é estipulado (“npreço:= preço*0,80”). Existe ainda, a possibilidade do proprietário da vídeo-locadora aumentar o preço da locação;
- quando a fita é emprestada (evento “emprestar”) ou devolvida (evento “devolver”) na vídeo-locadora, o atributo “disponível” recebe respectivamente, o valor falso ou verdadeiro. No caso da fita ser devolvida, o atributo “total” recebe o valor zero, indicando que o usuário pagou a locação.

As pré-condições definidas para esta classe são as que seguem:

- para a ocorrência do evento “remover”, a fita em questão não pode estar emprestada, isto é, o atributo “disponível” tem que ser verdadeiro;
- para a ocorrência dos eventos “emprestar” e “devolver”, o atributo “disponível” tem que ser respectivamente verdadeiro e falso.

3.2.3 Classe locação

Esta classe, controla o empréstimo das fitas para os usuários na vídeo-locadora.

```
class locação
identification
        ref : (usuario.n_usuario: string, fita.n_fita:string, data_loc:date);

constant attributes
        data_loc:date;

variable attributes
        data_devolução: date;
        multado:bool (false);

events
        emprestar new
                calling to members
                        fita.emprestar, usuario.emprestar;

        devolver (currenttime) destroy
                calling to members
                        fita.devolver, usuario.devolver;
trigger
        multa::insere_multa(n_fita, n_usuario,currenttime) when
            (day(currenttime) – day(data_loc) > 1) and (multado:=true)

valuations
        [devolver(currenttime) ] data_devolução:= currenttime;
        [multa::insere_multa(n_fita, n_usuario, currenttime)] multado:=true;
```

```
end class
```

A identificação deste classe é feita através do número do usuário, do número da fita e da data de locação.

Os atributos variáveis são os seguintes:

- "data_devolução": registra quando o usuário devolveu a fita;
- "multado": controla se o usuário foi multado ou não (o resultado é do tipo lógico).

Como esta classe é agregação da classe fita e da classe usuário, na definição dos eventos "emprestar" e "devolver" aparece a etiqueta *calling to members*, representando que estes eventos são implicados. Isto significa que a ocorrência de um destes eventos nesta classe (locação), implica necessariamente na ocorrência deles em seus objetos componentes (classe fita e classe usuário).

O gatilho desta classe é para inserir uma multa na classe multa (evento "insere_multa"), quando a data do empréstimo da fita ultrapassar 1 (um) dia e o atributo "multado" for verdadeiro. *Day* é uma função que recebe uma data e retorna o dia correspondente. *Currenttime* é o atributo implicitamente definido em todas as classes que retorna o data do sistema.

As avaliações para esta classe são as definidas abaixo:

- ao ser devolvido uma fita (evento "devolver") o atributo "data_devolução" recebe a data do sistema (*currenttime*);
- ao ser disparado o gatilho para inserir uma multa na classe multa, o atributo "multado" passa ter o valor verdadeiro.

3.2.4 Classe multa

Esta classe controla os usuários que estão em atraso na devolução das fitas.

```
class multa;
identification
        ref : (usuario.n_usuario : string, fita.n_fita:string, data_mul:date);

constant attributes
        data_mul:date;

variable attributes
        status_pagto: bool (false);
        valor_dia: nat;

derived attributes
        valor_multa: nat;

derivations
        valor_multa:= {(day(currenttime) – day(data_mul)) * valor_dia};

events
        insere_multa new;
        remove_multa destroy;
        altera_status_pagto;
        altera_valor_dia (nv);
        altera_valor_multa(0);

trigger
```

```
        ::altera_status (true) when {valor_multa:=0};

valuations
        [altera_valor_dia(nv)] valor_dia:=nv;
        [altera_valor_multa(0)] valor_multa:=0;

preconditions
        remove_multa if {status_pagto :=true and fita.total:=0};

end class
```

Esta classe é identificada através do número do usuário, do número da fita e da data da multa.

Os atributos variáveis são os seguintes:

- "status de pagto": verifica se o usuário pagou ou não a multa;
- "valor_dia": valor da multa por dia.

Esta classe apresenta o atributo derivado "valor_multa".

Para especificar o atributo derivado, a classe multa apresenta a derivação (*derivations*), que calcula o valor da multa a ser pago pelo usuário ("qtde_dias * valor_dia").

Os eventos definidos para esta classe são declarados abaixo:

- "insere_multa" e "remove_multa" são os eventos de inserir e remover uma multa para o usuário;
- "altera_status_pagto" é o evento que registra se o usuário pagou ou não a multa;
- "altera_valor_dia(nv)" é o evento que altera o valor da multa por dia;
- "altera_valor_multa(0)" é o evento que altera o valor da multa para zero quando esta for paga.

O gatilho estabelece que quando o valor da multa for zero (atributo "valor_multa) o evento "altera_status_pagto" deve ocorrer alterando o valor do atributo "status_pagto" para verdadeiro.

Abaixo, estão declaradas as avaliações para esta classe:

- ao ser alterado o valor da multa diária (evento "altera_valor_dia"), o atributo "valor_dia" recebe um novo valor ("nv");
- ao ser paga uma multa (evento "altera_valor_multa"), o atributo "valor_multa" recebe zero;

A pré-condição determina que para remover uma multa (evento "remove_multa"), é necessário que o usuário a tenha pago, isto é, o atributo "status_pagto" tem que ter o valor verdadeiro e o valor do atributo "total" tem que ser zero.

4 Dimensão Temporal

Os modelos de dados tradicionais apresentam duas dimensões:

- as instâncias dos dados (linhas de uma tabela);
- os atributos de cada instância (colunas desta tabela).

Cada atributo de uma instância apresenta um só valor. Se for feita uma alteração deste valor, o anterior é perdido. Por exemplo, um funcionário que tinha um salário de R$ 400,00 (quatrocentos reais), e teve um aumento de 20%, terá registrado no atributo "salário" somente o valor atual de R$480,00 (quatrocentos e oitenta reais).

Os modelos temporais acrescentam mais uma dimensão aos modelos tradicionais:

- a dimensão temporal.

Esta dimensão associa uma informação temporal a cada valor. Caso o valor de um atributo seja alterado, o valor anterior não é removido do banco de dados – o novo valor alterado é acrescentado, associado a alguma informação que defina, por exemplo, seu tempo inicial de validade. Todos os valores definidos ficam armazenados no banco de dados. Assim, é possível acessar toda a história dos atributos, sendo possível analisar sua evolução temporal.

Na primeira seção deste capítulo são definidos os conceitos relativos à dimensão temporal. Na segunda seção são descritos alguns modelos OO que apresentam extensão temporal.

4.1 Conceitos relativos à dimensão temporal

Os conceitos de banco de dados temporais abaixo definidos, utilizados pelo modelo proposto nesta dissertação, são extraídos do glossário de conceitos de banco de dados temporais [JEN 98] e de [EDE 98].

4.1.1 Eixo temporal

A forma que se tem mostrado mais adequada para tratar a dimensão temporal em sistemas de informação é assumi-la como uma seqüência discreta, linear e finita de pontos consecutivos do tempo. Esta seqüência também recebe a denominação de eixo temporal e é representada pela constante **T**.

Existem duas formas diferentes de representação temporal: tempo contínuo e tempo discreto. O tempo é contínuo por natureza. Entretanto, sem muita perda de generalidade, pode-se considerar que o tempo seja discreto, possibilitando a implementação do modelo.

Modelos de dados que suportam uma noção discreta de variação temporal são baseadas em uma linha do tempo composta de uma seqüência de intervalos temporais consecutivos, que não podem ser decompostos, de idêntica duração. Estes intervalos são denominados *chronons*. A duração de um *chronon* não é necessariamente fixada no modelo de dados, podendo ser definida em implementações particulares.

Assumir que o tempo flui linearmente implica, em uma total ordenação entre quaisquer dois pontos no tempo. Em alguns casos pode ser considerado tempo ramificado (*branching time*). Para estes, a restrição linear é abandonada permitindo a possibilidade de dois pontos diferentes serem sucessores (figura 3.1) ou antecessores (figura 3.2) imediatos de um mesmo ponto. Uma ramificação no futuro implica que podem ser considerados múltiplos possíveis desenvolvimentos futuros do domínio (por exemplo, diferentes hipóteses da história futura), enquanto que uma ramificação no passado admite múltiplas histórias passadas do domínio em questão. A combinação de "passado linear, futuro ramificado" trabalha com uma só história passada e admite múltiplas histórias futuras, representando desta maneira a realidade atual de uma forma bastante fiel.

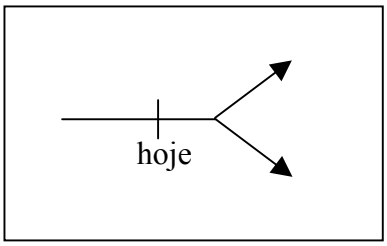

FIGURA 3.1- Ramificação no futuro

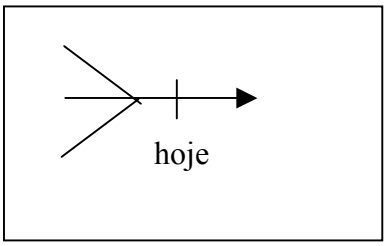

FIGURA 3.2- Ramificação no passado

Uma última opção de ordenação temporal é considerar o tempo circular. Esta forma pode ser utilizada para modelar eventos e processos recorrentes (figura 3.3).

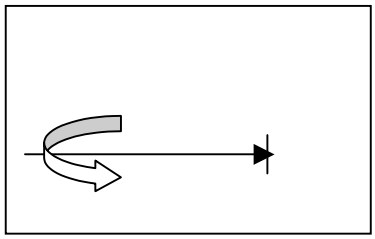

FIGURA 3.3- Tempo circular

A maior parte dos modelos temporais se baseia no tempo linearmente ordenado. A ordenação total do tempo pode ser definida com mais precisão através da teoria dos conjuntos, conforme mostrado a seguir [ANT 97].

Seja T o conjunto não vazio de todos os pontos do tempo. Por definição T, é um conjunto totalmente ordenado pela relação *Before* (antes), a qual satisfaz à seguinte condição:

$$\forall \text{ ta tb} : \text{ta tb} \in \mathbf{T} \wedge \text{ta} \neq \text{tb} \rightarrow (\text{ta } \textit{Before} \text{ tb} \vee \text{tb } \textit{Before} \text{ ta})$$

Para que a relação *Before* seja uma relação de ordem estrita total é necessário que possua as seguintes propriedades:

Irreflexibilidade:

$$\forall \text{ t} : \text{t} \in \mathbf{T} \rightarrow \neg (\text{t } \textit{Before} \text{ t})$$

Transitividade:

$$\forall \text{ ta tb tc} : \text{ta tb tc} \in \mathbf{T} \wedge \text{ta } \textit{Before} \text{ tb} \wedge \text{tb } \textit{Before} \text{ tc} \rightarrow \text{ta } \textit{Before} \text{ tc}$$

Assimetrias:

$$\forall \text{ ta tb : ta tb} \in \mathbf{T} \wedge \text{ta } \textit{Before} \text{ tb} \rightarrow \neg \text{ (tb } \textit{Before} \text{ ta)}$$

A relação *Before* é equivalente a relação "<" utilizada no âmbito dos números inteiros, sendo este operador muitas vezes utilizado para representar a ordem temporal.

A extensão temporal apresentada neste trabalho, apresenta a dimensão temporal como um eixo de pontos discretos, composto por uma seqüência de intervalos temporais consecutivos (ordem linear).

4.1.2 Tempo absoluto e tempo relativo

Tempo absoluto consiste de uma informação temporal que define um tempo específico, definido com uma granularidade determinada, associado a um fato. Exemplo: data de nascimento de uma pessoa.

O tempo é relativo quando sua validade é relacionada à validade de outro fato, ou ao momento atual. Exemplo: salário de uma pessoa, inauguração de uma loja.

No trabalho proposto, são utilizados o tempo absoluto e o tempo relativo.

4.1.3 Variação temporal

Considerando variação temporal discreta, a definição de informações ao longo do tempo, sob ponto de vista de sua validade, pode ser feita das seguintes formas [EDE 94b, 98]:

- variação ponto a ponto: o valor definido vale somente no ponto temporal onde foi definido. Não existe valor válido nos pontos para os quais não foram definidos valores (figura 3.4);
- variação por escada: o valor fica constante desde o ponto em que foi definido até o instante em que outro valor seja definido. Corresponde, geralmente, à definição de valores em conseqüência da ocorrência de eventos. Também é conhecida como variação por eventos (figura 3.5);
- variação temporal definida por uma função: existe uma função que define os valores e que permite a interpolação para obter os valores nos pontos não definidos. Esta função de interpolação pode ser definida pelo usuário ou incluída na modelagem conceitual (figura 3.6).

