ESPECIFICAÇÃO DE REQUISITOS: UTILIZANDO UM MODELO TEMPORAL ORIENTADO A OBJETOS

Nina Edelweiss*
José Palazzo M. de Oliveira*
José Mauro V. de Castilho*
Barbara Pernici+

* Universidade Federal do Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves, 9500 - Bloco IV - Agronomia
Caixa Postal 15064
91501 - Porto Alegre - RS
e-mail: nina@inf.ufrgs.br

+ Università degli Studi di Udine
Udine - Itália

RESUMO

O paradigma de orientação a objetos apresenta características apropriadas para sua utilização em especificações de requisitos. A representação das características dinâmicas de uma aplicação requer a representação de propriedades temporais, aspecto este não muito explorado nos métodos orientados a objetos. O objetivo deste artigo é apresentar as extensões feitas em um modelo de dados orientado a objetos, o modelo F-ORM [DeAntonellis 91] para permitir a especificação de aspectos temporais. O modelo obtido permite a representação de tempos de transação e de tempos válidos, utilizando para isto pontos no tempo e uma linguagem de lógica temporal. Apresenta, ainda, um conjunto de tipos de dados temporais e operações associadas.

ABSTRACT

Object-oriented approaches have appropriate characteristics for requirements specifications. An application's dynamic characteristics representation requires the possibility of representing temporal properties, not well explored in object-oriented methods. In this paper the main concern is to present the extensions made on an Object-Oriented Model, the F-ORM model [DeAntonellis 91] to support temporal aspects. The resulting model may be used to represent transaction and valid times, using timestamps and a temporal logic language. A set of temporal data types and their associate operations is also available.

1. INTRODUÇÃO

O paradigma de orientação a objetos apresenta características apropriadas para sua utilização em especificação de requisitos. Muitos trabalhos têm sido realizados utilizando este paradigma na representação de propriedades estáticas e comportamentais de sistemas de informação. A representação das características dinâmicas de uma aplicação requer a representação de propriedades temporais, aspecto este pouco explorado nos métodos orientados a objetos. Estudos relacionados a estes aspectos podem ser encontrados em [Arapis 91, Clifford 88a, Greenspan 86].

Os modelos orientados a objetos devem possibilitar a definição de propriedades temporais, principalmente se forem utilizados como métodos de especificação de sistemas onde o fator tempo é crítico, tais como sistemas de controle de plantas industriais e sistemas de informação de escritórios. O objetivo deste trabalho é desenvolver um modelo que possa ser utilizado na especificação formal de sistemas de informação. A representação de aspectos temporais é necessária neste tipo de aplicações para possibilitar a representação da evolução dinâmica dos objetos no ambiente modelado pelo sistema de informação. Propriedades temporais são utilizadas para definir características com valores temporais de entidades (atributos).

Pesquisas recentes [Arapis 91, Gabbay 91, Maiocchi 91a, Wiederhold 91] apresentam diferentes formas para representar o tempo. A definição de tempo pode ser feita de forma explícita, geralmente através da associação de um instante (ponto) de tempo a uma informação (timestamping), ou de forma implícita através da utilização de uma linguagem de lógica temporal. Para a representação explícita de tempo é necessária a definição de uma entidadea temporal primitiva, como por exemplo, pontos no tempo ou ontervalos.

Quando a entidade temporal primitiva é *ponto no tempo*, os eventos são representados ao longo de um eixo temporal, sendo o instante atual representado por um ponto particular que se move constantemente ao longo deste eixo. Em muitas aplicações surge a necessidade de definir diferentes granularidades para as informações temporais, tais como horas, dias e anos. A utilização de diferentes granularidades aumenta a complexidade dos mecanismos de recuperação das informações, mas dá origem a especificações bem mais fiéis da realidade.

Os *intervalos* são definidos por pares de pontos no tempo, representando os limites inferior e superior do intervalo. A representação de um ponto no tempo é feita através de um intervalo mutio pequeno.

Um importante enfoque para representação temporal é a ágebra intervalar de Allen [Allen 83]. Os eventos são representados por intervalos de tempo. Estes intervalos são relacionados entre si através de relacionamentos temporais, descritos por predicados de uma linguagem de lógica temporal. As linguagens TELOS [Mylopoulos 90] e RML [Greenspan 86] se baseiam nesta teoria.

A utilização de lógica temporal para especificação de propriedades temporais é encontrada em diversos sistemas e linguagens [Bolour 83, Corsetti 91, Greenspan 86, Loucopoulos 91, Maiocchi 91b, Schiel 83]. Esta forma de representação foi muito influenciada pelo Cálculo de Eventos [Kowalski 86]. A mais importante contribuição deste enfoque é a possibilidade de tratar de informações incertas e imprecisas, tais como *antes* e *depois*.

