UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO ORIENTADO A OBJETOS PARA ESPECIFICAÇÃO DE INFORMAÇÕES TEMPORAIS EM SISTEMAS DE INFORMAÇÃO DE ESCRITÓRIOS APROPRIADO AO PARADIGMA DE REUTILIZAÇÃO

por
NINA EDELWEISS

RP- 201 Novembro/92
Relatório de Pesquisa

UFRGS - II - CPGCC
Caixa Postal 15064 - CEP 91501-970
Porto Alegre - RS - BRASIL
Telefone: (051) 336-8399 e 339-1355
Fax: (051) 336-5576
Email: PGCC@INF.UFRGS.BR

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO ORIENTADO A OBJETOS PARA ESPECIFICAÇÃO DE INFORMAÇÕES TEMPORAIS EM SISTEMAS DE INFORMAÇÃO DE ESCRITÓRIOS APROPRIADO AO PARADIGMA DE REUTILIZAÇÃO

por
NINA EDELWEISS

RP- 201 Novembro/92
Relatório de Pesquisa

UFRGS - II - CPGCC
Caixa Postal 15064 - CEP 91501-970
Porto Alegre - RS - BRASIL
Telefone: (051) 336-8399 e 339-1355
Fax: (051) 336-5576
Email: PGCC@INF.UFRGS.BR

Sistemas de informação - SBD
Sistemas: Informação: Escritórios
Especificação: Requisitos
Orientação: Objetos
Reusabilidade: Software

CNPq 1.03.03.00-6

RESUMO

O presente trabalho consiste da proposta de uma tese de doutorado, a ser desenvolvida no CPGCC da UFRGS, sob orientação do prof. Dr. José Palazzo Moreira de Oliveira. O objetivo desta tese é o de definir um modelo de dados a ser utilizado para especificação dos requisitos de sistemas de informação de escritórios que, baseado no paradigma de orientação a objetos, evidencie a possibilidade de reutilização de partes de especificações anteriormente efetuadas. Especial ênfase será dada à representação de informações temporais.

PALAVRAS-CHAVE: ESPECIFICAÇÃO DE REQUISITOS, SISTEMAS DE INFORMAÇÃO DE ESCRITÓRIOS, REUTILIZAÇÃO DE ESPECIFICAÇÕES, REPRESENTAÇÃO DE ASPECTOS TEMPORAIS.

ABSTRACT

This work presents a PHD thesis proposal. The thesis will be developed in the CPGCC of the UFRGS, with orientation of Prof. Dr. José Plazzo Moreira de Oliveira. The main purpose is the definition of a data model to be used in office information systems requirement specification. The data model will be based on the object-orientation paradigm and will enphazise the possibility of reusing former specification. Special attention will be given to temporal information specification.

KEYWORDS: REQUIREMENT SPECIFICATION, OFFICE INFORMATION SYSTEMS, SPECIFICATION REUSE, TEMPORAL ASPECTS REPRESENTATION.

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho consiste da proposta de uma tese de doutorado, a ser desenvolvida no CPGCC da UFRGS. O objetivo desta tese é o de apresentar um modelo de dados a ser utilizado para especificação de sistemas de informação de escritórios, utilizando o paradigma de orientação a objetos e evidenciando a possibilidade de reutilização de partes de especificações anteriormente efetuadas em um determinado domínio de aplicação. Especial ênfase será dada à representação de informações temporais. O modelo deverá apresentar facilidade de utilização, através da definição de uma linguagem rigorosa, simples, não ambígua, que apresente termos familiares, comuns ao domínio considerado. Através da utilização desta linguagem será possível estabelecer a comunicação entre os usuários e os projetistas do sistema de informação de escritório, possibilitando a validação da especificação por parte dos usuários.

A proposta desta tese foi dividida em seis partes. No capítulo 2 é feita a definição do problema considerado por esta tese, seguido da descrição das soluções adotadas; após algumas considerações a respeito da necessidade e da importância de se desenvolver especificações de sistemas de informação, são apresentadas as características do domínio considerado - sistemas de informação de escritórios; em seguida são descritas a técnica utilizadas na solução do problema, a definição de um modelo de dados, e as características que este modelo deverá suportar. O capítulo 3 traz as idéias gerais a serem empregadas na formulação de um método de especificação baseado no modelo proposto. Alguns resultados preliminares obtidos em trabalhos anteriormente realizados são apresentados no capítulo 4. No capítulo 5 é feito o detalhamento do trabalho a ser realizado, explicando cada uma das partes a serem desenvolvidas. Um esquema preliminar da tese propriamente dita é apresentado no capítulo 6, seguido de uma previsão de utilização do tempo para seu desenvolvimento.

2 MODELO PARA ESPECIFICAÇÃO DE SISTEMAS DE INFORMAÇÃO

Diversos ciclos de vida de desenvolvimento de sistemas podem ser encontrados na literatura[BOE 88, GIR 90, HEN 90b]. O ciclo de vida do desenvolvimento de um sistema de informação geralmente apresenta as seguintes fases principais: (1) coleta dos requisitos junto aos usuários do sistema; (2) organização e análise deste requisitos, da qual resulta uma especificação inicial do sistema, geralmente semiformal; (3) definição do sistema, na qual é feita a opção das atividades que deverão ser automatizadas no sistema; (4) especificação formal dos requisitos, efetuada por um projetista com base nos requisitos definidos anteriormente; (5) implementação do sistema a partir de sua especificação; (6) testes e manutenção do sistema. Estas fases não são estritamente seqüenciais - em cada uma delas devem ser levados a efeito procedimentos de validação e/ou de verificação, procurando detectar o mais cedo possível eventuais erros de interpretação de requisitos ou de projeto. Os resultados dos processos de validação e de verificação podem provocar a retomada de uma fase anterior do desenvolvimento. Na maioria das vezes é construído um protótipo do sistema antes de sua implementação, com base na especificação dos requisitos, de modo a simular a execução do sistema e validá-lo junto aos usuários.

A formulação inicial dos requisitos, resultantes da coleta de requisitos, é feita geralmente em forma textual, através de linguagem natural. Esta formulação não é muito precisa, apresentando ambigüidades. A partir desta formulação inicial é feita a **especificação dos requisitos**, expressos então através de uma linguagem formal. A utilização de uma linguagem formal possibilita a definição precisa destes requisitos, além de permitir, em alguns casos, mecanismos de verificação através dos quais pode ser garantida a qualidade da especificação proposta. É importante que a formulação inicial dos requisitos seja compatível com o formalismo a ser utilizado na especificação formal.

Esta especificação serve de base não só para a prototipação do sistema, mas também para a sua implementação. Ela deve, entretanto, ser independente da implementação, para não sofrer a influência dos avanços tecnológicos que ocorrem. A especificação formal dos requisitos geralmente é obtida através de um processo incremental: a partir da primeira definição dos requisitos, o conhecimento é progressivamente refinado e enriquecido, até que a especificação esteja completa.