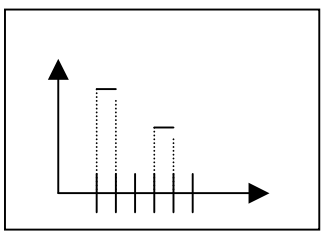

FIGURA 3.4- Ponto a ponto

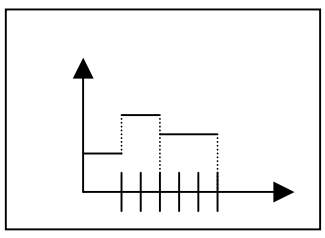

FIGURA 3.5- Em escada

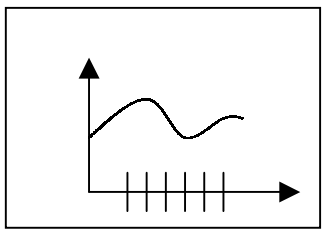

FIGURA 3.6- Por função

No modelo apresentado neste trabalho, a dimensão temporal é apresentada como um eixo de pontos discretos. São utlizadas duas formas de variação temporal:

- para representação dos atributos, a definição das informações é feita sob a forma de variação por escada. Por exemplo, o valor do atributo temporal "salário" fica válido até que ocorra um evento para alterá-lo;

- os eventos apresentam a dimensão temporal sob a forma de variação ponto a ponto.

Por exemplo, ao ocorrer um evento qualquer, o que fica válido é o ponto temporal onde foi definido este evento.

4.1.4 Granularidade temporal

Quando se considera a dimensão temporal como sendo um eixo discreto assume-se que cada ponto do tempo pertencente a este eixo é atômico e dura exatamente um *chronon*. O *chronon* define a granularidade da dimensão temporal. Dependendo da aplicação, às vezes é necessário considerar diferentes granularidades (minutos, dias, anos) para permitir melhor representação da realidade. Por exemplo, em um determinado segmento modelado a granularidade pode ser diária (o *chronon* equivale a um dia) e em um outro segmento a granularidade pode ser mensal. Para efeito do presente trabalho a duração de um *chronon* apresenta apenas uma granularidade que fica livre a ser escolhida no momento da implementação do modelo de dados.

4.1.5 Elementos primitivos de representação temporal

a) Instante no tempo

O conceito de instante, representando um particular ponto no tempo, depende da forma de variação temporal considerada. Quando é considerado tempo contínuo, um instante é o ponto no tempo de duração infinitesimal. Neste caso os instantes são isomórficos com números reais, o que significa que entre dois pontos do tempo sempre existe um outro ponto no tempo.

Quando, no entanto, é considerada a variação temporal discreta, um instante é representado por um dos *chronons* da linha do tempo suportada pelo modelo. Na variação discreta, os instantes são isomórficos aos números inteiros ou a um subconjunto destes. Assim, entre dois pontos do tempo consecutivos não existe outro ponto do tempo. Um *chronon*, que é a menor duração de tempo suportada por um SGBD temporal, pertence à representação discreta de tempo.

Considerando a ordem de variação temporal linear, temos a existência de um instante especial, correspondente ao instante atual (*now*), o qual se move constantemente ao longo do eixo do tempo. Este ponto define o que é considerado como passado e como futuro. A figura 3.6 apresenta o instante atual.

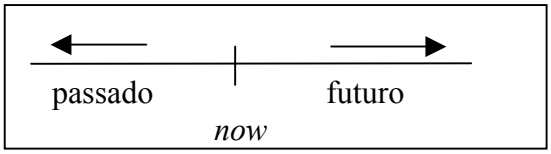

FIGURA 3.7- Instante no tempo

Os eventos temporais propostos neste trabalho apresentam rótulos temporais representados por um instante no tempo.

b) Intervalo temporal

Um intervalo temporal é caracterizado pelo tempo decorrido entre dois instantes – um subconjunto de pontos do eixo temporal. Depende também da forma de representação temporal definida no modelo. Quando é considerado tempo contínuo, o intervalo consiste de infinitos instantes de tempo. Na variação discreta um intervalo é representado por um conjunto finito de *chronons* consecutivos.

O intervalo temporal é representado pelos dois instantes que o delimitam. Dependendo da pertinência ou não dos instantes limites ao intervalo, este pode ser aberto (os limites não pertencem ao intervalo), semiaberto (um dos limites pertence ao intervalo) ou fechado (ambos os limites pertencem ao intervalo). Quando um dos limites é representado pelo instante atual (*now*) temos a representação de um intervalo particular cujo tamanho varia com a passagem do tempo.

Um intervalo temporal é representado por [t1,t2], onde t1 é o primeiro ponto do intervalo (limite inferior) e t2 é o último (limite superior). O próprio eixo temporal T pode ser considerado um intervalo de tempo, identificado pela expressão [«, »], onde os símbolos "«" e "»" significam respectivamente o início e o fim da contagem do tempo. Para qualquer intervalo temporal, uma das duas fórmulas a seguir deve ser verdadeira:

$$t1 < t2 \text{ ou } t1 = t2.$$

A segunda representa um intervalo cuja duração é exatamente um *chronon*.

Um intervalo temporal também é totalmente ordenado pela relação *Before*, sendo possível, através dos operadores *First* e *Last* extrair-lhe o primeiro e o último ponto de tempo. É o que se passa a demonstrar. A figura 3.8 mostra a representação do intervalo temporal.

Seja I, um intervalo de tempo e $I \subseteq T$, então:

First (I) é o elemento $t \in I$ tal que, $\forall t' \in I$: t *Before* t' $\vee$ t = t'

Last (I) é o elemento $t \in I$ tal que, $\forall t' \in I$: t' *Before* t $\vee$ t' = t

Para que um conjunto de pontos do tempo seja realmente considerado um intervalo, é necessário que todos os pontos sejam consecutivos, isto é, não pode haver qualquer lacuna entre eles . Esta condição é formalmente representada pela expressão abaixo.

Seja $I \subseteq T$, um intervalo, então:

$\forall$ ta $\in$ I: ta $\neq$ *Last* (I) $\rightarrow$

$\exists$ tb $\in$ I: (ta *Before* tb $\wedge$ $\neg\exists$ tc $\in$ T: ta *Before* tc $\wedge$ tc *Before* tb)

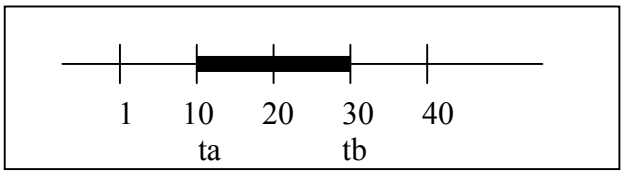

FIGURA 3.8- Intervalo temporal

c) Elemento temporal

É uma união finita de intervalos de tempo. Estes intervalos podem ser disjuntos, o que realmente o diferencia dos demais e enriquece o seu poder de expressão . O elemento temporal é fechado para as operações de união, interseção e complemento da teoria dos conjuntos, isto é, qualquer destas operações sobre um elemento temporal produz um novo elemento temporal. Como estas operações encontram contrapartida nos operadores booleanos *or*, *and* e *not*, isto produz uma substancial simplificação na habilidade do usuário de expressar consultas temporais. Tendo em vista que todos os intervalos temporais são subconjuntos do eixo temporal T, um elemento temporal, sendo composto por diversos intervalos temporais, também o é. Tanto um intervalo temporal como um instante temporal ([t, t']) são elementos temporais (figura 3.9).

Em termos de modelagem, o elemento temporal se mostra superior ao uso da primitiva intervalo de tempo pois, quando os intervalos são usados como rótulos temporais, os objetos são fragmentados em várias tuplas, uma para cada intervalo. Outro aspecto importante desta primitiva temporal é que possibilita a representação da "reencarnação" de objetos com facilidade.

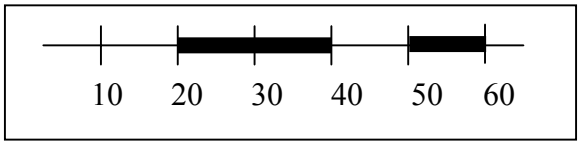

FIGURA 3.9- Elemento temporal

A representação dos atributos variáveis temporais neste trabalho é através de rótulos temporais na forma de elementos temporais.

d) Duração temporal

A representação de uma duração temporal pode também ser considerada como primitiva.

Durações temporais podem ser basicamente de dois tipos, dependendo do contexto em que são definidas: fixas e variáveis. Uma duração fixa independe do contexto de sua definição (a hora tem 60 minutos). Já a duração variável depende do contexto (duração do mês).

4.1.6 Limites no tempo

O conceito de limites no tempo pode variar dependendo da representação temporal utilizada.

Quando considerados somente pontos no tempo, os limites do tempo referem-se a considerar ou não o tempo como infinito. O conceito de tempo infinito consiste em considerar que todo ponto no tempo apresente sempre um sucessor e um antecessor. Em modelos orientados a objetos, como o proposto neste trabalho, este conceito fica limitado ao tempo de vida de um objeto. No caso das teorias baseadas em intervalos, os limites de tempo referem-se geralmente à pertinência ou não dos pontos limites ao intervalo, definindo se os intervalos são abertos ou fechados em uma ou em ambas extremidades.

4.1.7 Representação temporal explícita e implícita

A definição de tempo pode ser feita de forma explícita, através, por exemplo, da associação de um valor temporal a uma informação na forma de um rótulo temporal (*timestamping*), ou de forma implícita através da utilização de um linguagem de lógica temporal.

A associação explícita de tempo às informações consiste em associar a cada valor atribuído a um atributo, o valor correspondente a sua primitiva temporal. A representação temporal implícita é feita através da manipulação de conhecimentos sobre a ocorrência de eventos ou do relacionamento de intervalos de tempo.

Neste trabalho é utilizada uma definição de tempo de forma explícita através de rótulos temporais.

4.1.8 Tempo de transação e tempo de validade

Três diferentes conceitos temporais podem ser identificados em aplicações de banco de dados:

- tempo de transação, tempo no qual o fato é registrado no banco de dados;
- tempo de validade, tempo em que o valor é valido na realidade modelada;
- tempo definido pelo usuário, consistindo de propriedades temporais definidas explicitamente pelos usuários em um domínio temporal e manipuladas pelos programas de aplicação.

Estes tempos são ortogonais, podendo ser tratados separadamente ou em conjunto. O tempo de transação é suprido automaticamente pelo sistema gerenciador de banco de dados, enquanto que o tempo de validade é fornecido pelo usuário.

O tempo de validade pode ser representado de formas distintas, dependendo do elemento temporal básico utilizado no modelo. Quando for utilizado o elemento temporal ponto no tempo, o tempo de validade pode ser representado através de:

- um ponto no tempo indicando o início da validade, permanecendo o valor válido até que inicie o tempo de validade de outro valor;
- dois pontos no tempo, o primeiro indicando o início da validade e o segundo, o final da validade.

Nos modelos que utilizam intervalo temporal como elemento temporal básico, o tempo de validade é definido através do intervalo de validade do valor.

No trabalho proposto, os eventos temporais utilizam o tempo de transação porque o que interessa é o instante em que o evento ocorreu. Já os atributos temporais utilizam o tempo de validade para melhor modelar a realidade.

4.2 Modelos OO temporais encontrados na literatura

Diversos modelos de dados temporais foram propostos nos últimos tempos. A grande maioria dos modelos é constituído por extensões temporais realizadas sobre modelos de dados já existentes. O modelos de dados podem ser relacionais, ER (entidade-relacionamento) ou OO (orientados a objetos).

Neste trabalho é apresentado apenas modelos OO porque o modelo proposto para extensão também é OO.

O tratamento de tempo em modelos de dados orientados a objetos está presente em vários modelos recentemente apresentados [TAN 93]. Entretanto, a representação de aspectos temporais em bancos de dados orientados a objetos tem sido feita nestes modelos de uma forma bastante limitada. Em sua grande maioria os aspectos temporais são tratados da mesma forma como o foram nos modelos relacionais.

A utilização de um modelo temporal orientado a objetos tem por finalidade a representação de todos os estados assumidos pelo objeto durante sua existência. Para que isto seja possível é necessário que o modelo permita a representação do comportamento que o objeto deve apresentar durante sua evolução.