O objetivo deste artigo é apresentar as extensões feitas em um modelo de dados orientado a objetos, o modelo F-ORM [DeAntonellis 91], com a finalidade de possibilitar a especificação de propriedades temporais. Foram acrescentados a este modelo quatro diferentes conceitos para definições temporais: (1) um conjunto de tipos de dados temporais juntamente com suas funções e operações associadas, para ser utilizado na definição de domínios de propriedades; (2) pontos de tempo (*timestamps*) associados a instâncias dos objetos e a propriedades dinâmicas; (3) um valor especial *null* representando o estado de propriedades fora do período de validade; e (4) condições temporais associadas a regras de transição de estados.

Snodgrass and Ahn propuseram uma taxonomia para o tempo em bancos de dados [Snodgrass 85] que consiste basicamente de três conceitos temporais distintos: (1) tempo de transação, o tempo no qual é feita a atualização do banco de dados; (2) tempo válido, representado o período de validade da informação armazenada; e (3) tempo definido pelo usuário, consistindo de propriedades temporais definidas explicitamente pelos usuários em um domínio temporal e manipuladas pelos programas de aplicação.

Nas extensões feitas no modelo F-ORM, a definição de um conjunto de tipos de dados temporais tem por objetivo diminuir a necessidade de definição de tempos definidos pelo usuário. Instantes de tempo associados às instâncias e o valor *null* permitem a representação dos tempos válidos. Através dos instantes de tempo associados às propriedades dinâmicas são representados os tempos de transação. As condições temporais restringem o conjunto de possíveis transações, agindo como regras de integridade dinâmicas.

Este artigo está organizado em 6 seções. Na seção 2 é apresentado um pequeno caso de aplicação, utilizado nas ilustrações das seções seguintes. Os aspectos principais dos modelos ORM e F-ORM [Pernici 90, DeAntonellis 91] são apresentados na seção 3. Na seção 4 são apresentados os tipos de dados temporais, incluindo as funções associadas. A seção 5 descreve a representação de tempos de transação, incluindo o valor *null* utilizado para representar estados indefinidos. A linguagem de lógica temporal utilizada nas condições temporais é vista na seção 6.

2. EXEMPLO DE APLICAÇÃO

Os exemplos deste artigo utilizam como aplicação a especificação parcial de uma locadora de fitas de vídeo: informações referentes a clientes, funcionários, fitas e locações. A especificação parcial, através do modelo F-ORM estendido com os aspectos temporais propostos neste artigo, está no Anexo 1.

Um *cliente* é identificado através de um código único, de seu nome e de seu endereço. Informações adicionais podem ser necessárias, tais como a data de sua inscrição na agência de locação, todas as fitas que já alugou, períodos de locação correspondentes, e se está habilitado a alugar fitas. Um *funcionário* é identificado por seu nome, apresentando propriedades complementares tais como: endereço, data de sua contratação e dispensa, e salário. As *fitas* são identificadas por um código único. As informações complementares de uma fita são: o nome do filme, o tipo deste filme e a data de sua aquisição. Uma *locação* é definida pela associação do código da fita com o código do cliente e com a data inicial desta locação. Uma locação só é permitida se determinadas condições são satisfeitas: a fita está disponível; o cliente deve estar habilitado a alugar fitas e não estar de posse de alguma fita a mais de 30 dias nem de mais de 5 fitas ao mesmo tempo.

3. O MODELO F-ORM

O modelo F-ORM (Functionality in Object with Roles Model) [DeAntonellis 91] é um modelo orientado a objetos para ser utilizado em especificações de sistemas de informação de escritórios[1]. É uma extensão do modelo ORM (Object with Roles Model) [Pernici 90]. Os dois modelos descrevem o comportamento de objetos através do conceito de papéis (*roles*).

No modelo ORM uma classe é definida por um nome cn e por um conjunto de papéis R_i, cada papel representando um comportamento diferente deste objeto:

$$class = (cn, R_0, R_1, \ldots, R_n)$$

Cada papel R_i consiste do *nome do papel* Rn_i, de um conjunto de *propriedades* P_i deste papel (descrições abstratas dos tipos de dados implementados como variáveis de instância), de um conjunto de *estados abstratos* S_i que o objeto pode apresentar neste papel, de um conjunto de *mensagens* M_i que o objeto pode receber e enviar neste papel, e de um conjunto de *regras* Ru_i (regras de transição de estado e regras de integridade):

$$R_i = < Rn_i, P_i, S_i, M_i, Ru_i >$$

As instâncias dos papéis evoluem independentemente umas das outras, podendo haver interações através de trocas de mensagens. Um objeto pode desempenhar diferentes papéis em tempos diferentes, pode desempenhar simultaneamente mais de um papel e pode apresentar diversas instâncias do mesmo papel.

Todo objeto apresenta um *papel básico* R_0 que descreve as características iniciais de uma instância e as propriedades globais que controlam sua evolução. As propriedades deste papel básico aplicam-se a todos os demais papéis; suas mensagens são utilizadas para acrescentar, suprimir, suspender e reativar papéis; seus possíveis estados são pré-

---

[1] Projeto INFOKIT do Conselho Nacional Italiano de Pesquisa e projeto ITHACA - ESPRIT II.

definidos, representando se o objeto está ativo (*active*) ou suspenso (*suspended*); suas regras definem transições entre papéis e restrições globais para a classe. Na definição de propriedades é definido o domínio dos dados que cada propriedade pode apresentar.