A especificação de uma aplicação deve representar todas as suas características, tanto as estáticas como as dinâmicas. Deve apresentar a estrutura das informações manipuladas, seus relacionamentos, seu comportamento. Um dos principais aspectos a representar, necessário à representação da dinâmica da aplicação, são os **aspectos temporais.**

A possibilidade da **reutilização de especificações** anteriormente construídas, ou parte delas, torna a tarefa de especificação mais eficiente, mais econômica e mais segura - a especificação será construída mais rapidamente, baseada em especificações anteriormente validadas e implementadas. Um dos requisitos para que seja possível a reutilização a nível de especificações é que o domínio de aplicação seja definido; deve ser genérico o bastante para que diversas aplicações possam nele ser realizadas, mas não por demais amplo, caso contrário o tempo para encontrar uma parte de especificação a reutilizar seria grande demais, tornando o processo ineficiente.

Um domínio de aplicações no qual a reutilização de software pode ser eficiente, dada a sua dimensão e abrangência, é o de **sistemas de informação de escritórios.** Sistemas de informação de escritórios são tipos particulares de sistemas de informação. Um escritório é uma unidade funcional de processamento de informações, onde podem ser encontrados grande número de elementos (documentos, pessoas) interagindo de modos geralmente bem definidos. A comunicação entre as pessoas pode ser feita verbalmente ou através de troca de mensagens ou documentos. O conhecimento é compartilhado por todas os elementos do escritório. As atividades executadas são concorrentes e assíncronas. A representação de aspectos temporais é fundamental na especificação de um sistema de informação de escritórios, dada a importância que o tempo tem nestes ambientes (por exemplo, representação de prazos, de datas, de restrições temporais).

Existem diversos formalismos para serem utilizados na especificação formal dos requisitos. O **objetivo deste trabalho** é o de apresentar um formalismo que se aplique à especificação de sistemas de informação de escritórios, adequado à reutilização de especificações. Especial atenção deverá ser dada à representação de aspectos temporais, muito importantes no domínio de aplicações selecionado. O modelo proposto não tem o objetivo de ser inteiramente novo, mas de acrescentar alguns aspectos novos aos formalismos já existentes.

Duas são as **contribuições esperadas para este trabalho**: (1) estudo de sistemas de informação de escritórios com vistas à reutilização de especificações, levando a uma classificação de requisitos destes ambientes, especialmente no que diz respeito a aspectos temporais; e (2) formalismo para especificação de requisitos em sistemas de informação de escritórios, próprio para o paradigma de reutilização e incluindo aspectos temporais.

O formalismo proposto se baseia na definição de um modelo de dados. Este modelo de dados deverá permitir a especificação dos sistemas propostos, sendo ao mesmo tempo preciso e simples de ser utilizado. Será proposta ainda uma metodologia para a utilização deste modelo, integrando as diferentes partes do processo de especificação de sistemas e proporcionando a verificação e a validação das especificações construídas.

Apresentaremos, a seguir, alguns conceitos relativos à reutilização e à representação de aspectos temporais, indicando como são tratadas em alguns dos métodos existentes e quais as idéias iniciais para seu tratamento no modelo proposto.

2.1 Reutilização

Uma das técnicas mais estudadas atualmente para aumentar a produtividade e a qualidade do software é a de reutilização. Devido à sua evidente importância, no sentido de se obter especificações mais corretas em menor espaço de tempo, deverá ser considerada como um aspecto fundamental na definição do modelo a ser adotado.

A possibilidade de reutilização deve ser levada em consideração desde o início do desenvolvimento de um produto de software, para que seja realmente efetiva. Seu emprego à nível de especificações simplifica a reutilização nos níveis de projeto e implementação. Além de aumentar a produtividade do processo de especificação, a reutilização resulta ainda em um aumento da qualidade e da confiabilidade da especificação, uma vez que são utilizadas partes de outras especificações anteriormente testadas e depuradas, o que diminui a quantidade necessária de verificação.

Tipos distintos de reutilização podem ser considerados: (1) de componentes de software (especificações ou código); (2) de informações a respeito do processo de

desenvolvimento de um produto; e (3) de conhecimentos. Para cada um deles estão sendo desenvolvidas técnicas e ferramentas de apoio.

Neste trabalho serão contemplados os dois primeiros tipos de reutilização da taxonomia acima apresentada - reutilização de especificações, de partes de especificações e de informações do desenvolvimento. A possibilidade de reutilizar as especificações feitas deve ser levada em consideração na elaboração do modelo a ser utilizado. Este deverá apresentar facilidades para armazenamento de informações em uma biblioteca de componentes de software, para sua posterior recuperação e eventual modificação.

Uma das técnicas mais utilizadas para propiciar uma eventual reutilização de especificações é a da definição de um domínio de aplicações, no qual diversas aplicações específicas diferentes poderiam ser desenvolvidas. Uma vez definidos os elementos deste domínio genérico, as especificações das aplicações particulares seriam realizadas reutilizando e adaptando os elementos definidos. Exemplos de sistemas que utilizam este conceito são encontrados em [GAN 91, NEI 84, SUT 91]. A definição do domínio de aplicação é um ponto bastante crítico - se o domínio é por demais específico, dificilmente os elementos nele definidos poderão ser reutilizados em aplicações diferentes; se, no entanto, este domínio é muito genérico, a necessidade de adaptação dos elementos será tão grande que não se justifica o esforço de reutilização. Decidimos escolher Sistemas de Informação de Escritórios como domínio genérico para o modelo que está sendo proposto. Este domínio é genérico o bastante para permitir a modelagem de um grande número de aplicações, não sendo amplo demais.

Para permitir a reutilização de especificações, uma solução inicial é a adoção de um **modelo orientado a objetos**; modelos que se baseiam neste paradigma se prestam de maneira muito eficiente à reutilização através da definição de classes de objetos reutilizáveis. Uma análise criteriosa do domínio de aplicação permite a definição de classes de objetos comuns a várias aplicações. A característica básica dos objetos, ou seja, o encapsulamento de todas as informações relativas ao objeto, facilita a sua reutilização em aplicações diferentes. Os objetos do domínio são organizados em bibliotecas, de forma tal a permitir a sua recuperação eficiente. A seguir serão apresentados alguns conceitos de orientação a objetos, ressaltando de que maneira estes modelos podem ser utilizados para tornar a reutilização eficiente. Também são vistas algumas características que as bilbiotecas de objetos devem apresentar.

2.1.1 Orientação a Objetos

Neste trabalho opta-se pela adoção de um modelo de dados que suporte o paradigma de orientação a objetos [KOR 90, LAD 88]. Este paradigma tem recebido atenção especial nos meios de pesquisa, por se apropriar especialmente bem a especificações de problemas complexos. Apresenta características apropriadas à reutilização, tais como instanciação ou especialização, por herança, de classes de objetos pré-definidas. Dada uma biblioteca de classes de objetos de uso geral, definidas de forma mais ou menos abstrata, estas podem ser modificadas e/ou interligadas para gerar outras classes de objetos que serão reutilizados em aplicações específicas.