A identificação dos aspectos abaixo é fundamental para a utilização de um modelo temporal:

- tipo de rótulo temporal;

- tempo de validade / tempo de transação;
- possibilidade de definição de propriedades temporais e não temporais;
- representação de **eventos** temporais.

Nesta seção é apresentada uma visão geral de alguns modelos OO que foram estendidos com dimensão temporal. Em seguida é feito uma comparação destes modelos com o modelo proposto neste trabalho seguindo os aspectos citados anteriormente.

4.2.1 OODAPLEX

O modelo OODAPLEX [WUU 93] é uma extensão do modelo de dados funcional DAPLEX [SHI 81], incorporando a este o paradigma de orientação a objetos.

Neste modelo, todo objeto apresenta uma identidade única, imutável durante toda sua existência, independente de suas propriedades e do seu comportamento. Todos os objetos são abstratos, consistindo de um interface (o *tipo* do objeto) e de uma implementação (a *representação* do objeto). As propriedades dos objetos, os relacionamentos entre objetos e as operações sobre os objetos são todas modeladas de maneira uniforme através de funções aplicadas aos objetos. Uma função apresenta uma interface (composta de seu nome e uma assinatura que define os argumentos de entrada e saída da função) e uma implementação (corpo da função). Esta função pode ter zero ou mais argumentos de entrada e/ou de saída.

Objetos que apresentam propriedades e comportamento similares são agrupados em *tipos*. O tipo especifica o conjunto de funções que podem ser aplicadas às instâncias do tipo. Tipos complexos, tais como, conjuntos, tuplas e multiconjuntos, podem ser recursivamente construídos a partir dos tipos primitivos e dos definidos pelo usuário. Podem ser definidos subtipos formando hierarquias com herança. Nos subtipos podem ser definidas novas funções e, funções herdadas podem ser redefinidas (polimorfismo).

OODAPLEX suporta encapsulamento, uma vez que os objetos somente podem ser manipulados através das funções definidas para seus tipos. O modelo apresenta um conjunto de tipos pré-definidos, tais como *integer*, *real*, *boolean* e *string*. Além disto, os usuários podem definir tipos de objetos abstratos.

As propriedades que podem variar com o tempo são modeladas como funções que retornam uma outra função, a qual mapeia elementos temporais a valores da propriedade.

Objetos temporais são criados, modificados e removidos em tempos diferentes. A associação de tempo no modelo é feita tanto nos atributos quanto nos objetos como um todo.

A representação de aspectos temporais é feita através de rótulos temporais do tipo elemento temporal. Quanto ao tempo de validade ou de transação, este modelo utiliza o bitemporal (transação e validade).

4.2.2 TF-ORM

O desenvolvimento do modelo TF-ORM [EDE 93, 94 a] iniciou com o modelo ORM (*Object with Roles Model*) [PER 90], no qual era introduzido o conceito de papéis para representar separadamente diferentes comportamentos de um objeto. O modelo ORM foi posteriormente estendido dando origem ao modelo F-ORM (*Functionality in Object with Roles Model*) [DEA 91a,b,c,] com o objetivo de ser utilizado para especificação

das funcionalidades de sistemas de informações de escritórios. Uma terceira extensão deu origem ao TF-ORM (*Temporal Functionality in Object with Roles Model*) incorporando ao modelo anterior, a capacidade de representação de aspectos temporais e dos efeitos de decisões humanas no trabalho desenvolvido.

No modelo TF-ORM as classes são decompostas em três tipos distintos, pré-definidos:

- classes agente: representam as pessoas, incluindo a possibilidade de representação de trabalhos não estruturados executados por eventuais tomadas de decisão;
- classes recurso: define a estrutura de um recurso em termos dos papéis onde este recurso possa apresentar durante seu ciclo de vida;
- classes processo: integram as classes recurso, permitindo a descrição do trabalho realizado entre os agentes e recursos envolvidos.

Uma classe é definida por um nome único em toda a especificação e por um conjunto de papéis, onde cada qual representa um comportamento diferente deste objeto.

TF-ORM apresenta uma classe pré-definida denominada *OBJECT*, que desempenha o papel de superclasse para todas as demais classes definidas. As propriedades dessa superclasse são herdadas por todas as suas subclasses, sejam elas do tipo agente, recurso ou processo, não podendo ser redefinidas.

Cada classe apresenta um papel básico, onde são definidos o identificador da instância da classe, o início da vida da instância da classe, seus períodos de validade e o momento em o objeto que deixa de existir.

O tempo é acrescentado tanto ao nível de objetos (para definir criação, suspensões, término das instâncias) como ao nível das propriedades.

As propriedades são descrições abstratas das características de um objeto (seus atributos). Um dos principais aspectos do modelo é que ele registra implicitamente a evolução das propriedades, cujo conteúdo varia no tempo. Dois tipos de propriedades são definidas separadamente: propriedades estáticas, as quais mantém sempre o mesmo valor durante todo o tempo de existência de um objeto, e propriedades dinâmicas, que podem variar com a passagem do tempo. As propriedades dinâmicas possuem rótulo bitemporal (tempo de transação e um tempo de validade associados).

O comportamento dos objetos de uma classe é governado através de regras. Dois tipos de regras são utilizadas em TF-ORM: regras de transição de estados e regras de integridade. As regras de transição de estados descrevem o comportamento de um objeto ao desempenhar um determinado papel. As regras de integridade não causam transições de estado, mas restringem as possíveis evoluções dos objetos uma vez que não permite transições que gerem conteúdos do banco de dados que não satisfazem as condições.

No modelo TF-ORM o tempo é representado de forma linear e ordenado, variando de forma discreta. O elemento temporal primitivo adotado para representação de aspectos temporais é o ponto no tempo.

4.2.3 TDM

O modelo TDM (*Temporal Data Model*) [SEG 88, 93] representa um enfoque de modelagem independente de qualquer dos modelos tradicionais conhecidos. O modelo não se preocupa em modelar os eventos temporais, mas em tratar as propriedades dos objetos através da estrutura denominada seqüência temporal (*Time Sequence*), que é,

uma coleção de valores de dados assumidos por uma instância de uma entidade em diversos momentos no tempo.

Uma classe consiste de uma coleção de objetos que apresentam os mesmos atributos. Todo objeto apresenta um identificador único, denominado *surrogate*. Classes compostas podem ser definidas como resultado da agregação de outras classes. Os objetos de classes compostas necessitam de mais de um *surrogate* para a sua identificação.

No modelo TDM são definidas coleções de seqüências temporais, considerando que todas as seqüências temporais dos objetos são pertencentes a uma mesma classe, onde uma coleção de seqüências temporais é utilizada no modelo como um construtor para representar valores temporais associados a uma classe. Estas coleções podem ser simples ou complexas, dependendo do tipo da classe.

As seqüências temporais apresentam as seguintes propriedades:

- granularidade temporal: granularidade temporal ordinal ou de calendário;
- tempo de vida: tempo de vida associado a seqüência temporal;
- regularidade: uma seqüência temporal regular contém um valor para cada ponto no tempo do intervalo correspondente ao tempo de vida. Uma seqüência temporal irregular somente contém valores para um subconjunto dos pontos no tempo do intervalo de tempo de vida;
- tipo: os mais comuns são tipos de variação constante, contínua, discreta e definida pelo usuário.

4.2.4 OSAM*/T

OSAM*/T [SU 91, 93] é uma extensão temporal do modelo de dados OSAM* [ALA 90]. É utilizado para representação de informações em bases de conhecimento.

O modelo expressa as propriedades comportamentais dos objetos em termos de métodos e regras de conhecimento, e suas propriedades estruturais em termos de várias associações estruturais com outros objetos.

Um objeto representa uma entidade (representação abstrata, evento, processo, função). Todo objeto e toda classe têm um identificador único. As classes de objeto apresentam as mesmas características estruturais e o mesmo comportamento. Um objeto do banco de dados pode pertencer a mais de uma classe.

A instância é o par dado pelo identificador do objeto e pelo identificador da classe.

As propriedades têm duas definições:

- comportamentais de uma classe: são as operações definidas pelo sistema ou pelo usuário;
- estruturais de uma classe: são os dados descritivos (variáveis da instância) que definem o estado dos objetos;
- e as associações de dados que especificam os relacionamentos entre seus objetos e os objetos da classe relacionadas (agregação, generalização, interação, composição e produto cruzado).

A representação temporal é feita através da associação de rótulos temporais às informações, na forma de atributos. São rótulos temporais (*timestamps*) do tipo intervalo, associados aos objetos como um todo. As associações também são tratadas como objetos.

OSAM*/T utiliza o conceito de tempo discreto para a representação temporal, sendo o tempo considerado isomórfico aos números naturais e representados através de seqüências de tempo. Uma seqüência de tempo consiste de uma série ordenada de pontos no tempo contínuos.

A noção de tempo de validade é utilizada para registrar a evolução das instâncias de objeto.

4.2.5 Comparação entre os modelos OO com aspectos temporais

Analisando os modelos apresentados anteriormente e os aspectos que cada modelo deveria apresentar (tabela 4.1), é proposto neste trabalho um modelo OO com aspectos temporais, procurando satisfazer todos os aspectos anteriormente citados.

TABELA 4.1- Modelos OO com aspectos temporais

Aspectos/ Modelos	OODAPLEX	TDM	TF-ORM	OASM*/T
Rótulos temporais	elemento temporal	elemento temporal	ponto no tempo	intervalo temporal
Tempo de transação / Tempo de validade	bitemporal	bitemporal	bitemporal	tempo de validade
Propriedade temporais e não temporais	sim	sim	sim	não (tempo no objeto)
Eventos temporais	não	não	não	não

5 Extensão de OASIS

Este capítulo apresenta a extensão temporal dos aspectos estáticos e dinâmicos de OASIS.

Na primeira seção deste capítulo, é apresentada uma simplificação para a definição de atributos em OASIS. Na segunda seção são apresentadas as alternativas de extensão de OASIS. Tendo em vista a alternativa escolhida para estender OASIS são descritos os domínios temporais na terceira seção, e a definição das classes temporais na quarta seção. Na quinta seção, as fórmulas de OASIS são estendidas para suportar o aspecto tempo. Na sexta e última seção, é apresentado o OASIS Temporal, isto é, OASIS com aspectos temporais.

5.1. Simplificação da especificação para atributos em OASIS

Em OASIS não é possível definir atributos cujo valor é um objeto. Quando for necessário, cria-se uma classe e a define como uma agregação que contém a classe que seria atributo.

No exemplo da vídeo-locadora, suponha que exista uma classe endereço com seus respectivos atributos conforme especificação a seguir.

```
class endereço

identification
        ref: (cod);

constant attributes
        cod: string;

variable attributes
        rua: string;
        numero: nat;
        cidade: string;
        estado: string;
        cep: string;
        complemento: string;
events
        insere_endereço new;
        remove_endereço destroy;
end class
```

Deseja-se especificar que um usuário pode ter um ou mais endereços (residencial, comercial e outros). Isto significa que a classe endereço é agregada da classe usuário. A especificação desta agregação em OASIS é a seguinte:

```
usuario aggregation of
        dynamic inclusive endereco
                towards (1,*) from (1,1) list [1..*]
```

Quando os objetos do componente podem modificar-se, a agregação é definida como *dynamic*.

Para tornar especificações de OASIS mais sucintas e fáceis de entender, introduz-se aqui uma simplificação da notação. Nesta simplificação, admitem-se atributos cujo valor é um objeto (não somente literais como em OASIS original).

Usando a notação simplificada, na classe usuário passa a ser definido um atributo "ende" cujo valor é um objeto (classe endereço). Esta definição é apresentada abaixo:

```
class usuario

variable attributes
        ende: endereco;
        ...
```

Essa simplificação é utilizada na extensão temporal.

5.2 Alternativas de extensão

Para estender OASIS com aspectos temporais existem duas alternativas.

Uma delas, é modificar o formalismo original OASIS. Isto significa, estender a sintaxe e alterar a semântica. OASIS em sua primeira versão [PAS 92b] foi formalizado através de Teorias de Primeira Ordem. Na segunda versão [PAS 95a], OASIS foi baseado em uma variante da Lógica Dinâmica que permite representar os operadores de obrigação, proibição e permissão usados na Lógica Deôntica [AQV 84, MEJ 88]. Em sua última versão [LET 98b], o formalismo foi mantido.

A alternativa em questão seria modificar seu formalismo básico para uma lógica temporal [MAI 91, KOW 85]. Em uma lógica temporal é possível fazer referência aos estados dados passados, presentes e futuros da aplicação.