Dois tipos de regras são utilizados em ORM: regras de transição de estados e regras de integridade. Uma regra de transição de estados representa uma transição válida entre dois estados, dependendo eventualmente da ocorrência de uma mensagem. A transição pode causar o envio de outra mensagem. Uma regra de integridade deve sempre ser satisfeita por todas as instâncias de um papel.

Uma classe pode ser definida como subclasse de uma ou mais classes (herança múltipla). A subclasse herda todos os componentes especificados para as classes da qual deriva. Novos componentes podem ser acrescentados à definição da subclasse de duas maneiras: (1) acrescentando especificações de novos papéis; e (2) modificando a especificação dos papéis herdados.

Considerando a classe *fita* (*tape*) da aplicação proposta, as propriedades do papel básico são o código da fita, o nome do filme e o tipo do filme (drama, comédia, etc.). Os seguintes papéis podem ser identificados para esta classe: (1) *Locações* (*Rentals*), representado as possíveis locações desta fita; (2) *Tempo_de_vida* (*Life_time*), representando as ações a serem executadas para comprar uma fita, cadastrá-la na agência, colocá-la em disponibilidade para locação durante um certo período de tempo e vendê-la após este período; e (3) *Perda_da_fita* (*Tape_loss*), representando as ações a serem tomadas quando uma fita for extraviada.

Considerando o papel *Locações*, algumas das propriedades que devem ser definidas são: o código do cliente e as datas de início e de final de uma locação. Possíveis estados deste papel são *disponível* (*available*) e *alugada* (*rented*). As mensagens que podem ser enviadas e recebidas são as seguintes: *locação* (*rental*) vinda do controle de locações (*Rental_control*), *devolução_de_fita* (*tape_devolution*) também do controle de locações e *tempo_de_locação* (*rented_time*) que este papel envia para o controle de locações. As regras de transição de estado controlam o comportamento. Utilizando o símbolo "<-" para mensagens recebidas o "->" para as enviadas, a definição desta classe é a seguinte:

```
class (
  TAPE,
  < base_role,
    properties = [   (tape_number, INTEGER),
                     (tape_film, STRING),
                     (film_type, STRING) ]
    rules = [ ],
  >,
  < Life_time,
    ...
  >,
  < Rentals,
    properties = [   (client_code, INTEGER),
                     (beginning_date, DATE),
```

```
                    (end_date, DATE) ),
    messages = (    rental from Rental_control,
                    tape_devolution from Rental_control,
                    rented_time to Rental_control ),
    states = ( available, rented ),
    rules = (   rule1 : msg(<-add_role) => state(available),
                rule2 : state(available), msg(<-rental) => state(rented),
                rule3 : state(rented), msg(<-tape_devolution) =>
                                        msg(->rented_time), state(available) )
>,
< Tape_loss,
  ...
> )
```

O modelo F-ORM estende o modelo ORM incorporando conceitos apropriados à especificação das funcionalidades de sistemas de informação de escritórios. Decompõe as possíveis classes em dois tipos distintos, pré-definidos: *classes de recursos* e *classes de processos*. Uma classe do tipo recurso define a estrutura de um recurso (agente, dado ou documento) em termos dos papéis que este recurso pode apresentar durante seu ciclo de vida, com propriedades, mensagens permitidas, estados internos abstratos e correspondentes regras de transição. As classes do tipo processo integram as classes de recursos, permitindo a descrição do trabalho realizado no escritório em termos de sua organização e da cooperação entre os agentes envolvidos. Uma classe do tipo processo também é descrita através de papéis com propriedades, mensagens, estados e regras. Neste caso o conceito de papel é utilizado para especificar as diferentes tarefas executadas no processo e seus relacionamentos em termos de regras de comunicação e de cooperação, juntamente com os recursos envolvidos.

No exemplo da agência de locação de fitas de vídeo, as classes que representam as pessoas (clientes e funcionários) e as fitas constituem classes de recursos enquanto que as locações constituem uma classe do tipo processo. A definição destas classes está ilustrada no Anexo 1, utilizando as extensões propostas neste artigo para incluir a representação de aspectos temporais.

4. TIPOS DE DADOS TEMPORAIS

Uma das extensões realizadas no modelo F-ORM foi a definição de um conjunto de tipos de dados temporais a ser utilizado na definição de propriedades. Estes tipos de dados apresentam diferentes granularidades temporias, tais como hora, ano e intervalo. As diferentes granularidades são necessárias para que se possa especificar a realidade de uma maneira natural através de conceitos utilizados na prática.

O modelo F-ORM apresenta os seguintes domínios pré-definidos: BOOLEAN, DATE, IMAGE, INTEGER, PLACE, STRING, TEXT, TIME, TITLE. Dois tipos de dados temporais, a data (DATE) e o tempo (TIME), já se encontram definidos neste conjunto. Entretanto, dependendo da aplicação a ser especificada, outros tipos temporais se fazem necessários.