O paradigma de orientação a objetos baseia-se no princípio de que os problemas envolvem objetos do mundo real, interagindo entre si com comportamentos determinados e evoluindo no tempo. Uma especificação feita segundo este paradigma tem como aspectos principais a identificação e a representação destes objetos.

Um objeto é caracterizado por um conjunto de atributos (propriedades) que definem a sua estrutura, e por um conjunto de procedimentos (métodos, operações) que definem o seu comportamento. Objetos similares são agrupados em classes, podendo ser formadas hierarquias de classes através de abstrações de generalização, especialização, associação e agregação, com herança de atributos e de operações.

A solução de um problema segundo este paradigma é obtida através da definição de um conjunto de classes de objetos e dos possíveis inter-relacionamentos entre os objetos destas classes. A especificação de uma aplicação através de um modelo orientado a objetos é feita através da definição de um conjunto de objetos juntamente com um conjunto de restrições que definem o comportamento dos objetos na aplicação.

Diversos modelos de dados encontrados na literatura apresentam esta característica, com por exemplo, os dos Bancos de Dados ENCORE [HOR 87], ORION [KIM 87], e O^2 [LEC 88, BAN 88], da linguagem GALILEO [ALB 86], do sistema OFFIS [KON 82] e do projeto LOGRES [CAC 90]. A existência deste grande número de modelos, com suas vantagens e desvantagens específicas, não justifica a definição de mais um novo modelo. Optamos, portanto, por adotar um modelo dos já existentes, que se

aplique especificamente ao domínio de aplicação considerado, estendendo-o de modo a representar todos os aspectos considerados importantes.

O modelo escolhido foi F-ORM (Functionality in Object with Roles Method) [DEA 91], definido dentro do projeto ITHACA, um projeto ESPRIT da comunidade européia. Esta escolha se deve ao fato de se tratar de um modelo novo, ainda em desenvolvimento, no qual são considerados aspectos específicos do domínio de aplicações escolhido - sistemas de informação de escritórios. A reutilização de especificações é uma das preocupações no desenvolvimento deste modelo. Os aspectos temporais, entretanto, ainda não são contemplados, permitindo uma real contribuição na definição do modelo. Além disso, existe atualmente uma colaboração informal entre o Instituto de Informática e o grupo que desenvolve o modelo na Itália, o que permite que os resultados obtidos neste trabalho sejam efetivamente utilizados.

O modelo F-ORM tem por objetivo integrar a descrição de elementos isolados de um escritório com a sua funcionalidade, isto é, descrever como as atividades do escritório são executadas em termos de organização do trabalho e de cooperação entre agentes.

No modelo F-ORM, para cada classe de objetos é definido um conjunto de papéis (*roles*) que este objeto pode assumir, papéis estes que descrevem o comportamento do objeto em diferentes situações. Para cada papel é definido um conjunto de atributos, as mensagens que podem ser enviadas ou recebidas, os estados que o objeto pode assumir e as regras que definem as transições válidas entre estes estados. Um objeto pode assumir diferentes papéis em tempos diferentes, e também pode desempenhar mais de um papel ao mesmo tempo. Os comportamentos de objetos em diferentes papéis podem ser relacionados entre si através de regras e restrições que governam o comportamento concorrente.

Podemos identificar dois tipos basicamente diferentes de objetos: (1) objetos que representem somente entidades físicas ou abstratas, correspondendo aos agentes e aos recursos manipulados em uma aplicação; para estes devem ser definidas a estrutura (atributos, papéis, mensagens) e as ações que podem sofrer; e (2) objetos que representem atividades funcionais, para as quais devem ser definidas, além da estrutura (atributos e papéis), pré e pós-condições de execução e um conjunto de ações que caracterizam a sua execução. O modelo F-ORM identifica estes dois tipos de objetos, mas os trata de uma maneira uniforme. A separação dos objetos nestas duas classes permite a modelagem dos

agentes e dos recursos, os quais constituem os elementos menos mutáveis da aplicação, da forma a mais estática possível, através de sua estrutura e características fixas. Quanto às funções, estas dependem de como são executadas, das pessoas que as executam, do relacionamento entre ela, apresentando geralmente mais de uma forma possível de modelagem.

O paradigma de orientação a objetos permite a definição de classes de objetos através de abstrações feitas com base em outras classes, definidas anteriormente (classificação, generalização, agregação), com herança de propriedades (atributos). Em alguns modelos é permitida a redefinição de atributos em subclasses, em outros não. Vamos adotar a primeira opção de modo a permitir a modelagem de exceções e de classes parcialmente contraditórias.

É importante que o desenvolvimento de um sistema seja efetuado desde o início através do mesmo paradigma. Existem atualmente diversas propostas de metodologias de desenvolvimento de sistemas (projeto e implementação) segundo o paradigma de orientação a objetos [BOO 86, COA 91, HEN 90a, WIR 90]. A proposta do novo modelo de dados será incorporada a uma metodologia para projeto de Sistemas de Informação (capítulo 3). A utilização do paradigma de orientação a objetos na fase de especificação do sistema é fundamental para facilitar a sua posterior implementação através da utilização de uma linguagem orientada a objetos.

Características da orientação a objetos que propiciam a reusabilidade são, segundo [PRI 90], a abstração, a passagem de mensagens e a herança. A abstração induz a que sejam criadas classes genéricas de objetos, associando funções a estruturas de dados, sem considerar detalhes de implementação. A comunicação entre os objetos através da passagem de mensagens leva à definição de um protocolo de comunicação, através do qual é facilitada a localização de componentes. A herança de propriedades encoraja fortemente a reutilização, tornando possível a instanciação de uma determinada classe, acrescentando-lhe novas características e comportamentos.

2.1.2 BIBLIOTECA DE CLASSES

A reutilização de classes prossupõe a existência de uma biblioteca de classes, organizadas de maneira a permitir sua recuperação rápida e efetiva [GIB 91, GIR 90].

Exemplos de bibliotecas de classes são a do sistema DRACO [NEI 84], a biblioteca MUSEION [PON 91] e a "Basic Eiffel Libraries" [MEI 90]. Três aspectos importantes devem ser considerados quando da utilização de uma biblioteca de classes: como armazenar classes em uma biblioteca, como recuperar as classes armazenadas e como reutilizar as classes recuperadas.

A construção da biblioteca de classes é realizada por um especialista, sendo específica para um determinado domínio de aplicação. Construir uma biblioteca para um domínio genérico seria por demais trabalhoso, dando origem a um número muito grande de classes, de difícil reutilização. Já quando se fixa um determinado domínio, as classes específicas deste domínio podem ser mais facilmente identificadas e, se necessário, modificadas. A identificação e seleção das classes que devem ser inseridas na biblioteca é feita através de uma análise de domínio. Muitas vezes são utilizadas ferramentas de análise do domínio, como por exemplo a ferramenta RECAST [BEL 91, FUG 90] implementada para o modelo F-ORM.