Modelos orientados a objetos encapsulam estado e comportamento através da definição de classes. Assim a segunda alternativa é definir a extensão temporal de OASIS, utilizando o próprio modelo para a extensão. Isto significa definir classes temporais e domínios temporais utilizando a semântica de OASIS.

Neste trabalho, preferiu-se a segunda alternativa, para não modificar a semântica da linguagem.

Nas sub-seções que seguem, a extensão de OASIS para aspectos temporais é definida na forma de:
- domínios temporais;
- classes temporais e;
- extensão das fórmulas e expressões.

5.3 Domínios temporais

Em OASIS, domínios são conjuntos de valores que os atributos podem assumir.
Para tratamento do tempo, são definidos os seguintes domínios temporais:

- time_point;
- closed_time_interval;
- begin_open_time_interval;
- end_open_time_interval;
- time_interval.

OASIS admite uma seção <*equations*>, para definição das equações que definem formalmente as funções. Preferiu-se neste trabalho definir as funções apenas de maneira informal.

A seguir são apresentadas as funções que operam sobre domínios temporais.

5.3.1 Domínio "time_point"

O domínio "time_point", foi definido para representar um ponto no tempo. Os pontos no tempo são dados em segundos (*default*).

Foram definidas funções que operam com pontos no tempo, para transformá-los em minutos, horas, dias. Outras funções podem ser definidas conforme a necessidade do usuário.

```
domain time_point
functions
        in_sec (the_point:time_point) : int;
        in_year (the_point:time_point) : int;
        in_minute (the_point:time_point) : int;
        in_hour (the_point:time_point) : int;
        in_month (the_point:time_point) : int;
        in_day (the_point:time_point) : int;
        in_time_stamp (the_point:time_point) : string;
        sec_in_time_point (seconds:int): time_point;
        year_in_time_point (year:int): time_point;
        hour_in_time_point (hour:int): time_point;
        minute_in_time_point (minute:int): time_point;
        month_in_time_point (month:int): time_point;
        day_in_time_point (day:int): time_point;
        time_stamp_in_time_point (the_point:string): time_point;
        addition (the_point1, the_point2:time_point): time_point;
        difference (the_point1, the_point2:time_point): time_point;
end domain
```

A função "in_sec (the_point: time_point) : int", recebe um ponto no tempo com domínio "time_point (the_point)" e retorna o ponto no tempo em segundos com domínio inteiro. As funções "in_year", "in_hour", "in_month", "in_day", "in_minute" são semelhantes a função "in_sec", retornando o ponto no tempo respectivamente em anos, horas, meses, dias e minutos, todos com domínio inteiro.

A função "in_time_stamp (the_point: time_point) : string", recebe um ponto no tempo com domínio "time_point (the_point)" e, retorna o ponto no tempo com domínio *string* no formato dia/mes/ano-hora:minuto:segundo.

As funções "sec_in_time_point", "minute_in_time_point", "hour_in_time_point", "day_in_time_point", "month_in_time_point", "year_in_time_point" recebem o ponto no tempo em segundos, minutos, horas, dias, meses e anos respectivamente, transformando-o em um ponto no tempo com domínio "time_point".

A função "time_stamp_in_time_point (the_point: string) : time_point", recebe um ponto no tempo ("the_point") com domínio *string* no formato dia/mes/ano-hora:minuto:segundo e, retorna o ponto no tempo com domínio time_point.

As funções "addition" e "difference" recebem dois pontos do tempo com domínio "time_point" e, retornam um novo ponto no tempo que é respectivamente, a soma e a subtração dos dois pontos no tempo, com domínio "time_point". Na função "difference", o primeiro ponto irá ter subtraído o segundo.

5.3.1.1 Constantes definidas para o domínio “time_point”

A constante “n”, definida abaixo, com domínio inteiro, representa a quantidade de anos, meses, dias, horas, minutos ou segundos.

<n> [year | month | day | hour | minute |seconds]

A seguir, é apresentada uma constante para representar um ponto no tempo:

“ano/mes/dia-hora:minuto:segundo”

- “ano”: é um número inteiro de 4 algarismos que representa o ano do ponto no tempo;
- “mes”: é um número inteiro pertencente ao intervalo de 01 (janeiro) a 12 (dezembro);
- “dia”: é um número inteiro que pode variar de 1 até 31 dependendo do mês ao qual irá referenciar;
- “hora”: é um número pertencente ao intervalo de 0 (zero) a 23 (vinte e três), representando a hora do ponto no tempo;
- “minuto”: é um número pertencente ao intervalo de 0 (zero) a 59 (cinquenta e nove) representado a quantidade de minutos;
- “segundo”: é um número pertencente ao intervalo de 0 (zero) a 59 (cinquenta e nove) representado a quantidade de segundos;

5.3.1.2 Notações infixadas para o domínio “time_point”

- 1999/12/04: significa que o ano é 1999, o mês 12, o dia 04 e implicitamente a hora, o minuto e o segundo igual a zero;
- 3 months: representa 3 (três) meses;
- 5 years: representa 5 (cinco) anos;
- (the_point1 + the_point2): equivale a escrever “addition(the_point1, the_point2)”;
- (the_point1 – the_point2): equivale a escrever “difference(the_point1, the_point2)”;
- “+” : equivalente a função “addition”, soma dois pontos no tempo e;
- “-” : equivalente a função “difference”, subtrai dois pontos no tempo.

Na função “difference” (ou na notação infixada), o primeiro parâmetro (“the_point1”) irá subtrair o segundo (“the_point2”).

Para exemplificar, considere a notação infixada abaixo:

2000/02/06 - 5 months;

Essa notação é equivalente a ativação da função “difference(2000/02/06, 5 months)”. A partir do ano de 2000, do mês 02 e do dia 06, pretende-se tirar 5 meses.

5.3.2 Domínio “time_interval”

O domínio “time_interval” é definido como a união dos domínios “closed_time_interval”, “begin_open_time_interval” e “end_open_time_interval” definidos nesta sub-seção.

```
domain time_interval
functions
        pertence (the_point:time_point, interval:time_interval): boolean;
        is_begin_open (interval:time_interval): boolean;
        is_end_open (interval:time_interval): boolean;
        endpoint(interval:time_interval):time_point;
```

```
        beginpoint(interval:time_interval):time_point;
end domain
```

A função “pertence”, verifica a partir de um intervalo de tempo (“interval”) e um ponto no tempo (“the_point”), se este ponto pertence ao intervalo. O resultado da função é booleano.

A função “is_begin_open”, verifica a partir de um intervalo de tempo (“interval”), se este intervalo é aberto no ínicio. O resultado da função é booleano.

A função “is_end_open”, verifica a partir de um intervalo de tempo (“interval”), se o intervalo é aberto no final. O resultado da função é booleano.

As funções “endpoint” e “beginpoint”, recebem um intervalo de tempo e retornam, respectivamente, o último ponto e o primeiro ponto.

A seguir é apresentado um exemplo:

```
        pertence (1992, [1990, 1998]);
```

A função “pertence” verifica se o ponto 1992 pertence ao intervalo de tempo [1990,1998]. O resultado é verdadeiro (*true*).

5.3.2.1 Domínio “closed_time_interval”

O domínio “close_time_interval” é definido para representar um intervalo de tempo fechado, isto é, quando sabe-se o início e o final do intervalo.

```
domain closed_time_interval
functions
        add_end (interval:time_interval, end_point:time_point):  time_interval;
        add_begin (begin_point:time_point, interval:time_interval):  time_interval;
        subtract_end (interval: time_interval, end_point:time_point):  time_interval;
        subtract_begin(begin_point:time_point,interval:time_interval): time_interval;
        open_end (interval:time_interval, end_point:time_point):  time_interval;
        open_begin (begin_point:time_point, interval:time_interval):  time_interval;
end domain
```

As funções que podem ser aplicadas a esse domínio são as seguintes:

- “add_end” e “subtract_end”: aumentar e diminuir o final do intervalo;
- “add_begin” e “subtract_begin”: aumentar e diminuir o início do intervalo e;
- “open_begin” e “open_end”: alterar o intervalo, deixando-o aberto no início ou aberto no final;

Para resolver estas funções é necessário que elas recebam como parâmetros o intervalo original que tem domínio intervalo de tempo (“interval”) e, o ponto que será alterado no início (“begin_point”) ou no final do intervalo (“end_point”) com domínio time_point.

Para exemplificar considere, a ativação abaixo, da função “subtract_end”:

```
        subtract_end ([1999/05/05-15:30:35, 2000/06/30-12:00:30], 4 months);
```

A função acima subrai 4 (quatro) meses do final do intervalo. De 2000/06/30-12:00:30 para 2000/02/30-12:00:30.

5.3.2.2 Domínio “begin_open_time_interval”

Para representar o intervalo de tempo aberto no início foi definido o domínio begin_open_time_interval.

```
domain begin_open_time_interval
functions
        close_begin (interval:time_interval, begin_point:time_point): time_interval;
        add_end (interval:time_interval, end_point:time_point): time_interval;
end domain
```

Este domínio apresenta duas funções. A função “close_begin” que recebe um intervalo de tempo (“interval”) e um ponto no tempo (“begin_point”) fechando o intervalo no início com o ponto recebido e, a função “add_end” que recebe um intervalo de tempo (“interval”) e um ponto no tempo (“end_point”) estendendo o intervalo no final com o ponto recebido.

A seguir é apresentada a ativação da função “add_end”:

add_end([1990/01/01-18:25:12, 2000/02/20-15:10:45], 2000/06/08-12);

A funcão acima, estende o intervalo de tempo no final (2000/05/20-15:10:45) pelo ponto no tempo determinado (2000/06/08-12). O resultado é: [1999/01/01-18:25:12,2000/06/08-12].

5.3.2.3 Domínio “end_open_time_interval”

O domínio end_open_time_interval representa o intervalo de tempo aberto no final.

```
domain end_open_time_interval
functions
        close_end (interval:time_interval, end_point:time_point): time_interval;
        add_begin (interval:time_interval, begin_point:time_point): time_interval;
end domain
```

Este domínio apresenta duas funções. A função “close_end” que recebe um intervalo de tempo (“interval”) e um ponto no tempo (“end_point”), e fecha o intervalo no final com o ponto recebido, e a função “add_begin” que recebe um intervalo de tempo (“interval”) e um ponto no tempo (“begin_point”), e estende o intervalo no início com o ponto recebido.

Abaixo é apresentado um exemplo da função “add_begin”:

add_begin([1990/01/01-18:25:12, 2000/05/20-15:10:45], 1995);

A função “add_begin”, estende o ínicio do intervalo (1990/01/01-18:25:12) com ponto no tempo 1995. O resultado é: [1995, 2000/05/20-15:10:45]

5.3.2.4 Constantes definidas para o domínio “time_interval”

- “>>” : significa que o intervalo está aberto no final;
- “<<” : significa que o intervalo está aberto no início.
- [begin_point, end_point]: representa um intervalo de tempo fechado, onde o primeiro argumento é o valor inicial do intervalo (“begin_point”) e o segundo o valor final (“end_point”);
- [begin_point, >>]: representa um intervalo de tempo aberto no final (>>), sendo o início do intervalo determinado pelo ponto no tempo “begin_point”;
- [<< , end_point]: representa um intervalo de tempo aberto no início (<<), sendo o final do intervalo determinado pelo ponto no tempo “end_point”;

A seguir é apresentado um exemplo da ativação função “close_begin” utilizando a constante:

close_begin ([<<, 2000] , 1990);

Esta função acima, fecha o início do intervalo (<<) com o ponto no tempo 1999. O resultado é: [1990,2000].

5.3.2.5 Notações infixadas para o domínio "time_interval"

- (the_point ϵ interval): onde "ϵ" é símbolo da lógica, representando a palavra "pertence";

Essa notação verifica se um ponto ("the_point") pertence a um intervalo ("interval"). É equivalente a escrever: "pertence (the_point, interval)".

Um exemplo utilizando a notação infixada:

1992 ϵ [1990, 1998];

A notação acima verifica se o ponto 1992 pertence ao intervalo de tempo [1990,1998]. O resultado é verdadeiro (*true*). Equivalente a ativação da função "pertence (1992, [1990, 1998])";

5.4 Classes temporais

Além de incluir domínios temporais, a extensão temporal de OASIS permite a definição de classes temporais.

Em OASIS, uma classe é composta por um conjunto de instâncias, um mecanismo de identificação para elas e um *template* comum a todas as instâncias.