Quatro diferentes tipos de dados temporais podem ser identificados [Adiba 85,87]: pontos no tempo, intervalos, duração e período. Decidimos introduzir somente os três primeiro tipos no modelo, considerando que o quarto pode ser definido através de regras de restrições aplicadas a intervalos. A seguir serão discutidos os tipos definidos para representar cada um destes dados temporais.

4.1. PONTOS NO TEMPO

Foi selecionado a entidade primitiva temporal *ponto no tempo*, sendo a menor granularidade o *minuto*. Conseqüentemente, a definição completa de um ponto no tempo requer a definição de uma *data* (ano, mês e dia) e do *tempo* dentro desta data (hora e minuto). Isto é feito através da definição do tipo temporal básico INSTANT (instante). Usando uma notação BNF simplificada, o tipo temporal básico é composto por:

<instante> ::= <dia> "/" <mês> "/" <ano> <hora> ":" <minuto>

Os tipos de dados temporais pré-definidos do modelo F-ORM, DATE e TIME, são considerados aqui como tipos derivados:

<data> ::= <ano>"/" <mês> "/" <dia>
<tempo> ::= <hora> ":" <minuto>

Foi definido um conjunto de tipos temporais, derivados do tipo temporal básico, para representar informações temporais específicas. São eles: YEAR (ano), MONTH (mês), DAY (dia), HOUR (hora) e MINUTE (minuto). Outros tipos se fazem necessários para especificar a realidade de uma maneira natural, representando intervalos especiais considerados como pontos no tempo: WEEK (semana) e SEMESTER (semestre). O conjunto de tipos temporais que representam pontos no tempo é completado com o tipo WEEKDAY que define o dia da semana de uma informação temporal (domingo a sábado).

4.2. INTERVALOS

Intervalos de tempo são utilizados para definir todos os instantes entre dois pontos de tempo. Os limites devem apresentar a mesma granularidade temporal, a qual define a unidade de tempo para pontos dentro do intervalo. No caso dos limites de um intervalo serem do tipo INSTANT, sua definição seria dada por:

<intervalo> ::= <instante> ":" <instante>

Quatro tipos diferentes de intervalos podem ser definidos, dependendo da pertinência ou não dos pontos limites ao intervalo: *intervalos fechados* quando ambos os limites pertencem ao intervalo, *intervalos semiabertos* quando somente um dos limites pertence ao intervalo, *intervalos abertos* quando nenhum dos limites pertence ao intervalo e *intervalos flutuantes* nos casos em que um dos limites é representado pelo momento atual.

4.3. DURAÇÃO

Outro tipo de dado muito utilizado na especificação de sistemas é a duração de uma atividade. Esta informação é representada por um número inteiro sem sinal seguido de uma unidade de tempo apropriada - dias, horas, semanas. Um possível tipo duração é:

<duração> ::= <inteiro> MONTHS <inteiro> DAY

4.4. FUNÇÕES E OPERAÇÕES PARA OS TIPOS DE DADOS

A utilização de tipos de dados com diferentes granularidades provoca algumas dificuldades na manipulação e operação com tempos diferentes [Clifford 88b, Wiederhold 91]. Para que esta manipulação seja possível foi definido um conjunto de funções (predicados). A especialização de classes com herança de propriedades e de mensagens permite a definição de funções para instantes (tipo temporal básico) as quais podem ser especializadas para os tipos data e tempo, conforme a necessidade. Alguns exemplos das **funções** definidas são:

year(<instante>)	- trunca um instante para o ano
month(<ano/ mês>)	- trunca o ano, retornando o mês
weekday(<data>)	- retorna o dia de semana correspondente à data
month(<instante>)	- extrai o mês de um instante
begin(<intervalo>)	- fornece o limite inferior de um intervalo
duration(<intervalo>)	- calcula a duração de um intervalo
before(<instante>:<instante>)	- verdadeiro quando o primeiro instante for anterior ao segundo
equal(<instante>:<instante>)	- verdadeiro quando os dois instantes forem iguais
contains(<intervalo>,<intervalo>)	- verdadeiro quando o primeiro intervalo estiver contido no segundo

Foram também definidas **operações** envolvendo os diferentes tipos temporais:

- operações aritméticas, tais como soma e subtração, que podem ser aplicadas: (1) quando os dois operandos são do tipo duração, resultando em um valor do mesmo tipo; (2) quando o primeiro operando é do tipo instante (INSTANT), data (DATE) ou tempo (TIME) e o segundo do tipo duração, resultando um valor do mesmo tipo do primeiro operando; (3) quando os dois operando são intervalos de mesma granularidade temporal, resultando um outro intervalo ou um valor indefinido.

- operações sobre conjuntos operando sobre intervalos, tais como *union* (união), *intersection* (intersecção) e *ownership* (pertinência), resultando intervalos, valores indefinidos ou valores lógicos.