A análise de domínio pode ser efetuada de duas maneiras diversas: (1) a priori (precursora), sendo definidos os elementos a serem utilizados e reutilizados no futuro, e (2) a posteriori, quando são selecionados elementos reutilizáveis a partir de um conjunto de componentes produzidos por aplicações desenvolvidas neste domínio. O povoamento de uma biblioteca de classes sempre deverá iniciar por uma análise a priori, sendo aconselhável que sua manutenção seja realizada de forma constante, por pessoas especializadas, através de análises a posteriori.

A biblioteca de classes deve ser organizada de modo a facilitar a posterior recuperação de classes, e apresentar flexibilidade suficiente para aceitar inclusões de novas classes.

No método proposto deverá ser utilizada uma base de informações para armazenar tanto as especificações a serem realizadas para aplicações específicas, como as classes a serem reutilizadas. O armazenamento das classes deverá ser realizado através de uma ferramenta de análise do domínio.

A recuperação de classes em uma biblioteca é um dos aspectos principais para a efetiva validade da reutilização [STA 84]. A reutilização não terá validade se o esforço dispendido para recuperar uma classe for muito superior ao que seria gasto para a

construção de uma nova classe. As classes armazenadas devem ser facilmente encontradas, de acordo com aspectos que se quer utilizar. O ideal é que a recuperação seja auxiliada por ferramentas [PIN 90]. O armazenamento das classes deve ser efetuado levando em consideração a sua posterior recuperação.

A definição de objetos em diversos níveis de abstração através de hierarquias de classificação, de generalização e de agregação permite uma organização facilmente recuperável. Um dos modos mais comuns de armazenar classes é através de uma rede semântica, cujos elos representem as abstrações utilizadas na definição da classe. A recuperação é efetuada percorrendo esta rede e analisando a estrutura e o conteúdo de cada uma das classes encontradas.

A localização de classes em uma biblioteca pode ser efetuada através de uma busca exploratória ou sistemática [PRI 90]. Na busca exploratória são percorridas as hierarquias de generalização/especialização, sendo analisadas as descrições das classes, suas características, etc. Quando a biblioteca é muito grande esta busca pode se tornar muito demorada e complexa. Na busca sistemática são procuradas informações específicas nas classes armazenadas.

Uma maneira que facilita a recuperação de classes é através de um catálogo associado à biblioteca. Neste as classes podem ser classificadas (associadas a determinadas categorias) e indexadas (através de, por exemplo, palavras-chave), além de apresentarem uma descrição textual que facilite a sua identificação. Além disso, podem ser definidos interfaces padronizados para os objetos, sendo que conjuntos de classes podem apresentar o mesmo interface, diminuindo assim a busca.

Existem diferentes **formas de reutilizar** uma classe: (1) a classe pode ser reutilizada tal como se apresenta na biblioteca de classes; (2) pode ser especializada, através do acréscimo de novas características; ou (3) pode ser modificada, de modo a ter alterados alguns de seus atributos e/ou operações.

A reutilização de uma classe sem qualquer adaptação é muito difícil de acontecer. É importante, portanto, que a biblioteca apresente, além de informações a respeito da estrutura das classes armazenadas, informações adicionais que auxiliem a sua posterior reutilização através de indicações de como adaptá-las a aplicações específicas. Isto se aplica, principalmente, nos casos de especialização de classes.

No método aqui proposto o conhecimento a respeito das possíveis adaptações das classes para aplicações específicas deverá ser armazenado juntamente a cada uma das classes, em modo semelhante ao utilizado pelo projeto ITHACA para o modelo F-ORM. Neste, informações quanto à adaptação da classe são definidas através de metaclasses associadas a cada classe definida [PER 90]. Uma metaclasse apresenta informações relativas a possível reutilização da classe, sendo descritas dependências entre elementos da classe e outros elementos definidos, além de sugestões para a reutilização desta classe. Estas sugestões são definidas quando da criação da classe, sendo atualizadas durante toda a sua existência de modo a incorporar informações colhidas de experiências de sua reutilização, através de análises de projetos realizados e das ações tomadas para sua adaptação. A especificação de informações objetivas de projeto junto a cada uma das classes permitirá uma reutilização mais efetiva.

No presente trabalho, a forma como as classes devem ser estruturadas na biblioteca e a maneira como deve ser realizada a sua recuperação não foram ainda definidas, pois dependerão do modelo a ser proposto.

2.2 Tempo

A possibilidade de modelar informações temporais é um aspecto de fundamental importância em aplicações de sistemas de informação de escritórios. O tempo está presente não só nos recursos manipulados (informações relativas a datas, momento de criação do recurso, tempo de validade das informações, etc.) como também nas atividades exercidas (tempos de início e de final da atividade, duração da atividade, tempo de validade, etc.). Relacionamentos temporais entre atividades, como por exemplo, a ordem de execução entre duas atividades, também devem ser representados. A possibilidade de definir que um evento possa acontecer no futuro, ou que deva acontecer, são diferentes informações a modelar.

É necessário permitir não só a definição de conhecimentos temporais - registros de eventos - como também permitir que sejam realizados raciocínios temporais com base em informações que têm seus valores alterados com a passagem do tempo. Para isto é necessário que possam ser armazenados os valores de estados passados, permitindo uma análise da evolução dos dados.

Três diferentes áreas de pesquisa têm desenvolvido estudos a respeito de representação de informações temporais [MAI 91]: (1) modelagem conceitual de dados, que se preocupa com a representação do tempo propriamente dita, no que diz respeito a tempo de ocorrência de eventos e relacionamentos temporais entre eventos; (2) sistemas de bancos de dados, que se preocupam principalmente com o gerenciamento de informações temporais; e (3) Inteligência Artificial, cujo interesse recai no raciocínio temporal. Embora situado na primeira área, nosso trabalho é fortemente influenciado pelos resultados obtidos nas outras duas.

Muitos trabalhos têm sido realizados com o intuito de estender modelos convencionais para incorporar suporte a dados temporais, como os modelos ERT (Entity Relationship Time) [LOU 91] e TEER (Temporal Extended ER) [SUB 90]. Em [GAB 91] é encontrada uma extensão de um banco de dados relacional através da utilização de lógica temporal. Estes trabalhos, entretanto, não dão muito atenção ao gerenciamento das informações temporais, preocupando-se principalmente com sua modelagem. Também são encontrados diversos trabalhos que se preocupam em estender linguagens de "query" para tratar de informações temporais, como o TSQL [NAV 88] e o HSQL [SAR 90]. O ideal, no entanto, é o desenvolvimento de novos modelos e linguagens, dirigidos à representação de informações temporais.

A modelagem de aspectos temporais nos sistemas atualmente em desenvolvimento é efetuada principalmente através de representações explícitas de tempo associadas às informações ("timestamps" e intervalos temporais) ou de lógica temporal.