O comportamento de um objeto pode ser alterado devido à ocorrência de um evento (fato ocorrido em um determinado instante de tempo). Neste caso, é importante que o modelo permita apresentar a história destes eventos. Para os eventos que são declarados como temporais, é guardado o registro das suas ocorrências, associando a cada ocorrência o seu tempo de transação.

Suponha que o sistema possui um relógio e que a aplicação tenha que guardar o valor do relógio sempre atualizado. Para isto, implicitamente em todas as classes, é definido um atributo denominado ***currenttime*** com domínio do tipo *date*, que recebe como valor inicial, o valor do relógio. Para que esse atributo seja atualizado, é definido um evento denominado ***tick*** que se encarregará de atualiza-lo.

Nas seções que seguem estão definidas as seguintes classes temporais:

- it_nat;
- it_nat_value;
- it_int;
- it_int_value;
- temporal_event;
- event_occurrence_list;
- temporal_event_occurrence.

OASIS foi concebido para especificar uma única aplicação, isto significa que não apresenta o conceito de *genericidade* [MEY 97]. Este conceito permite utilizar tipos parametrizados. Por exemplo, se OASIS fosse genérico, seria permitido definir uma classe "it<tipo>" e o "tipo" poderia ser inteiro, real, caracter entre outros domínios.

Assim, as classes apresentadas devem ser consideradas apenas como esqueletos para as classes que compõem uma aplicação. Por exemplo, a classe “it_nat” serve de esqueleto para as classes “it_float”, “it_string” e assim por diante. O mesmo se aplica as demais classes.

A seguir são apresentadas as classes temporais.

5.4.1 Classe “it_nat”

A classe “it_nat” é definida para representar os intervalos temporais para os números naturais.

```
class it_nat
variable attributes
        the_intervals: it_nat_value list [1,*]
        the_point: time_point;

derived attributes
        first_point: time_point;
        last_point: time_point;
        the_values (the_intervals): nat list [1,*];
        min_value: nat;
        max_value: nat;
        actual_value: nat;
        value_at_time (the_point): nat;
        values_at_interval (interval): nat list [1,*];
        pair_values_interval_at_interval (interval): time_interval list [1,*];
        interval_at_values (value): nat list [1,*];


events
        insert_interval (interval, value);
        remove_interval (interval, value);
        remove_all_intervals;
        from_now_new_value (interval, value);

valuations
        [insert_interval] := insert (the_intervals[interval],value);
        [remove_interval] := remove (the_intervals[interval], value);
        [remove_all_intervals] := remove_all (the_intervals);
        [from_now_new_value (valor)] the_intervals[interval], value:= valor;

preconditions
        insert_interval (interval, value)
            if (not exists a_interval in the_intervals
                where (not is_end_open (a_interval) impl
                    (endpoint (a_interval) > beginpoint (interval))
                        and (beginpoint(a_interval) < endpoint (interval)))
                and  (not exists a_interval in the_intervals
                    where not is_begin_open (a_interval) impl
                        (endpoint (a_interval) > beginpoint (interval))
                            and (beginpoint(a_interval) < endpoint (interval)))


end class
```

Em OASIS, a especificação de atributos derivados da-se através de uma cláusula *derivations* e a especificação da semântica de um evento através de cláusulas *valuations* e *preconditions*.

Para simplificar o presente texto, optou-se por apresentar a semântica dos atributos derivados de maneira informal.

A classe “it_nat” apresenta um atributo variado do tipo objeto (“the_intervals”), que é uma lista onde ficam registrados todos os intervalos de tempo.

Os atributos derivados são os seguintes:

- “first_point”, retorna o primeiro intervalo no tempo (“interval”) pertencente à lista de intervalos (“the_intervals”). A função “beginpoint”, definida no domínio “time_interval”, retorna o primeiro ponto do intervalo em questão;
- “last_point” é semelhante ao “first_point”: retorna o último intervalo (“interval”) pertencente à lista de intervalos (“the_intervals”);
- “the_values”, retorna a lista de valores que existem em todos intervalos (“the_intervals”);
- “min_value” e “max_value”, retornam respectivamente, o menor e o maior valor do ponto no tempo ;
- “actual_value”, retorna o valor contido no ponto no tempo atual (*currenttime*);
- “value_at_time”, recebe um ponto no tempo e retorna o valor correspondente ao ponto;
- “values_at_interval”, recebe um intervalo de tempo e retorna os valores contidos no intervalo;
- “pair_values_interval_at_interval”, recebe um intervalo de tempo e retorna para cada valor o intervalo de tempo correspondente;
- “interval_at_values”, recebe um valor e retorna o intervalo que contém esse valor.

Os eventos são definidos abaixo:

- “remove_interval”, recebe um intervalo e um valor (“interval”, “value”), e remove-os da lista de intervalos (“the_intervals”);
- “remove_all_intervals”, remove toda a lista de intervalos (“the_intervals”);
- “from_now_new_value”, recebe um intervalo e um valor (“interval”, “value”), definindo que a partir de agora *(“now”),* o intervalo passa a ter um novo valor (“value”:=valor);
- “insert_interval”, recebe um intervalo e um valor (“interval”, “value”) e insere o intervalo com seu respectivo valor se a pré-condição permitir.

A pré-condição não permite que o intervalo sobreponha outro intervalo já existente na lista.

5.4.2 Classe “it_nat_value”

A classe “it_nat_value” apresenta um atributo para especificar o intervalo de tempo (“interval”) e o seu respectivo valor (“value”).

```
class it_nat_value
variable attributes
        value: nat;
        interval: time_interval;
end class
```

A classe “it_nat_value” armazena um intervalo (“interval”) e um valor (“value”). Esse par intervalo-valor tem que ser anexado a lista de intervalos (“the_intervals”) que está definida na classe “it_nat”. Portanto, a classe “it_nat_value” é agregada da classe “it_nat”, conforme a definição abaixo. Esta definição também pode ser vista na figura 5.1, através da notação UML [LAR 97].

```
it_nat aggregation of
        dynamic inclusive it_nat_value
                towards (1,*) from (1,1) list [1,*]
```

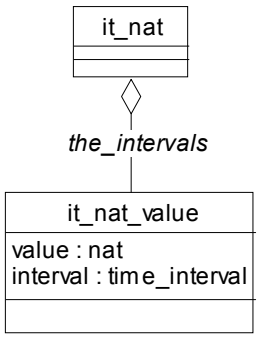

FIGURA 5.1- Diagrama de classes (intervalos_temporais)

5.4.3 Classe “it_int”

As classes “it_int” e “it_int_value”, são apresentadas a seguir somente para mostrar sua diferença com as classes anteriormente definidas (“it_nat” e “it_nat_value”). Esta diferença é o domínio, isto é, as primeiras classes definidas apresentam intervalo de tempo para o domínio dos *naturais* e, apresentar para o domínio dos *inteiros*. Se houver necessidade de utilizar intervalo de tempo para o tipo caracter, as classe “it_char” e “it_char_value” devem ser definidas e, assim por diante.

```
class it_int
variable attributes
        the_intervals: it_int_value list [1,*]

derived attributes
        first_point (interval) : time_point;
        last_point (interval) : time_point;
        the_values (the_intervals): int list [1,*];
        min_value (the_values): int;
        max_value (the_values) : int;
        actual_value: int;
        value_at_time (the_point): int;
        values_at_interval (interval): int list [1,*];
        pair_values_interval_at_interval (interval): time_interval list [1,*];
        interval_at_values (value): int list [1,*];


events
        insert_interval (interval, value);
        remove_interval (interval, value);
        remove_all_intervals;
        from_now_new_value (interval, value);

valuations
```

```
        [insert_interval] := insert (the_intervals[interval],value);
        [remove_interval] := remove (the_intervals[interval], value);
        [remove_all_intervals] := remove_all (the_intervals);
        [from_now_new_value (valor)] the_intervals[interval], value:= valor;

preconditions
        insert_interval (interval, value)
            if (not exists a_interval in the_intervals
                where (not is_end_open (a_interval) impl
                    (endpoint (a_interval) > beginpoint (interval))
                        and (beginpoint(a_interval) < endpoint (interval)))
                and (not exists a_interval in the_intervals
                   where not is_begin_open (a_interval) impl
                       (endpoint (a_interval) > beginpoint (interval))
                          and (beginpoint(a_interval) < endpoint (interval)))

end class
```

5.4.4 Classe “it_int_value”

```
class it_int_value
variable attributes
        value: int;
        interval: time_interval;
end class
```

5.4.5 Classe “temporal_event”

A classe temporal “temporal_event” foi definida para representar eventos temporais.

```
class temporal_event
variable attributes
        the_event_list: event_occurrence_list list [1,*]

derived attributes
        event_time_point: event_occurrence_list;

events
        occurs;

end class
```

Esta classe apresenta um atributo “the_event_list” que é do tipo objeto (class “event_occurrence_list”), que gera uma lista das ocorrências dos eventos.

O atributo derivado “event_time_point”, gera uma lista dinamicamente dos pontos no tempo em que o evento ocorreu. A seguir é apresentada a sintaxe deste atributo:

(<ref_objeto>.<id_evento>) onde:

- <ref_objeto>.<id_evento> indica qual o evento que ocorreu. Esse evento tem que ser declarado como temporal.

O resultado da derivação é uma lista de pontos no tempo (“event_occurrence_list”), nos quais o evento em questão foi disparado pelo objeto.

O evento “occurs”, é disparado quando um evento é ativado.

5.4.6 Classe “event_occurrence_list”

A classe “event_occurrence_list” define a ocorrência de um evento (quem ativou o evento e quando este evento ocorreu).

```
class event_occurrence_list
variable attributes
        the_events : temporal_event_occurrence;
events
        insert;
end class
```

O registro é feito através do evento “insert”.

5.4.7 Classe “temporal_event_occurrence”

A classe “temporal_event_occurrence” (figura 5.2) apresenta dois atributos conforme especificação a seguir:

```
class temporal_event_occurrence
variable attributes
        when : time_point;
        actor: object;
end class
```

O atributo “when” define quando ocorreu o evento, e o atributo “actor” define quem ativou o evento.

Na semântica de OASIS, existe o conceito de “ações” que incluem três elementos: referência do cliente, referência do servidor e descrição do serviço. Assim, uma ação está definida pela tupla (cliente, servidor, serviço). Esse conceito é demonstrado na classe “temporal_event_occurrence”, através do atributo “actor”, que é do tipo objeto, isto é, o cliente que ativou a ocorrência do evento temporal.

O relacionamento entre as classes “temporal_event”, “event_occurrence_list” e “temporal_event_occurrence” são apresentadas a seguir através da notação UML [LAR 97] (figura 5.2), e através da especificação OASIS.

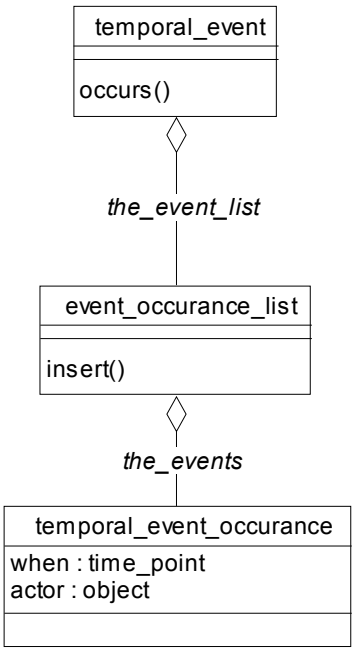

FIGURA 5.2- Diagrama de classes (eventos temporais)

```
temporal_event  aggregation of
        dynamic event_occurrence_list
                towards (1,*) from (1,1) list [1,*]

event_occurrence_list aggregation of
        dynamic inclusive temporal_event_occurrence
                towards (1,*) from (1,1) list [1,*]
```

5.5 Extensão das fórmulas de OASIS

As fórmulas em OASIS foram estendidas (figura 5.3) para inserir as expressões *after, before* que são utilizadas, principalmente, em especificações temporais. Além disso, as expressões *impl, for all* e *exists* foram estendidas para tratamentos de listas.