Como exemplo de definição de propriedades através de tipos de dados temporais podemos citar o início e o final de uma locação:

```
Rentals,
dynamic properties = [  (client_code, INTEGER),
                        (beginning_date, DATE),
                        (end_date, DATE)
                     ]
```

5. REPRESENTAÇÃO DE TEMPOS DE TRANSAÇÃO E VÁLIDOS

Como definido em [Snodgrass 85], dois tipos diferentes de tempos podem ser representados - *tempos de transação*, correspondente ao tempo em que uma informação é registrada, e *tempos válidos*, correspondendo ao tempo em que a informação modela a realidade.

5.1. TEMPO DE TRANSAÇÃO

A definição dos tempos de transação segundo o paradigma de orientação a objetos pode ser feito através de dois diferentes enfoques. O primeiro associa o instante de definição a uma instância de um objeto. O segundo associa o instante de criação a cada uma de suas propriedades.

Na extensão do modelo F-ORM, a associação de tempo a uma instância é realizada implicitamente no momento de criação da referida instância. Este tempo servirá de referência para o objeto durante toda a sua existência, como um *surrogate* [Codd 79] ou uma *essência* [Clifford 88a]. Este valor será armazenado em uma propriedades estática especial do papel básico, denominada *creation* (criação). A existência de uma instância inicia neste instante, podendo apresentar períodos de validade disjuntos.

Analisando as possíveis propriedades que um papel pode apresentar identificamos algumas que nunca mudam, como por exemplo, o CIC uma pessoa. Estas propriedades são denominadas de *propriedades estáticas*. Somente um valor pode ser definido para uma propriedades estática, permanecendo o mesmo durante toda a existência da instância.

Para representar propriedades cujos valores podem variar com a passagem do tempo foi definido um novo tipo de propriedades, denominadas *propriedades dinâmicas*. Uma propriedade dinâmica consiste de um conjunto de pares mapeando o tempo de definição ao tipo de dados da propriedade, como proposto em [Clifford 88a]. O domínio temporal destes pares é sempre o mesmo, dado pela concatenação da data com o tempo. Operações apropriadas podem ser utilizadas na linguagem de recuperação de informações e nas regras para comparar estes valores e para extrair informações específicas (tais como ano, mês, hora, dia de semana). A história completa de uma instância pode ser recuperada através das propriedades dinâmicas. Os tempos de transação são associados às propriedades dinâmicas através destes rótulos temporais.

Na definição do papel *empregado* da classe *PESSOA* encontramos exemplos de propriedades estáticas e dinâmicas:

```
Employee,
        static properties = { (nome, STRING) },
        dynamic properties = { (salary, REAL), (hire_date,DATE), (out_date,DATE) }
```

Para representar os períodos de tempo durante os quais propriedades dinâmicas apresentam valores indefinidos foi definido um valor especial *null*. Imediatamente após a criação de uma instância de um papel, todas as suas propriedades recebem um valor default *null*. Para as propriedades dinâmicas, este valor é mantido até o momento em que um novo valor é definido. No final do período de validade do novo valor a propriedade passa a valer novamente *null*. Todas as variações de valores de uma propriedade dinâmica são rotuladas com os correspondentes instantes de definição. Propriedades estáticas também recebem o valor *null* no momento de criação da instância, sendo permitida somente uma alteração deste valor - o novo valor definido será mantido durante toda a existência da instância.

5.2. TEMPO VÁLIDO

A representação de tempos válidos é feita através da linguagem de recuperação e manipulação de informações, a qual deve apresentar dois argumentos especiais utilizados em mensagens que modificam as propriedades dinâmicas: STARTING_TIME (tempo inicial de validade) e FINAL_TIME (tempo final). Estes argumentos são opcionais, devendo ser utilizados somente quando o tempo válido é diferente do de transação.

Como exemplo de aplicação na qual o tempo de transação pode ser diferente do tempo válido consideremos a atualização de salário de um empregado. Consideremos, por exemplo, que no dia 5 de maio (tempo de transação) foi definido um novo salário para um funcionário. Este valor de salário corresponde ao mês de maio todo, ou seja, já era válido no dia 1 de maio (tempo válido). Duas mensagens possíveis para este exemplo são:

```
messages = { modify_salary( Value:REAL, STARTING_TIME:DATE) from employee_control,
             end_employment (FINAL_TIME:DATE) from employee_control }
```

6. CONDIÇÕES TEMPORAIS

As regras de transição de estados do modelo F-ORM controlam as transições válidas entre pares de estados. Uma transição pode depender da ocorrência de uma mensagem e pode ocasionar o envio de outra mensagem. Aplicações que envolvem aspectos temporais podem necessitar da definição de condições de integridade temporal - condições que comparam dois estados diferentes das informações. Para possibilitar a definição deste tipo de condição de integridade foi acrescentada uma condição às regras de transição. Deste modo é possível comparar tempos diferentes de informações. Esta condição é avaliada no instante em que a transição está sendo ativada, sendo que a

transição somente será executada se a condição for verdadeira. A regra de transição de estados estendida tem a seguinte forma:

<regra de transição de estado> ::= <identificação da regra> ":"
<estado1> "," <mensagem1> "=>" <mensagem2> "," <estado2> ";" <condição>

A condição temporal é escrita em uma linguagem de lógica temporal. A lógica temporal é uma especialização da lógica modal - enquanto o domínio de interpretação da lógica modal é um conjunto genérico de estados e as relações entre estes estados, a lógica temporal requer que estes estados formem uma seqüência discreta linear [Manna 81]. Uma seqüência discreta de dois estados pode ser utilizada para descrever as mudanças dinâmicas em instantes discretos.