Na **representação explícita do tempo** são assumidos eventos temporais representados como pontos de tempo definidos ao longo de um eixo temporal (seqüencia linear de pontos no tempo). Um ponto especial é o tempo atual que representa o momento presente (agora) e se move no tempo. Podem ser utilizados intervalos temporais, definidos através de pares que representam o momento inicial e o momento final. A granularidade de um determinado elemento temporal é a unidade do nível de abstração no qual o elemento é definido (por exemplo, horas, dias, anos). Permitir que uma definição seja feita utilizando diferentes granularidades permite uma representação mais coerente da realidade. O modelo que permite este tipo de representação deve, no entanto, fornecer apoio à manipulação destas informações, fazendo as necessárias conversões entre as unidades utilizadas.

Quando se utiliza um modelo orientado a objetos, a representação temporal pode ser associada diretamente a um objeto ou a cada um de seus atributos. A associação de tempo a cada um dos atributos permite a modelagem de informações com diferentes granularidades.

Exemplo de sistema que utiliza pontos no tempo é a linguagem GRAPES [HEL 91], uma linguagem orientada a objetos para modelagem de Sistemas de Informação.

Da Inteligência Artificial resultou a **teoria de Allen** [ALL 83, ALL 84], onde os eventos são representados através de intervalos. Intervalos de tempo podem ser relacionados uns aos outros através de um conjunto de relacionamentos temporais, sendo que cada um destes relacionamentos é representado por um predicado na correspondente lógica temporal. Foi desenvolvido um conjunto de axiomas para deduzir relacionamentos entre quaisquer dois intervalos, com base no comportamento transitivo dos relacionamentos. A linguagem TELOS [MYL 90] se baseia nesta teoria.

A utilização de **lógica temporal** foi estudada tanto na área de bancos de dados como em inteligência artificial, engenharia de software e sistemas concorrentes. O desenvolvimento de capacidades dedutivas temporais em bancos de dados tem forte influência do **Cálculo de Eventos** [KOW 86, SRI 88], um enfoque para representação e raciocínio a respeito de eventos e de tempo, em um ambiente de programação em lógica. A noção de evento é considerada mais primitiva do que a de tempo, sendo ambas representadas explicitamente através de cláusulas de Horn aumentadas com negação por falha. Uma das maiores aplicações do Cálculo de Eventos é na atualização de bancos de dados, uma vez que permite a atualização de informações do passado. A utilização de negação por falha permite tratar de informações incompletas. Este tipo de representação permite a representação de eventos com tempos desconhecidos e eventos parcialmente ordenados e concorrentes, uma vez que os eventos são diferenciados do tempo.

Na lógica temporal o tempo é representado implicitamente, através de regras que determinam o comportamento temporal. São definidos operadores temporais a serem utilizados nestas regras. Uma das vantagens desta forma de representação é a possibilidade de utilizar a lógica temporal como linguagem de "query" de bancos de dados temporais. Exemplos de modelos e sistemas que utilizam lógica temporal podem ser encontrados em [ARA 90, ARA 91, COR 91, FIN 91, SEG 88].

Muitos dos sistemas implementados utilizam uma combinação destas duas formas: representação explícita para atributos, para representar a evolução dos objetos, e regras de conhecimento para representar requisitos temporais especiais. É o caso do modelo OSAM*/T [SU 91].

O modelo F-ORM, no qual basearemos o modelo a ser proposto, não apresenta possibilidade de representação de aspectos temporais. A idéia inicial que pensamos adotar para a representação temporal é tomar como base a proposta de [CLI 88], que consiste em definir um "objeto histórico" - um objeto cujos atributos são representados por funções. Estas funções, que por sua vez são também objetos, mapeiam do tipo "tempo" para o tipo do atributo. O período de tempo durante o qual um objeto modela uma entidade do mundo real é denominado de tempo de vida ("lifespan") do objeto. O domínio de uma função representada por um atributo de um objeto é restrito aos tempos do vida deste objeto. Este tempo de validade pode ser definido através de um atributo especial.

Ex: Objeto empregado

Atributos	salário	: TEMPO -> REAL
	departamento	: TEMPO -> STRING
	nome	: TEMPO -> STRING
	lifespan	: TEMPO -> TEMPO

Além de permitir a definição de diferentes granularidades para os atributos, o modelo deverá ainda possibilitar a definição de intervalos temporais, do tipo de variação temporal em um intervalo (discreto, contínuo, por degraus, etc.) e de regras para interpolação de valores. Deverão, ainda, ser utilizadas regras em lógica para expressar condições temporais.

2.3 Concorrência

A possibilidade de execução simultânea de mais de uma tarefa também deverá ser considerada no modelo de especificação proposto. Existem diversos métodos de especificação de atividades concorrentes. Entre eles podemos citar: (1) métodos baseados em redes, como por exemplo as Redes de Petri [DEA 85, LEE 88, LIU 89]; (2) a representação através de regras de lógica de primeira ordem; (3) especificação através de linguagens de programação, enfoque utilizado pela linguagem ESTELLE [BOC 90],

representando o sincronismo através de eventos que habilitam uma atividade; e (4) formalismos matemáticos, como o método LOTOS [BOC 90].

Diversas pesquisas têm sido realizadas unindo orientação a objetos com concorrência, entre as quais podemos citar as linguagens Hybrid [KON 88, PAP 90], POLL-I [AME 90], Emerald [BLA 87], Abacus [NIE 90], e a metodologia COOP [AGH 90].

No método F-ORM é feita a definição de regras e de restrições para controlar o comportamento concorrente. Baseados na idéia de utilizar uma base de informações para armazenar a especificação, optou-se por também utilizar regras para a representação do comportamento concorrente no presente trabalho. Serão definidas regras de pré-condições para a execução de cada uma das tarefas modeladas. A habilitação das regras de pré-condições permite a execução da tarefa, sendo possível a execução de mais de uma tarefa ao mesmo tempo. Serão definidas também regras de pós-condições, a serem avaliadas ao término da execução de uma atividade, modelando restrições à sua execução. Provavelmente o tratamento temporal incluirá a representação da concorrência.

Esta maneira de representar a concorrência pode ser associada ao paradigma de orientação a objetos através da definição das tarefas como objetos. A representação das pré e pós-condições é feita diretamente no modelo de dados, nos objetos.

3 METODOLOGIA PARA UTILIZAÇÃO DO MODELO PROPOSTO

A metodologia proposta para a utilização do modelo deve incorporar a natureza interativa do desenvolvimento de sistemas de informação. A reutilização de especificações anteriormente realizadas será encorajada através de apoio à especificação. A metodologia apresentará formas de validação e de verificação das especificações construídas.

Será formada por: (1) uma linguagem para a utilização do modelo proposto; (2) uma base de informações para armazenar o conhecimento do domínio, o conhecimento do método de especificação utilizado e a própria especificação que está sendo construída; (3) uma ferramenta de análise do domínio; e (4) uma ferramenta para apoiar a especificação. A seguir faremos algumas considerações a respeito de técnicas de validação e de verificação, e a descrição sucinta de cada uma das partes citadas.

3.1 Validação e Verificação de Especificações

O desenvolvimento de sistemas de software é uma tarefa demorada, complexa e dispendiosa. A construção de especificações destes sistemas diminui sensivelmente o esforço na implementação dos sistemas propriamente ditos. A garantia de que estas especificações sejam as mais corretas possíveis é um fator importante na sua efetividade.