Através da extensão temporal de OASIS, é possível determinar eventos que ocorram condicionalmente a eventos que ocorreram no passado ou a estados que o objeto assumiu no passado.

```
<fórmula>       ::= '(' <fórmula> ')' | not '(' <fórmula> ')'
                    | <fórmula> and <fórmula>
                    | <fórmula> or <fórmula>
                    | <fórmula> impl <fórmula>
                    | <time_point> after <time_point>
                    |  <time_point> before <time_point>
                    | for all <atributo/componente> in <lista>
                                [where '{'<fórmula>'}']
                    | exists <atributo/componente> in <lista>
                                [where '{'<fórmula>'}']
                    | <expressão_aritmética> <op_rel> <expressão_aritmética>
                    | true | false
<op_rel>        ::= '=' | '<>'| '<' | '<=' | '>' | '>='
```

FIGURA 5.3- Fórmulas estendidas em OASIS

A seguir, é apresentado um exemplo de uma pré-condição que condiciona a ocorrência de um evento a estados que o objeto assumiu no passado.

Na vídeo-locadora, existem três categorias de usuário: ouro, prata e bronze. A categoria atual de um usuário é indicada pelo atributo "categoria". Quando um usuário associa-se a locadora, ele assume a categoria bronze.

Para que um usuário tenha um desconto de 10% no pagamento da locação de uma fita deve ser disparado o evento "desconto". A pré-condição para a ocorrência deste evento é que o usuário não tenha estado em débito durante o último ano. O fato de um usuário estar em débito é indicado pelo atributo temporal "devedor", que é verdadeiro quando o usuário encontra-se nesta condição.

Abaixo está especificada uma pré-condição que garante que um usuário não obtenha desconto se tiver sido devedor durante o último ano.

```
preconditions
        desconto (preço*0,90) if
                        (categoria="ouro" or categoria="prata")
                    impl for all d_value in devedor.the_values [now-(1 year), now]
```

(not d_value)

A pré-condição especifica que o evento “desconto” somente pode ocorrer, caso a categoria seja ouro ou prata, e que deve valer para todos os valores do atributo “devedor” durante o último ano, não sendo este valor verdadeiro (o usuário não esteve no estado de devedor).

5.6 Especificação para atributos e eventos temporais

Nesta seção é mostrado como atributos e eventos temporais são especificados. Em outros termos, é mostrado como os conceitos das seções anteriores são usados para especificar uma classe.

Utilizando a sintaxe original de OASIS [PAS 92b, PAS 95a, LET 98b], a especificação de um atributo temporal (“preço”) e de um evento temporal (“emprestar”) na classe fita da vídeo-locadora é a seguinte:

```
classe fita
…
variable attributes
        localização : string;
        lançamento: bool (true);
        preço: it_nat;
        emprestar: temporal_event;
..
events
        insere_fita new;
        ...
```

“Preço” é um atributo temporal com domínio “it_nat” representando o domínio dos números *naturais* com dimensão temporal por intervalo de tempo.
“Emprestar” é um evento temporal. Isto significa que o sistema irá manter o registro das ocorrências deste evento, associando a cada uma delas o seu tempo de transação.
Nesta sintaxe, para referenciar a ocorrência de um evento temporal, é necessário usar o termo abaixo:

fita.emprestar.occurs;

Ou seja, como eventos temporais são declarados como sendo atributos variáveis, seu tratamento difere do de eventos não temporais. Para contornar esta restrição e homogeneizar o tratamento de atributos e eventos temporais, introduz-se uma nova sintaxe para definição destes elementos, através de seções para atributos temporais (*temporal attributes*) e eventos temporais (*temporal events).*

A sintaxe para especificação de classe passa a ser a seguinte:

```
<def_classe>      ::=   class <id_classe>
                  <mecanismos_identificação>
                  <atributos_constantes>
                  [ <atributos_variáveis> ]
                  [ <atributos_temporais>]
                  [ <atributos_derivados> ]
                  [ <derivações> ]
                  [ <restrições_integridade> ]
                  <eventos>
                  <eventos temporais>
```

```
[ <operações> ]
[ <gatilhos> ]
[ <avaliações> ]
[ <pré-condições>]
[ <protocolos> ]
end class
```

A classe fita mencionada anteriormente, é especificada com a inserção das cláusulas temporais para atributos e eventos, como segue:

```
variable attributes
        localização : string;
        lançamento: bool (true);
        ...
temporal attributes
        preço: nat;
        ...
events
        insere_fita new;
        ...
temporal events
        emprestar;
                . . .
```

Observe que esta notação pode ser considerada como uma forma abreviada da notação original de OASIS, sem extensão da sua semântica.

Esta notação é utilizada no estudo de caso do próximo capítulo.

6 Estudo de caso em OASIS Temporal

Este capítulo apresenta o estudo de caso em OASIS Temporal. Na primeira seção deste capítulo é apresentada a continuação da descrição do sistema de vídeo-locadora definido na seção 3.1. Na segunda seção, é apresentada a especificação do sistema em OASIS Temporal.

6.1 Descrição de um sistema de vídeo-locadora

Para exemplificar OASIS Temporal, considere a descrição apresentada na seção 3.1. O enunciado do estudo de caso é estendido como segue.

Suponha que existem três tipos de categorias de usuários na vídeo-locadora: ouro, prata e bronze. Quando um usuário associa-se a locadora, assume a categoria bronze. Conforme vai ascendendo de categoria ele terá vantagens na locadora como por exemplo, tolerância a mais de um dia na devolução de uma fita, ou até mesmo desconto no pagamento.

Para que um usuário consiga atingir a próxima categoria, ele não poderá estar em débito durante o último ano. Caso ele pertença à categoria ouro ou prata e estiver em débito nos últimos três meses, ele será rebaixado automaticamente para a categoria bronze.

Se o usuário pertencer a catogoria ouro ou prata, ele poderá emprestar mais de 3 (três) fitas por dia na vídeo-locadora.

Se o usuário for da categoria prata ele poderá ter um desconto de 5% (cinco por cento) ao efetuar o pagamento. Se for da categoria ouro o desconto é de 10% (dez por cento).

6.2 Especificação do sistema OASIS Temporal

É necessário definir as mesmas classes do estudo de caso apresentado no capítulo 3, ou seja, as classes fita, usuário, locação e multa. Em OASIS Temporal a definição de agregação continua a mesma, definida abaixo:

```
multa aggregation of
        static relational usuario towards (1,1)
        static relational fita towards (1,1)

locação aggregation of
        static relational usuario towards (1,1)
        static relational fita towards (1,1)
```

A seguir, é apresentado o esquema conceitual da vídeo-locadora, onde são definidas as classes necessárias para especificação do sistema em OASIS Temporal. As informações inseridas ou modificadas com aspectos temporais estão em destaque.

6.2.1 Classe usuário

```
conceptual schema video_locadora
-----------------------------------------
class usuario

identification
        n_usuario : (n_usuario);
```

```
constant attributes
        n_usuario : string;
        nome_usu : string towards (1,1);

variable attributes
        endereço : string;
        telefone: string towards (0,*) list[0..*];
        qtde_fitas: int (0);
        desconto: nat;
        categoria: string (“bronze”);

temporal attributes
        devedor: bool (false);

constraints
        if (categoria = “bronze”)
                impl  (qtde_fitas <= 3)

events
        insere_usu new;
        remove_usu destroy;
        modifica (endereço: string , telefone:string);
        desconto;
        upgrade (new_class: string);
        downgrade (“bronze”);

temporal events
        emprestar;
        devolver;

triggers
        ::downgrade (“bronze”) when
                for all dev_value in
                     devedor.the_values [currenttime-(3 months), currenttime]
                        (dev_value)

valuations
        [modifica (ende,fone)] endereço:= ende, telefone:= fone,
        [emprestar ] qtde_fitas:= qtde_fitas +1;
        [devolver ] qtde_fitas:= qtde_fitas -1;
        [upgrade (new_class)] categoria:= new_class;
        [downgrade (“bronze”)] categoria := “bronze”;

preconditions
        remove_usu if qtde_fitas = 0;
        emprestar if {qtde_fitas <= 3} or {categoria <> “bronze”}
        upgrade (new_class) if
                (new_class = “ouro” or new_class = “prata”)
                        impl for all d_value in
                                devedor.the_values [currenttime-(1 year), currenttime]
                                        (not d_value)

        desconto (fita.preço.actual_value*0,90)
                if  (categoria= “ouro”)

        desconto (fita.preço.actual_value*0,95)
                if  (categoria=“prata”)
```

```
end class
```

Na extensão temporal de OASIS, na especificação da classe usuário, é inserido uma seção para declaração dos atributos temporais (*temporal attributes*) e uma seção para declaração dos eventos temporais (*temporal events*).

Através da extensão temporal o atributo variável "categoria" foi definido para indicar qual a categoria que o usuário pertence (ouro, prata ou bronze). Ao ser cadastrado ele assume a categoria bronze.

O atributo "devedor" é definido como temporal para guardar toda a história de débito do usuário e com isto atender os requisitos da vídeo-locadora.

Para a especificação da restrição de integridade utiliza-se a expressão *impl*. A restrição expressa que quando o atributo "categoria" for bronze implica que a quantidade de fitas tem que ser menor ou igual a 3 (três).

Nesta classe são definidos os seguintes eventos temporais:

- "emprestar";
- "devolver".

Isto significa que o sistema, internamente, mantém o registro de todas as ocorrências destes eventos.

O disparo define que o usuário será rebaixado automaticamente para a classe bronze, se tiver sido devedor alguma vez nos últimos três meses.

Com a extensão temporal, são definidas as avaliações a seguir:

- na ocorrência do evento "upgrade", o atributo "categoria" recebe um novo valor (determinado pelo atributo "new_class");
- na ocorrência do evento "downgrade", o atributo "categoria" recebe o valor "bronze".

As pré-condições com aspectos temporais desta classe são as seguintes:

- um usuário não pode ascender de classe (evento "upgrade") caso tenha sido devedor durante o último ano. Para definir esta pré-condição, utilizou-se o atributo temporal "devedor";
- o evento "desconto" será disparado para que o usuário tenha um desconto de 10% no pagamento da locação de uma fita se pertencer a categoria ouro.
- o evento "desconto" será disparado para que o usuário tenha um desconto de 5% no pagamento da locação de uma fita se pertencer a categoria prata.

Através da extensão temporal, é possível especificar que um gatilho depende da história da ocorrência de um determinado evento. Nesta classe, é definido um gatilho que rebaixa um usuário para a categoria bronze (evento "downgrade"), sempre que, durante um período de três meses, este usuário permaneceu devedor (atributo temporal "devedor").

6.2.2 Classe fita

```
class fita

identification
        n_fita : (n_fita);
```

```
constant attributes
        n_fita : string;
        nome_autor:  string;
        titulo: string;
        assunto: string;
        descrição: string;

variable attributes
        localização : string;
        disponível: bool (false) towards (1,1);
        lançamento: bool (true);

temporal attributes
        preço: nat;


derived attributes
        total: nat (0);

derivations
        total:= preço.actual_value + multa.valor_multa;

constraints
        for all a_interval in (emprestar.event_time_point)
                (first (cadastrar.event_time_point ))
                        before a_interval)
events
        insere_fita new;
        remove_fita destroy;
        modifica_lançamento;
        alterar_preço(npreço);

temporal events
        emprestar;
        devolver;
        cadastrar;

trigger
        ::modifica_lançamento when
                {month(currenttime) – month (first (cadastrar.event_time_point)) > 6
                        and lançamento:= true}

valuations
        [modifica_lançamento] lançamento:= false, npreço:=preço.actual_value * 0,80
        [alterar_preço(npreço)] preço.actual_value :=npreço;
        [emprestar] disponível:= false;
        [devolver] disponível:= true; total:=0;

preconditions
        remove_fita if {disponível :=true};
        emprestar if {disponível :=true};
        devolver if {disponível :=false};

end class
```

Na classe fita, com a extensão temporal, o atributo “preço” é definido como temporal, para guardar toda a história dos preços de uma fita na vídeo-locadora.

O cálculo do atributo derivado “total”, expressa que o valor do total da fita é definido pelo preço atual da fita acrescido da multa, caso o usuário não tenha entregue a fita na data correta.

A restrição de integridade garante que o usuário só empreste uma fita na vídeo-locadora depois dela ter sido cadastrada.

Na classe fita apresentada na seção 3.2.2 é definido um atributo constante “data_compra”, que define quando a fita foi adquirida na vídeo-locadora.

Com a extensão temporal, a definição desse atributo não é mais necessária, já que foi definido o evento temporal “cadastrar”. Os eventos “emprestar” e “devolver” também são definidos como temporal, para que se possa referenciar o histórico de todas as suas ocorrências.

O gatilho especifica que o evento “modifica_lançamento”, deve ocorrer quando o mês do primeiro ponto no tempo em que ocorreu o evento “cadastrar” subtraído do mês atual do sistema for maior que seis, e o atributo “lançamento” for verdadeiro.