Através da utilização de lógica temporal podemos representar situações que variam com a passagem do tempo. Assumimos que a variação do tempo é discreta, apresentando um passado linear e permitindo ramificações no futuro. Formalismos lógicos tem sido largamente utilizados para expressar requisitos que envolvem tempo e para especificar aplicações dinâmicas [Castilho 82, Carmo 88, Finger 91, Gabbay 91, Lipeck 87, Segev 88]. Uma das vantagens da utilização deste formalismo é a possibilidade de representar informações incompletas e indefinidas através da utilização de operadores temporais tais como *desde* e *até*.

Os símbolos que poderão ser utilizados nas fórmulas das condições temporais são: (1) proposições atômicas, fazendo referência a valores de propriedades estáticas e dinâmicas; (2) operadores relacionais; (3) conectivos lógicos *and*, *or* e *not*; (4) quantificadores existencial *exists* e universal *forall*; (5) valores transmitidos como argumento pelas mensagens recebidas; e (6) um conjunto de operadores temporais. Como estamos associando uma condição a um regra de transição entre dois estados, necessitamos comparar somente o momento atual a momentos do passado. Para isto foi definido o seguinte conjunto de operadores temporais:

Operador	Semântica
sometime past A	*A* valeu em algum momento do passado
immediately past A	*A* valeu no momento anterior
always past A	*A* valeu em todos os momentos do passado

A regra de transição de atualização do salário de um funcionário mostra como uma condição pode ser expressa. Existe uma lei que diz que um salário nunca pode diminuir. A condição abaixo controla que esta lei seja sempre satisfeita:

ri: state(employed), msg(modify_salary(V)) => state(employed) ;
immediately past exists V1 (salary(V1) and V>V1)

7. CONCLUSÃO

Sistemas sócio-técnicos compõem uma categoria de aplicações para as quais a representação de aspectos temporais é um requisito essencial. A forte interação entre atividades humanas e automatizadas faz com que a *especificação de características temporais* seja fundamental. Modelos orientados a objetos são uma boa opção para representar este tipo de interações. Entretanto, eles devem também possibilitar a definição de propriedades temporais, principalmente quando utilizados em especificações de sistemas de criticamente dependentes do tempo. Aspectos temporais são necessários para representar a evolução dinâmica dos objetos no ambiente de sistemas de informação.

O modelo ORM e sua extensão F-ORM são modelos orientados a objetos para os quais foram pré-definidos os domínios temporais DATE (data) e TIME (tempo), de modo a possibilitar a manipulação de tempos explícitos. Este trabalho descreveu uma extensão realizada no modelo F-ORM com o objetivo de permitir a especificação de aspectos temporais. Acrescentamos a este modelo quatro diferentes conceitos: (1) um conjunto de tipos de dados temporais e funções associadas, para ser utilizado na definição de domínios de propriedades; (2) pontos de tempo (*timestamps*) associados a instâncias dos objetos e a propriedades dinâmicas; (3) um valor especial *null* representando o valor de propriedades fora do período de validade; e (4) condições temporais acopladas a regras de transição de estados, escritas em uma linguagem de lógica temporal.

Está sendo implementado um ambiente de especificação de sistemas de informação temporais (extensão de ambiente YPY [Edelweiss 89]) que adota o modelo proposto associado a uma linguagem adequada para recuperação de informações. Este ambiente deverá ser utilizado na especificação de requisitos de sistemas de informação de escritórios, apresentando conhecimentos desta área de aplicação para auxiliar na construção das especificações. Uma biblioteca de classes específicas deste domínio auxiliará na reutilização de classes, armazenadas conforme o modelo proposto.

REFERÊNCIAS BIBLIOGRÁFICAS

[Adiba 85] ADIBA,M.; QUANG,N.B.; de OLIVEIRA,J.P.M. Time concept in generalized data bases. In: ACM ANNUAL CONFERENCE, Denver Oct. 14-16, 1985. **Proceedings**. New York, ACM, 1985. p.214-23.

[Adiba 87] ADIBA,M.; BUI QUANG,N.; COLLET,C. Aspect temporels, historiques et dynamiques des bases de données. **TSI - Technique et Science Informatiques**. AFCET-Bordas, 6(11):832-43, Nov. 1983.

[Allen 83] ALLEN,J.F. Maintaining knowledge about temporal intervals. **Communications of the ACM**. New York, 26(11):832-43, Nov. 1983.

[Arapis 91] ARAPIS,C. Specifying object interactions. TSICHRITZIS,D. (ed.) **Objects Composition**. Geneb ra, Université de Genève, 1991. p.303-22.

[Bolour 83] BOLOUR,A. & DEKEYSER,L.J. Abstractions in temporal informations. **Information Systems**, Great Britain, 8(1):41-9, 1983.