Três tipos diferentes de erros podem ser introduzidos em uma especificação, cometidos por pessoas diferentes, em diferentes momentos: (1) o próprio usuário que solicita o desenvolvimento do sistema, ao não especificar corretamente as suas aspirações; e (2) o projetista, ao construir uma especificação que não atenda corretamente aos requisitos solicitados pelo usuário, ou ao construir especificações incompletas, inconsistentes, que contém erros de utilização do formalismo empregado na especificação. Técnicas de **validação** são utilizadas para detectar os dois primeiros tipos de erros, procurando mostrar ao usuário como foram interpretados os requisitos por ele definidos. Os erros introduzidos na especificação propriamente dita podem ser detectados através de técnicas de **verificação**. Estas últimas se baseiam em provas rigorosas, formalmente justificáveis.

Procuram demonstrar que a especificação está sendo construída corretamente mostrando a consistência entre duas representações do mesmo comportamento, uma delas substancialmente mais detalhada do que a outra.

Uma especificação de requisitos deve apresentar as seguintes características [FAI 85]: corretude, completeza, consistência, rastreamento, funcionalidade, verificabilidade, não ser ambígua e poder ser facilmente modificada. A validação de uma especificação usa como principal critério a funcionalidade, enquanto que a sua verificação se preocupa principalmente com a corretude, além de verificar os demais aspectos.

Para possibilitar a **validação** das especificações construídas, procuramos considerar dois aspectos diferentes: (1) permitir ao usuário entender a especificação propriamente dita, através da utilização de um modelo o mais simples possível, representado através de uma linguagem bastante semelhante à linguagem natural, específica para o domínio considerado; e (2) a possibilidade de construção de um protótipo desta especificação, validando os requisitos através da simulação da execução do sistema especificado.

Quanto à **verificação**, como esta será a primeira especificação a ser construída, não é possível aplicar as técnicas que comparam duas representações. A única forma de efetuar uma verificação é verificando a consistência e a completude da especificação durante a sua construção [GOL 86, MAI 87]. Este tipo de verificação é efetuado através de um sistema que controla as informações que estão sendo definidas. Este sistema detém o conhecimento do método utilizado na especificação e do domínio de aplicação. Estes conhecimentos são geralmente expressos através de regras que são avaliadas a cada nova informação definida. As regras de verificação de consistência impedem que informações inconsistentes sejam definidas. As de verificação de completude podem interagir com o projetista, indicando o que falta para que a definição esteja completa. O sistema guia a especificação verificando a correção do produto com base nas regras de consistência. Um exemplo de utilização desta técnica pode ser encontrado em C-TODOS [PER 89].

As regras de verificação da construção da especificação serão incluídas na mesma base de informações em que é armazenada a especificação. Representarão o conhecimento a respeito do método e do modelo, guiando o projetista em seu trabalho.

3.2 Linguagem para Representação do Modelo

A escolha de uma linguagem para a representação dos requisitos se deu após uma análise detalhada dos diferentes meios utilizados em métodos existentes. Grande número destes métodos permite a apresentação através de representações gráficas, com o objetivo de facilitar o seu entendimento por parte do usuário. Deixam livre, no entanto, a rotulação de figuras e de setas, nas quais é utilizada linguagem natural, podendo ser introduzidas ambigüidades na especificação. Além disso, as técnicas gráficas requerem um conhecimento prévio do formalismo empregado, o que pode dificultar o entendimento por parte de um usuário comum. Outros métodos são completamente formais, utilizando embasamento matemático avançado, não sendo muito fáceis de serem compreendidos. O problema consiste em encontrar um meio termo - um método simples mas que seja rigoroso em sua representação, dando assim origem a uma especificação formal. A opção recaiu em uma linguagem, a mais próxima possível daquela utilizada em sistemas de informação de escritórios, mas que apresente regras fixas para sua utilização. A escrita da especificação seria efetuada por um projetista, o qual seguiria as regras definidas para a linguagem. Para o usuário só é requerido o entendimento da especificação já construída.

Na definição de uma linguagem para representar o modelo definido deverá ser feito um balanço entre seu poder de expressão e as dificuldades encontradas para sua utilização. Quanto mais complexa for a linguagem, mais fiel será a modelagem efetuada, mas mais difícil será sua utilização. Um formalismo muito complexo leva à introdução de muitos erros na especificação. Já um formalismo muito simples, fácil de ser utilizado, não permitirá a representação de todos os aspectos relevantes do sistema.

A linguagem proposta para a utilização do modelo deverá ser ao mesmo tempo simples e precisa. Simples para que possa ser entendida por usuários não versados em computação, utilizando palavras normalmente empregadas em sistemas de informação de escritórios. Precisa na medida em que é formada por construções fixas, definidas através de uma gramática (BNF). A eventual dificuldade de construção de uma especificação através de uma linguagem limitada não deverá ser considerada, uma vez que esta construção é prevista para os projetistas da aplicação. A linguagem deverá apresentar forma semelhante àquela definida para o método T.A.& A. (ANEXO 3).

3.3 Base de Informações

Optamos por utilizar somente uma base de informações para armazenar tanto as informações necessárias à especificação como a especificação propriamente dita, como em [GRE 86], [LUT 88] e nos projetos TODOS [HEI 88] e ITHACA [PRO 89].

A base de informações deverá, portanto, armazenar os objetos a serem reutilizados, os objetos relativos à especificação que está sendo construída e o conhecimento do método utilizado. A forma como estas informações serão estruturadas está diretamente relacionada com o modelo a ser definido, levando em consideração a possibilidade de recuperação rápida e eficiente das informações armazenadas. A possibilidade de armazenar o conhecimento do método também deverá ser considerada.

3.4 Ferramentas

A metodologia deverá apresentar duas ferramentas, uma para a análise do domínio de aplicação, outra para apoiar a especificação. Ambas as ferramentas devem ser interativas, possuindo interfaces claros para facilitar sua utilização. Não deverão ser utilizadas por usuários comuns, mas por projetistas do sistema.

A ferramenta de análise do domínio de aplicação será utilizada para definir os objetos a serem reutilizados. Vai ser a responsável pelo fornecimento do conhecimento do domínio. A aplicação desta ferramenta deverá ser efetuada em um momento anterior ao da especificação de um sistema. Sistemas que utilizam este tipo de análise precursora para a montagem da base de conhecimentos inicial são o projeto ITHACA [TAK 90] e DRACO [NEI 84].

A ferramenta de apoio à especificação deverá permitir a especificação a partir da reutilização de objetos recuperados da base de conhecimentos, eventualmente modificados e/ou integrados. Será basicamente um editor da linguagem, dirigido por sua sintaxe. Além de verificar as construções da linguagem, esta ferramenta deverá guiar a especificação, proporcionando a verificação de sua corretude e de sua consistência. Isto é realizado com base no conhecimento do próprio método de especificação utilizado, conhecimento este que deverá estar armazenado na mesma base de conhecimentos utilizada.