A avaliação especifica que quando for alterado o preço de uma fita o preço atual da fita (*currenttime*) receberá o valor estipulado pelo atributo “npreço”. No caso da fita deixar de ser lançamento (seis meses após seu cadastro), o atributo “npreço” será o preço atual da fita reduzido em 20% (vinte por cento).

6.2.3 Classe locação

```
class locação

identification
        ref : (usuario.n_usuario: string, fita.n_fita:string, data_loc:date);

constant attributes
        data_loc:date;

variable attributes
        multado:bool (false);
        data_devolução: date;

events
        emprestar new
                calling to members
                        fita.emprestar, usuario.emprestar;

        devolver (day(currenttime) destroy
                calling to members
                        fita.devolver, usuario.devolver;

trigger
        multa::insere_multa(n_fita, n_usuario,currenttime) when
           ((last(emprestar.event_time_point) before
                        (day(currenttime) – 1day) and (multado:=true)

valuations
        [devolver] data_devolução:= currenttime;
        [multa::insere_multa(n_fita, n_usuario,currenttime)] multado:=true;

end class
```

Nesta classe é definido um gatilho para inserir uma multa na classe multa (evento "insere_multa") quando o último ponto no tempo do evento "emprestar" anteceder o dia registrado no sistema menos 1 (um) dia (prazo máximo que o usuário pode ficar com uma fita é de um dia), e o atributo "multado" for verdadeiro.

Quando o usuário devolve a fita, a data de devolução recebe o dia correspondente a data do sistema (*day(currenttime*)).

6.2.4 Classe multa

```
class multa

identification
        ref : (usuario.n_usuario : string, fita.n_fita:string, data_mul:date);

constant attributes
        data_mul:date;

variable attributes
        status_pagto: bool (false);
        valor_dia: nat;

derived attributes
        valor_multa: nat;

derivations
        valor_multa:= {(day(currenttime) –
                                day(last(emprestar.event_time_point)) * valor_dia};

events
        insere_multa new;
        remove_multa destroy;
        altera_valor_dia (nv)
        altera_valor_multa(0);
        altera_status_pagto;

trigger
        ::altera_status (true) when {valor_multa:=0};

valuations
        [altera_valor_dia(nv)] valor_dia:=nv;
        [altera_valor_multa(0)] valor_multa:=0;

preconditions
        remove_multa if {status_pagto :=true and fita.total:=0};

end class
```

A derivação do atributo "valor_multa" é calculada multiplicando o valor contido no atributo "valor_dia" (especifica o valor da multa diária) pela quantidade de dias em atraso (determinada pelo dia do sistema subtraído do dia do último ponto no tempo em que ocorreu o evento "emprestar").

7 Conclusão

O modelo OASIS Temporal proposto neste trabalho é uma extensão do modelo OASIS original para especificação de atributos e eventos temporais.

A tabela 7.1 compara o modelo OASIS temporal proposto neste trabalho com os modelos OO apresentados na seção 4.2.

TABELA 7.1- OASIS Temporal X Modelos OO

Aspectos/ Modelos	OODAPLEX	TDM	TF-ORM	OASM*/ T	OASIS Temporal
Rótulos temporais	elemento temporal	elemento temporal	ponto no tempo	Intervalo temporal	elemento temporal
Tempo transação / Tempo de validade	bitemporal	bitemporal	bitemporal	Tempo de validade	tempo de validade
Propriedade temporais e não temporais	sim	sim	sim	não (tempo no objeto)	sim
Eventos temporais	não	não	não	não	sim

Como pode ser verificado, o modelo OASIS Temporal proposto satisfaz os aspectos principais identificados. Utiliza elemento temporal como rótulo temporal por ser o mais completo para representação de tempo. Foi decidido utilizar somente tempo de validade para atributos, devido à necessidade das aplicações. Permite a representação de propriedades temporais e não temporais, incluindo eventos, os quais são fundamentais para as aplicações previstas.

7.1 Extensão temporal de OASIS

Na literatura, há várias propostas de estender abordagens formais de especificação de software com aspectos temporais. Estas propostas concentram-se na extensão da perspectiva **estática**, isto é no aspecto temporal de atributos ou associações. Isto permite que, na linguagem de especificação, seja feita referência à história dos valores da propriedade estática em questão. Modelos formais orientados a objetos incluem também a perspectiva **dinâmica** através do conceito de evento. Neste trabalho é mostrado como OASIS, uma abordagem formal de especificação OO, pode ser estendida incorporando o conceito de **evento temporal**.

Este conceito aumenta grandemente a expressividade da abordagem de especificação.

Caso a linguagem não possua o conceito de evento temporal, o mesmo deve ser "implementado" através de atributos temporais. Exemplificando, no caso de uma vídeo-locadora, se for considerada somente uma extensão temporal das propriedades estáticas, para poder referenciar a história dos empréstimos de uma fita, é necessário criar um atributo temporal (ou uma associação temporal) como “data_de_empréstimo”. Este atributo tem seu valor alterado cada vez que ocorre o evento “emprestar”. Já em uma linguagem que possua o conceito de evento temporal, o atributo “data_de_empréstimo” não é necessário, bastando a qualificação do evento “emprestar” como sendo temporal. Referências aos vários pontos no tempo em que ocorreu o evento em questão são feitas diretamente ao evento temporal.

A extensão temporal de OASIS inclui a extensão das fórmulas e expressões de OASIS para referenciar a história da ocorrência de eventos, e com isto, aumentar a expressividade de pré-condições, gatilhos, restrições de integridade e avaliações.

7.2 Limitações de OASIS

OASIS não apresenta o conceito de genericidade. Isto porque foi concebido para modelar aplicações específicas. Ao definir as classes temporais, dependendo da aplicação, é necessário definir várias classes temporais, sendo que a única diferença entre elas é o domínio. Este problema foi contornado através do uso da sintaxe específica para atributos e eventos temporais.

Além de atributos temporais e eventos temporais, uma extensão completa incluiria classes temporais e agregações temporais. Estes aspectos não foram aqui tratados pois demandariam tempo excessivo e seu tratamento seria análogo ao aqui proposto.

7.3 Trabalhos futuros

Na Universidade Politécnica de Valência está em desenvolvimento um trabalho de doutorado que tem como objetivo inserir o aspecto tempo em OO-Method (proposta metodológica baseada em OASIS). É importante compatibilizar os aspectos temporais inseridos em OASIS com os aspectos temporais inseridos em OO-Method.

A extensão temporal proposta pode ser ainda mais enriquecida com a possibilidade de definição classes temporais na aplicação. Por exemplo, na vídeo-locadora poderia-se declarar a classe usuário com temporal. Com isto, se o usuário deixasse de ser sócio da locadora, ficaria com seu cadastro inativo caso resolvesse voltar após algum tempo, seu cadastro seria ativado novamente.

Finalmente, para identificar outros aspectos a serem aperfeiçoados ou modificados seria necessário experimentar o modelo OASIS Temporal em casos reais.

Glossário

Atributo.
Um atributo é um par (nome, conjunto de valores), sendo que o nome é o identificador e conjunto de valores são os valores que o atributo pode assumir.

Agregação.
Operador entre classes que permitem modelar a noção de objeto composto (chamado agregado), de outros chamados componentes.

Avaliação.
Fórmula que estabelece como as ações afetam o estado de um objeto.

Ciclo de vida de um objeto.
Seqüência de passos que acontecem a um objeto, desde sua criação até sua destruição.

Classe.
Termo que contém um conjunto de instâncias, um mecanismo de identificação e um *template* comum a todas as suas instâncias.

Classe de objetos e Classes de regras.
Tanto os objetos como as regras são vistos como objetos individuais que possuem estado e comportamento. Se as instâncias são objetos, então a classe chama-se classes de objetos. Se as instâncias são regras, então a classe chama-se classe de regras.

Cliente.
Objeto que atua como gerador de uma determinada ação.

Derivação.
Fórmula que estabelece o valor de um atributo em função dos valores de outros atributos.

Domínio.
Conjuntos de valores que os atributos podem assumir.

Estado do objeto.
É o conjunto de seus atributos com valor.

Espécies.
São as subclasses de mais baixo nível.

Especificação de processo.
Representação de seqüências de ações que são permitidas (ou obrigatórias) na vida de um objeto.

Evento.
Fato ocorrido em um determinado instante de tempo.

Evento new.

Primeiro evento na vida de um objeto, implicando sua criação.

Evento destroy.
Último evento na vida de um objeto, implicando sua destruição.

Gatilhos.
Um gatilho especifica um evento que deve ocorrer, quando o objeto atinge um determinado estado.

Generalização.
Forma de herança onde uma classe é definida como superclasse de um conjunto de subclasses.

Herança.
Mecanismo pelo qual uma classe (chamada subclasse) receba propriedades (estrutura e comportamento) de outra classe (chamada superclasse).

Herança múltipla.
A herança múltipla estabelece que as instâncias de uma classe recebam propriedades a partir de duas ou mais superclasses.

Instância de uma classe.
Objeto que pertence a uma classe.

Mecanismo de identificação.
Conjunto de pares atributo-valor que determinam um objeto de forma única no *template* da classe.

Metaclasse.
Todos os objetos são instâncias de uma alguma classe, e as classes, por sua vez instâncias da metaclasse.

Objeto.
Unidade de modelo no enfoque orientado a objetos, encapsulando estrutura (determinada por seus atributos) e comportamento (aspecto associado às ações que lhe acontecem).

***OID (Object Identifier)*.**
Cada objeto possui um identificador *(oid)*. O *oid* estabelece a identidade do objeto.

Operação.
Seqüência de ações que obrigatoriamente devem ocorrer na vida de um objeto.

Partição estática.
Mecanismo de herança para que um objeto, desde sua criação, se especialize como instância de alguma das subclasses da partição.

Partição dinâmica.
Mecanismo de herança para que um objeto possa em um instante, especializar-se como instância de alguma das subclasses definidas nesta partição.

Passo.
É um conjunto consistente de ações que ocorrem em um mesmo instante da vida de um objeto.

Pré-condição.
Condição que deve satisfazer o estado de um objeto para que uma ação ocorra e produza uma transição entre estados.

Protocolos.
Estabelece sequências permitidas de ações.

Regra.
Estrutura e comportamento que um objeto pode desempenhar temporariamente. É um novo objeto com um *oid* próprio.

Restrição de Integridade.
São fórmulas baseadas no estado do objeto e que devem ser verdadeiras para cada um de seus estados.

Serviço.
Um serviço é um evento ou uma operação. No primeiro caso trata-se da abstração de uma mudança de estado atômico e instantâneo. Uma operação é um serviço não atômico, e que geralmente tem duração.

Subclasses.
Uma subclasse é a classe derivada a partir de outra mediante herança.

Superclasses.
Uma superclasse é a classe na qual outras se derivam por herança.

Transição válida.
É a transição de um objeto de um estado a outro estado válido.

Template.
Padrão de estrutura e comportamento para um conjunto potencial de objetos.

Vida ou traço de um objeto.
Seqüência finita de passos na vida de um objeto, contendo o passo inicial da ação de criação.

Bibliografia

[ALA 90] ALASHQUR, A. M.; SU, S. Y. S.; LAM, H. A Rule-based language for deductive object-oriented dababases. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 6., 1990, Los Angeles. **Proceedings…** Los Angeles: IEEE, 1990. p.58-67.

[ANT 97] ANTUNES, D. C. **Modelagem Temporal de Sistemas**: uma abordagem fundamentada em Redes de Petri. Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de Mestrado.

[AQV 84] AQVIST, L. Deontic Logic. In: GABBAY D. M.; GUENTHNER F. (Eds.). **Handbook of Phisophical Logic II**. Reidel:[s.n.], 1884. p. 605-714.

[BAT 92] BATINI, C.; CERI, S.; NAVATHE, S. B. **Conceptual Database Design – an Entity-Relationship Approach**. Redwood City: The Benjamin/Cummings, 1992.

[CAN 91] CANÓS, J. H. et al. Object Oriented and Funcional Specification of Information Systems. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 1991. **Proceedings...** Berlim: [s.n.], 1991.

[CER 98] CERDÁ, J. H. C. **OASIS:** Un Lenguaje Único para Base de Datos Orientadas a Objetos. Valencia: Departament de Sistemes Informàtics i Computació, Universitat Politècnica de Valencia,1998. Tesis Doctoral.