[Carmo 88] CARMO,J. & SERNADAS,A. A Temporal logic framework for a layered approach to systems specification and verification. In: ROLLAND,C.; BODART,F.; LEONARD,M. (eds.) **Temporal Aspects in Information Systems**. Amsterdam, North-Holland, 1988. p.31-46.

[Castilho 82] CASTILHO,J.M.V.; CASANOVA,M.A.; FURTADO,A.L. A Temporal framework for database specifications. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 8., Mexico City, Sept. 1982. **Proceedings**. Mexico City, 1982. p.280-91.

[Clifford 88a] CLIFFORD,J. & CROKER,A. Objects in time. **Data Engineering**, Washington, 11(4):11-18, Dec. 1988.

[Clifford 88b] CLIFFORD,J. & RAO,A. A Simple, general structure for temporal domains. In: ROLLAND,C.; BODART,F.; LEONARD,M. (eds.) **Temporal Aspects in Information Systems**. Amsterdam, North-Holland, 1988. p.17-28.

[Corsetti 91] CORSETTI,E. et al. Dealing with different time scales in formal specifications. INTERNATIONAL WORKSHOP ON SOFTWARE SPECIFICATION AND DESIGN, 6., Como, Italy, Oct. 25-6, 1991. **Proceedings**. IEEE Computer Society Press, 1991. p.92-101.

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

[Edelweiss 89] EDELWEISS,N. Um Ambiente para Desenvolvimento de Protótipos de Bancos de Dados Dedutivos Temporais. SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 4., Campinas, 5-7 abril 1989. **Anais**. Campinas, R. Vieira Gráfica e Editora, 1989. p.163-73.

[Finger 91] FINGER,M.; McBRIEN,P.; OWENS,R. Databases and executable temporal logic. IN: ESPRIT '91 ANNUAL ESPRIT CONFERENCE, Brussels, Nov. 25-29, 1991. **Proceedings**. Brussels, ECSC, 1991. p.289-302.

[Gabbay 91] GABBAY,D. & McBRIAN,P. Temporal logic & historical databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 17., Barcelona, Sept. 3-6, 1991. **Proceedings**. Barcelona, Industria Gratica, 1991. p.423-30.

[Greenspan 86] GREENSPAN,S.J.; BORGIDA,A.; MYLOPOULOS,J. A Requirements modeling language and its logic. In: BRODIE,M.L. & MYLOPOULOS,J. (eds.) **On Knowledge Base Systems**. Springer-Verlag, New York, 1986. p.471-502.

[Kowalski 86] KOWALSKI,R. & SERGOT,M. A Logic based calculus of events. **New Generation Computing 4**, 1986. p.67-95.

[Lipeck 87] LIPECK,U.W. & SAAKE,G. Monitoring dynamic integrity constraints based on temporal logic. **Information Systems**, GB, 12(3):255-69, 1987.

[Loucopoulos 91] LOUCOPOULOS,P. et al. TEMPORA - Integrating database technology, rule-based systems and temporal reasoning for Information Systems development. (To be included in the IEEE Office Knowledge Engineering Newsletters, Feb. 1991).

[Maiocchi 91a] MAIOCCHI,R.; PERNICI,B. Temporal Data Management Systems: a Comparative View. **IEEE Transactions on Knowledge and Data Engineering**, 3(3), Dec 91.

[Maiocchi 91b] MAIOCCHI,R.; PERNICI,B.; BARBIC,F. **Automatic deduction of temporal information**. University of Udine, Dipartimento de Matematica e Informatica, 1991. 58p. (Research Report). (To be published in ACM Transactions on Database Systems).

[Manna 81] MANNA,Z. & PNUELI,A. Verification of concurrent programs: the temporal framework. In: BOYE MOORE (de.) **The Correctness Problem of Computer Science**. Academic Press, 1981. p.215-73.

[Mylopoulos 90] MYLOPOULOS,J.et al. Telos: representing knowledge about Information Systems. **ACM Transactions on Information Systems**, New York, 8(4):325-62, Oct. 1990.

[Pernici 90] PERNICI,B. Objects with Roles. In: CONFERENCE ON OFFICE INFORMATION SYSTEMS, Cambridge, Massachusetts, April 25-27, 1990. **Proceedings**. SIGOIS Bulletin, 11(2,3):205-15, 1990.

[Schiel 83] SCHIEL,U. An Abstract introduction to the Temporal-Hierarchic Data Model (THM). INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 9., Florence (Italy), Oct. 31 - Nov. 2, 1983. **Proceedings**. Italy, VLDB, 1983. p.322-30.

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

[Snodgrass 85] SNODGRASS,R. & AHN,I. A Taxonomy of time in databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, Texas, May 28-31, 1985. **Proceedings**. New York, ACM, 1985. p.236-46.

[Wiederhold 91] WIEDERHOLD,G.; JAJODIA,S.; LITWIN,W. Dealing with granularity of time in temporal databases. In: INTERNATIONAL CONFERENCE CAISE'91, 3., Trondheim, Norway, May 13-15, 1991. **Proceedings**. Berlin, Springer-Verlag, 1991. p.124-40.