Um exemplo de editor dirigido por sintaxe para especificações orientadas a objetos pode ser encontrado em [CAR 90]. A ferramenta gráfica RECAST [BEL 91] apoia a especificação guiando o usuário na escolha das classes armazenadas e possibilitando sua modificação.

4 RESULTADOS PRELIMINARES OBTIDOS

Foram realizados diversos trabalhos preliminares de especificação de Sistemas de Informação, com o intúito de obter a experiência necessária para desenvolver a tese propriamente dita. Inicialmente foi feito um estudo dos "Métodos de Especificação para Sistemas de Informação de Escritórios" [EDE 89b] encontrados na literatura. Diversos métodos de especificação foram encontrados, baseados em diferentes modelos de dados e apresentando ambientes de apoio à especificação através de ferramentas apropriadas. Este estudo mostrou a importância da definição precisa de um modelo de dados apropriado às características das aplicações que se deseja especificar, além da necessidade da construção de ferramentas de apoio à especificação.

Com o objetivo de testar a construção de uma ferramenta de apoio à especificação, foi escolhido um modelo de dados, implementado através de um "Ambiente de Desenvolvimento de Protótipos de Bancos de Dados Dedutivos Temporais" [EDE 89a]- o ambiente YPY (ANEXO 1). Para esta experiência foi escolhido o modelo relacional binário [GRI 81], em parte devido à facilidade de sua implementação mas também pelo fato de se apropriar à modelagem orientada a objetos. Este modelo tem origem nos trabalhos de Abrial [ABR 74].

Os conceitos primitivos do modelo relacional binário são entidades, nomes de entidades e relacionamentos binários entre pares de entidades, além de restrições. Todas as informações são representadas através de entidades e de instâncias dos relacionamentos binários. As restrições servem para delimitar a possibilidade de definição de entidades e de instâncias dos relacionamentos, possibilitando a modelagem de restrições estáticas. Quando o sistema implementado permite comparar dois estados diferentes da base de informações, pode-se utilizar as restrições para representar transições de estado inválidas, possibilitando assim a modelagem também de restrições dinâmicas.

No ambiente implementado, todas as informações são representadas através de classes, entidades definidas nestas classes, relações binárias entre entidades destas classes e de tuplas definidas nestas relações. As restrições são representadas através de regras de violação de integridade, testadas a cada vez que uma nova informação é inserida no banco de dados. Como o banco de dados implementado é temporal (histórico, isto é,

nenhuma informação é removida, ficando somente não-habilitada em caso de pedido de remoção), é possível incluir regras de violação que testem as propriedades dinâmicas do sistema especificado.

Diversos sistemas de modelagem encontrados na literatura utilizam este modelo de dados relacional binário. É o caso, por exemplo, de TAXIS [MYL 80, NIX 87], uma linguagem utilizada para a construção interativa de Sistemas de Informação, implementada na Universidade de Toronto. TAXIS apresenta um modelo de dados semântico, utilizando o conceito de redes semânticas para a modelagem de dados e de procedimentos, associado ao conceito de bancos de dados relacionais. É uma linguagem orientada a objetos, suportando mecanismos de abstração (generalização, agregação) com hierarquias que apresentam herança de atributos.

O ambiente YPY foi posteriormente integrado à metodologia T.A.& A. [HOP 88], em desenvolvimento conjunto pelo CPGCC/UFRGS e PPGA/UFRGS [EDE 90a,b] (ANEXO 2). A metodologia T.A.& A. se destina à coleta, organização e análise dos requisitos de Sistemas de Informação de Escritórios. O objetivo deste trabalho foi o de construir um ambiente mais amplo para a especificação, utilizando inicialmente a metodologia T.A.& A. para a análise e organização dos requisitos coletados, e efetuando posteriormente a especificação e a prototipação através do ambiente YPY.

O trabalho anterior mostrou a necessidade de formalizar a metodologia T.A.& A., até então tratada de forma conceitual [EDE 90c]. Um resumo deste trabalho está apresentado no ANEXO 3. Inicialmente foi definida uma linguagem para ser empregada quando da utilização do método T.A.& A. [FIC 89, EDE 90c]. A formalização do método se baseou nesta linguagem, tendo sido efetuada através de um subconjunto do formalismo Z [DEL 82]. Este trata da definição de conjuntos de entidades e de relações binárias entre estas entidades. A escolha deste formalismo deveu-se ao fato de empregar os mesmos conceitos utilizados no modelo de dados do ambiente YPY, possibilitando a posterior utilização desta especificação na integração dos dois métodos.

A realização dos exames de qualificação proporcionou a oportunidade de aprofundamento em áreas diferentes da de especificação formal. Foram escolhidos dois assuntos que trouxeram subsídios à implementação de ferramentas de apoio à especificação. Um dos aspectos importantes deste tipo de ferramenta é a necessidade de apresentarem algum mecanismo de verificação/validação. Para o primeiro exame foi escolhida a área de

Engenharia de Software, tendo sido realizado um estudo das técnicas existentes de validação e de verificação de produtos de software [EDE 91a]. Outro aspecto importante das ferramentas de apoio é a maneira como são armazenadas as informações. Uma pesquisa nas ferramentas existentes mostrou a grande utilização de Bases de Conhecimento. O segundo exame foi realizado na área de Inteligência Artificial, especificamente em formas de representação de conhecimento em Engenharia do Conhecimento [EDE 91b].

5 DETALHAMENTO DO TRABALHO A SER REALIZADO

No presente trabalho nos propomos as seguintes tarefas:

a) definição do modelo de dados a ser utilizado;

b) definição de uma metodologia para a utilização deste modelo de dados:

b1) definição de uma linguagem para a utilização do modelo, através de uma gramática (BNF);

b2) especificação de uma ferramenta de análise de domínio;

b3) especificação de uma ferramenta de apoio à especificação;

b4) construção de um protótipo da ferramenta de apoio, provavelmente em PROLOG;

c) aplicação da ferramenta construída a um problema do domínio de Sistema de Informação de Escritórios (estudo de caso), com o objetivo de validar o modelo de dados proposto.

6 ESQUEMA DA TESE

A tese deverá apresentar o seguinte esquema preliminar:

1 - INTRODUÇÃO

Motivação que levou ao desenvolvimento da tese; importância do assunto no contexto de especificações de Sistemas de Informação.

2 - MODELO PARA ESPECIFICAÇÃO DE SISTEMAS DE INFORMAÇÃO

Descrição dos aspectos fundamentais que o modelo deve suportar; apresentação de alguns modelos atuais; apresentação do modelo proposto.

3 - METODOLOGIA PARA O MODELO PROPOSTO

Apresentação da metodologia; definição da linguagem a ser utilizada para representação de informações segundo este modelo; estudos de melhores formas para estruturar a base de informações; especificação das ferramentas de análise de domínio e de apoio à especificação.

4 - PROTÓTIPO DA FERRAMENTA DE APOIO À ESPECIFICAÇÃO

Descrição da ferramenta implementada, ressaltando quais os aspectos que não foram observados e quais as dificuldades encontradas.