[CLI 87] CLIFFORD J.; CROCKER, A. The Historical relational data model (HRDM) and algebra based on lifespans. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 1987, Los Angeles, California. **Proceedings...** Los Angeles: [s.n.], 1987. p.528-537.

[CLI 95] CLIFFORD, J. TUZHILIN, A. (Eds.). **Recent Advances in Temporal Databases**. Berlin: Springer-Verlag, 1995.

[DEA 91a] DeANTONELLIS, V.; PERNICI, B. **ITHACA Object-Oriented Methodology Manual – Application Development Manual**. Milano: Politecnico di Milano, 1991. (ITHACA ESPRIT Project).

[DEA91b] DeANTONELLIS, V.; PERNICI, B; SAMARATI, P. Object-Orientation in the analisys of work organization and agent cooperation. In: INTERNATIONAL CONFERENCE ON DYNAMIC ASPECTS IN INFORMATION SYSTEMS, 2., 1991, Washington, DC. **Proceedings…** Washington: [s.n], 1991.

[DEA 91c] DeANTONELLIS, V.; PERNICI, B; SAMARATI, P. F-ORM Method: a F-ORM Methodology for reusing specifications. In: ASSCHE F. V.;MOULIN, B.; ROLLAND, C. (Eds.). **Object Oriented Approach in Information Systems**. Amsterdam: North-Holland, 1991. p.117-35.

[EDE 93] EDELWEISS, N.; OLIVEIRA, J.P.M.; PERNICI, B. An Object-Oriented Temporal Model. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING – CAISE, 5., 1993, Paris, France. **Proceedings…** Heidelberg: Springer-Verlag, 1993. p.397-415. (Lecture Notes in Computer Science, 685).

[EDE 94a] EDELWEISS, N. **Sistemas de Informações de Escritórios**: um modelo para especificações temporais. Porto Alegre: CPGCC da UFRGS, 1994. Tese de Doutorado.

[EDE 94b] EDELWEISS, N.; OLIVEIRA J. P. M. de. **Modelagem de Aspectos Temporais de Sistemas de Informação**. Recife: UFPE-di, 1994. 163p. Trabalho apresentado na Escola de Computação, 9., 1994, Recife.

[EDE 98] EDELWEISS, N. Banco de Dados Temporais: Teoria e Prática. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 17., 1998, Belo Horizonte. **Anais…** Belo Horizonte: SBC, 1998. p.225-282.

[ELM 93] ELMASRI, R.; WUU, G. T. J.; KOURAMAJLAN, V. A temporal model and query language for EER Databases. In: TANSEL, A. et al. (Eds.). **Temporal databases theory, design and implementation**. Redwood City: The Benjamin/Cummings, 1993. p. 212-229.

[GAD 93] GADIA, S.; NAIR, S. Temporal databases: a prelude do parametric data. In: TANSEL, A. et al. (Eds.). **Temporal databases** – theory, design and implementation. Redwood City: The Benjamin/Cummings, 1993. p. 28- 66.

[HEU 96] HEUSER C.; ANTUNES D. ER-Tr Diagrams: Conceptual Specification of transations in OO systems. In: JORNADAS DE TRABAJO EN INGENIERIA DEL SOFTWARE, 1., 1996, Sevilla, ES. **Actas**… Sevilha: Departamento de Lenguajes y Sistemas Informaticos, Universidad de Sevilha, 1996. p.1-10.

[JEN 98] JENSEN, C. S. et al. The Consensus Glossary of Temporal Database Concepts. In: ETZION, O; JAJODIA, S.; SRIPADA, S. (Eds.). **Temporal Databases Research and Practice**. Berlin: Springer-Verlag, 1998. p. 367-405.

[JUN 95] JUNGCLAUS R. et al. TROLL – A Language for Object-Oriented Specification of Information Systems. **ACM Transactions on Informations Systems,** New York, v. 14, n. 2, p. 175-211, 1995.

[KOW 85] KOWALSKI, R.; SERGOT, M. **A Logic-based Calculus of Events**. London: University of London, 1985. [S.l.]: OHMSHA, LTD and Springer-Verlag, 1986. p. 67-95.

[LAR 97] LARMAN, C. **Appliying UML and Patterns**: An Introduction to Object-Oriented Analysis and Design. New Jersey: Prentice Hall, Inc., 1997.

[LET 98a] LETELIER P. et al. **Formalización de OASIS en Lógica dinámica incluyendo especificaciones de proceso**. Valência: Universidad Politécnica

de Valência, 1998. (Informe técnico DSIC-II/2/98).

[LET 98b] LETELIER, P. et al. **OASIS version 3.0**: Um enfoque formal para o modelo conceitual orientado a objetos. Valência: Universidad Politecnica de Valencia, Serviço de publicaciones, 1998.

[LOU 91] LOUCOPOULOS, P.; THEODOLULIDIS, C.; WANGLER, B. The entity relationship time model and conceptual rule language. In: INTERNACIONAL CONFERENCE ON THE ENTITY RELATIONSHIP APPROACH, 10., 1991, San Mateo. **Proceedings...** San Mateo: The E/R Institute, 1991.

[LOR 88] LORENTZOS, N.A.; JOHNSON, R.G. Extending relational algebra to manipulate temporal data. **Information Systems**, Oxford, v.13, n.3, p.289-296, 1988.

[MAI 91] MAIOCCHI, R.; PERNICI, B. Temporal Data Management In Real-Time Systems: A comparative View. **IEEE Transactions on Knowledge and Data Engineering**, New York, Sept. 1991.

[MEN 99] MENESES, C. A. et al. Entorno Automáticos de Produccion de Software a partir de Modelos Conceptuales orientados a Objetos y Temporales. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 25., 1999, Assuncion PY. **Memorias...** Assuncion: Universidad Autonoma de Assuncion, 1999.

[MEY 97] MEYER, Bertrand. **Object-Oriented Software Construtions**. Upper Saddle River, New Jersey: Prentice Hall PRT, 1997.

[MEJ 88] MEYER, J.-J. Ch. A different approach to deontic logic. Deontic logic viewed as a variant of dynamic logic. **Notre Dame Journal of Formal Logic**, [S.l.], v. 29, p. 109-136, 1988.

[PAS 92a] PASTOR, O. **OO-METHOD**: An Object Oriented Methodology for Software Production. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 1992. **Proceedings...** [S.l.]: Springer-Verlag, 1992. p. 121-127.

[PAS 92b] PASTOR, O. **Diseno y Desarrollo de un Entorno de Produccion Automatica de Software basado en el modelo OO**. Valencia: Departament de Sistemes Informàtics i Computació, Universitat Politècnica de Valencia,1992. Tesis Doctoral.

[PAS 95a] PASTOR, O.; SALVERT I. **OASIS version2 (2.2**): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach. Valencia: Universidad Politecnica de Valencia, Serviço de Publicaciones, 1995.

[PAS 95b] PASTOR, O.; SALAVERT, I; CANOS, J.H. **OASIS v2:** A Class Definition Language. In: INTERNATIONAL CONFERENCE ON DATABASE AND

EXPERT SYSTEMS APPLICATIONS, DEXA, 1995. **Proceedings...** [S.l.]: Springer-Verlag, 1995.

[PAS 96] PASTOR, O. et al. An OO Methodological Approach for Making Automated Prototyping Feasible. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 1996. **Proceedings...** [S.l.]: Springer-Verlag, 1996.

[PAS 97a] PASTOR, O. et al. Ingemieria de Ambientes Software: Combinando Expresividades Temporales Y Objetuales En La Fase de Modelizacion Conceptual. In: JORNADAS DE TRABAJO EN INGENIERIA DEL SOFTWARE, San Sebastian, Spain, 1997. **Proceedings...** [S.l.:s.n.], 1997.

[PAS 97b] PASTOR, O. et al. **OO-METHOD**:An OO Software Production Environment Combining Conventional and Formal Methods. [S.l.:s.n.], 1997. p.145-158.

[PAS 98] PASTOR, O. et al. Especificacion de Temporalidad en Modelos Conceptuales Orientados a Objetos. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 1999, Quito, Equador . **Memorias...** [S.l:s.n.], 1999.

[PER 90] PERNICI, B. Objects with Roles. **SIGOIS Bulletin**, New York, v.11, n. 2-3, p.205-15, 1990. Trabalho apresentado na ACM/IEEE CONFERENCE ON OFFICE INFORMATION SYSTEMS.

[PRA 91] PRAKASH, N. Specifying Operational Characteristics of Information Systems In OOD. In: VAN ASSCHE, F.; MOULIN, B.; ROLLAND, C. (Eds.). **Object Oriented Approach in Information Systems**. Amsterdam: Elsevier, 1991.

[RAM 92] RAMOS, I. et al. On the use of Algebras as Semantic Domains of Object Societies. In: INTERNACIONAL WORKSHOP OF THE DEDUCTIVE APPROACH FOR DB AND IS, 3., 1992, Roses ,Catalania. **Procedings...** [S.l.:s.n.], 1992.

[RAM 96] RAMAMRITHAM, K; SIVASANKARAN, R.; STANKOVIC, J. A. et al. Integration Temporal, Real-Time, and Active Databases. **SIGMOD Record**, New York, v. 25, n.1, Mar. 1996.

[SAR 90] SARDA, N.L. Extensions to SQL for historical databases. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.2, n.2, p. 220-230, June 1990.

[SEG 88] SEGEV, A; SHOSCHANI, A. Modeling temporal semantics. In: ROLLAND, C; BODART, F.; LEONARD, M. (Eds.). **Temporal Aspects in Information Systems**. Amsterdam: North-Holland, 1988. p.47-57.

[SEG 93] SEGEV, A.; SHOSCHANI, A. A temporal data model based on time Sequences. In: TANSEL, A. et al. (Eds.). **Temporal databases**: Theory, design and implementation. Redwood City: The Benjamin/Cummings, 1993. p.248-270.

[SER 92] SERNADAS A. et al. **Oblog**: Object-Oriented, logic-based conceptual modelling. Lisboa: Instituto Superior Técnico, Secção de Ciência da Computação, Departamento de Matemática, 1992. (Research Report, 1096).

[SHI 81] SHIPMAN, D. W. The funcional data model and data language DAPLEX. **ACM Transactions on Database Systems**, New York, v.6, n.1, p.140-173, Mar. 1981.

[SNO 87] SNODGRASS, R. The Temporal query language TQuel. **ACM Transactions on Database Systems**, New York, v.12, n.2, p.247-298, June 1987.

[SU 91] SU, Y. W. S.; CHEN, H.-H. M. A temporal knowlegde modelo OSAM*/T and its query language OQL/T. INTERNATIONAL CONFERENCE OF VERY LARGE DATABASES, 17., 1991, Barcelona. **Proceedings…** [S.l.:s.n.], 1991.

[SU 93] SU, Y. W. S.; CHEN, H.-H. M. Modeling and Management of temporal data in object-oriented knowlegde bases**.** In INTERNATIONAL WORKSHOP ON AN INFRASTRUCTURE FOR TEMPORAL DATABASES, 1993, Arligton, Texas. **Proceedings…** [S.l.:s.n.], 1993. p.HH-1-HH-18.

[TAN 86] TANSEL, A.U. Adding time dimension to relational model and extending relational algebra. **Information Systems**, New York, v.11, n.4, p.343-355, 1986.

[TAN 93] TANSEL, A. et al. (Eds.). **Temporal Databases:** Theory, Design and Implementation. [S.l.]: The Benjamim/Cummings , 1993.

[TAN 97] TANSEL, A. U.; TIN, E**.** The Expressive power for temporal relational query languages. **IEEE Transactions on Knowlegde and Data Engineering**, New York, v.9, n.1, Jan. 1997.

[TAU 91] TAUZOVICH, B.Towards temporal extensions to the entity-relationship model. In: INTERNACIONAL CONFERENCE ON THE ENTITY RELATIONSHIP APPROACH, 10.,1991, San Mateo. **Procedings...** San Mateo, California, 1991.

[WUU 93] WUU, G. T. J.; DAYAL, U. A Uniform model for temporal and versioned objetc-oriented databases. IN: TANSEL, A. et al. (Eds.). **Temporal databases:** theory, design and implementation. Redwood city: The Benjamim/Cummings , 1993. p. 230-247.

[WAN 96] WANG, L. et al. An Algebra for a Temporal Object Data Model. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 1996. **Proceedings...** [S.l.:n.s.], 1996.

[ZAN 9?] ZANIOLO, C. et al. **Advanced Database Systems**. San Francisco, CA: Morgan Kaufmann Publishers, [199?]. p.110