ANEXO 1

EXEMPLO DE APLICAÇÃO - AGÊNCIA DE LOCAÇÃO DE FITAS DE VÍDEO

Ilustramos a utilização do modelo F-ORM estendido através da especificação de três classes de uma agência de locação de fitas de vídeo. Como o objetivo é tão somente apresentar o modelo, não descrevemos o exemplo completo, somente algumas partes significativas.

```
process class (
 Rental,
 < base-role,
        static properties = [ (creation, INSTANT) ],
        dynamic properties = [ (client, CLIENT), (tape, TAPE) ],
        rules = [rule1 : msg(->create_object) => msg(<-allow_role(rental_service))
                 rule2 : msg(->create_object) =>
                          msg(<-allow_role(tape_devolution))
                 rule3 : msg(->create_object) => msg(<-allow_role(rental_control))
                 rule4 : msg(->create_object) => msg(<-allow_role(clients_rental))]
 >

/* O funcionário recebe o pedido de locação do cliente e pede informações sobre o cliente; se o cliente
estiver habilitado a alugar fitas, o funcionário informa ao controle de locações as condições desta locação
e dá a fita ao cliente; se o cliente não está habilitado, o funcionário lhe fornece esta informação. */

 < Rental_service,
        static properties = [ ]
        states = [ wait_request, wait_check ]
```

```
        messages = {
                tape_request(client:CLIENT, tape:TAPE) from clients_rental,
                check_client (client:CLIENT) to rental_control,
                allowed from rental_control
                rejected from rental_control,
                begin_rental to rental_control,
                request_denied to clients_rental }
        rules = {rule1 : msg(<-add_role) => state(wait_request),
                rule2 : state(wait_request), msg(<-tape_request) =>
                        msg(->check_client), state( wait_check);
                rule3 : state(wait_check), msg(<-rejected) =>
                        msg(->request_denied), state(wait_request),
                rule4 : state(wait_check), msg(<-allowed) =>
                        msg(->begin_rental), state(wait_request) }
 >.
```

/* O funcionário recebe a devolução da fita e envia as informações correspondentes ao controle de locações. */

```
 < Tape_devolution.
        ...
 >.
```

/* Recebe pedido de verificação de cliente, verifica se o cliente está habilitado a alugar uma fita e envia a resposta ao serviço de locação; recebe informações de início e final de locações e armazena os dados correspondentes. */

```
 < Rental_control.
        ...
 >.
```

/* O cliente faz um pedido de locação ao serviço de locações, recebe a fita ou uma recusa; no caso de receber a fita, mais tarde devolve a fita ao serviço de locações. */

```
< Clients_rental.
        ...
 > )

process class (
 ACCOUNTING,
 < base-role.
        ...
 >
 < employee_control.
        ...
 >.
 < rental_account.
        ...
 > )

resource class (
 PERSON,
 < base-role.
        static properties = { (creation,INSTANT), (name, STRING) },
        dynamic properties = { (address, STRING) },
        rules = { }.
 >.
```

```
< Client,
        static properties = { (client_code, INTEGER) },
        dynamic properties = { (rented_tape, TAPE) },
        messages = { rent from Rental_control, disable from Rental_control },
        states = { active, renting, disabled },
        rules = {rule1 : msg(<-add_role) => state(active),
                 rule2 : state(active), msg(->rent) => state(renting),
                 rule3 : state(renting), msg(<-rent) => state(renting),
                 rule4 : msg(<-disable) => state(disabled) }
>,
< Employee,
        dynamic properties = { (salary, REAL), (hire_date,DATE), out_date,DATE) },
        messages = {
                 modify_salary(Value:REAL, STARTING_TIME:DATE) from employee_control,
                 end_employment (FINAL_TIME:DATE) from employee_control },
        states = { employed, disconnected },
        rules = {rule1 : msg(<-add_role) => state(employed),
                 rule2 : state(employed), msg(<-modify_salary(V)) => state(employed) :
                          immediately past exists V1 (salary(V1) and V>V1)
                 rule3 : state(employed), msg(<-end_employment) => state(disconnected) }
>)

resource class (
 TAPE,
 < base_role,
        static properties = { (creation:INSTANT), (tape_number, INTEGER) },
        dynamic properties = { (tape_film, STRING), (film_type, STRING) },
        rules = { },
 >,
 < Life_time,
        ...
 >,
 < Rentals,
        dynamic properties = {
                 (client_code, INTEGER),
                 (starting_date, DATE),
                 (end_date, DATE) },
        messages = {
                 rental from Rental_control,
                 tape_devolution from Rental_control,
                 rented_time to Rental_account},
        states = { available, rented },
        rules = {rule1 : msg(<-add_role) => state(available),
                 rule2 : state(available), msg(<-rental) => state(rented),
                 rule3 : state(rented), msg(tape_devolution) => msg(->rented_time), state(available) }
 >,

 < Tape_loss,
        ...
 > )
```