5 - VALIDAÇÃO DO MODELO

Apresentação dos resultados obtidos no estudo de caso realizado. Avaliação do modelo.

6 - CONCLUSÃO

Conclusões tiradas do trabalho realizado; idéias para posteriores continuações deste trabalho.

7 PLANO DE AÇÃO

A contar de março de 1992 teremos dois anos para o desenvolvimento da tese proposta. O seguinte cronograma preliminar deverá ser seguido:

6 meses - definição do modelo de dados, com determinação da forma de modelar as informações temporais e a concorrência;

6 meses - especificação do método com definição da linguagem a ser utilizada; aplicação preliminar manual a alguns estudos de caso simples; especificação das ferramentas de análise de domínio e de apoio à especificação.

6 meses - implementação do protótipo da ferramenta e sua aplicação a um estudo de caso pré-selecionado;

6 meses - conclusão da documentação da tese.

BIBLIOGRAFIA

[ABR 74] ABRIAL, J.R. Data Semantics. In: **Data Management Systems**. Amsterdam: North Holland, 1974. p.1-59.

[AGH 90] AGHA, G. Concurrent object-oriented programming. **Communications of the ACM**, New York, v.33, n.9, p.125-141, Sept. 1990.

[ALB 86] ALBANO, A. et al. A Strongly typed, interactive object-oriented database programming language. In: INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED DATABASE SYSTEMS, 1986. **Proceedings...** 1986.

[ALL 83] ALLEN, J.F. Mantaining knowledge about temporal intervals. **Communications of the ACM**, New York, v.26, n.1,p.832-43, Nov. 1983.

[ALL 84] ALLEN, J.F. Towards a general theory about action and time. **Artificial Inteligence**, v.23, p.123-54, July 1984.

[AME 90] AMERICA, P.; LINDEN, F.V.D. A Parallel object-oriented language with inheritance and subtyping. **SIGPLAN Notices**, New York, v.25, n.10, p.161-168, Oct. 1990.

[ARA 90] ARAPIS, C. Specifying object life-cycles. In: **Object Management**. Genève: Universitè de Genève, 1990. p.197-225.

[ARA 91] ARAPIS, C. Specifying object interactions. In: **Objects Composition**. Genebra, Université de Genève, 1991. p.303-22.

[BAN 88] BANCILLON, F. et al. The Design and implementation of O2, an Object-Oriented Database System. In: DITTRICH,K.R. (ed.) **Advances in Object-Oriented Database Systems**. Berlin: Springer-Verlag, 1988. p.1-22.

[BEL 91] BELLINZONA, R.; FUGINI, M.G. **RECAST PROTOTYPE DESCRIPTION**. Milano: Politecnico di Milano, 1991. (Tech. Repport).

[BLA 87] BLACK, A. et al. Distribution and abstract types in Emerald. **IEEE Transactions on Software Engineering**, New York, v.13, n.1, p.65-76, Jan. 1987.

[BOE 88] BOEHM, B.W. A Spiral model of software development and enhancement. **Computer**, Los Alamitos, v.21, n.5, p.61-72, May 1988.

[BOC 90] BOCHMANN, G. Protocol specification for OSI protocol networks and ISDN systems. **Computer Networks**, New York, v.18, n.3, p.167-85, Apr. 1990.

[BOO 86] BOOCH, G. Object-oriented development. **IEEE Transactions on Software Engineering**, New York, v.12, n.2, p.211-21, Feb. 1986.

[CAC 90] CACACE, F. te al. Integrating object-oriented data modeling with a rule-based programming paradigm. **SIGMOD Record**, New York, v.19. n.2, p.225-36, June 1990.

[CAR 90] CARRINGTON, D.; HAYES, I.; WELSH, J. A Syntax-directed editor for object-oriented specifications. In: INTERNATIONAL CONFERENCE TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES & SY STEMS, 3., 1990, Sydney. **Proceedings...** Sydney: TOOLS Pacific, 1990. p.46-57.

[CLI 88] CLIFFORD, J; RAO, A. A Simple, general structure or temporal domains. In: **Temporal Aspects in Information Systems**. Amsterdam: North-Holland, 1988. p.17-28.

[COA 91] COAD, P.; JOURDON, E. **Object-Oriented Analysis**. Englewood Cliffs: Prentice Hall, 1991. 233p.

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

[DEA 85] DeANTONELLIS, V.; DiLEVA, A. DATAID-1: A Database design methodology. **Information Systems**, U.S.A, v.10, n.2, p.181-95, 1985.

[DEA 91] DeANTONELLIS, V.; PERNICI, B.; SAMARATI, P. F-ORM Method: a F-ORM methodology for reusing specifications. In: **Object Oriented Approach in Information Systems**. Amsterdam: North-Holland, 1991. (Proceedings of the IFIP TC8/WG8.1 Working Conference - Oct. 28-31, 1991, Quebec City, Canada). p.117-135.

[DEL 82] DELOBEL, C.; ADIBA, M. **Bases de Donnés et Systèmes Relationnels**. Paris: Dunod Informatique, 1982. 449p.

[EDE 89a] EDELWEISS, N.; COSTA, A.C.R. Um Ambiente para desenvolvimento de protótipos de bancos de dados dedutivos temporais. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 4., 5-7 abril, 1989, Campinas. **Anais...** Campinas: R. Vieira Gráf. e Ed. Ltda, 1989. p.163-73.

[EDE 89b] EDELWEISS, N. **Métodos de Especificação para Sistemas de Informação de Escritórios**. Porto Alegre: CPGCC - UFRGS. 1989. 117p. (T.I.123).

[EDE 90a] EDELWEISS, N.; OLIVEIRA, J.P.M. T.A.& A. and YPY: Integration of two OIS specification methods. In: CONFERENCIA INTERNACIONAL DE LA SOCIEDAD CHILENA DE CIENCIA DE LA COMPUTATION, 10, 23-27 jul., 1990, Santiago., Chile. **Actas...** Santiago: SCCC, 1990. p.325-36.

[EDE 90b] EDELWEISS, N.; OLIVEIRA, J.P.M. **Integração do Método T.A.& A. e do Ambiente YPY na Especificação de um SIE.** Porto Alegre: CPGCC - UFRGS, 1990. 21p. (R.P.129)

[EDE 90c] EDELWEISS, N. **Especificação Formal do Método T.A.& A.** Porto Alegre: CPGCC - UFRGS, 1990. 56p. (T.I.179).

[EDE 91a] EDELWEISS, N. **Um Estudo de Técnicas de Validação e de Verificação de Produtos de Software.** Porto Alegre: CPGCC - UFRGS, 1991. 42p. (R.P.159).

[EDE 91b] DELWEISS, N. **Representação de Conhecimento em Engenharia do Conhecimento.** Porto Alegre: CPGCC - UFRGS, 1991. 74p. (R.P.163).

[FAI 85] FAIRLEY, R. **Software Engineering Concepts.** Singapore: McGraw-Hill, 1985. 364p.

[FIC 89] FICHMAN, V.R. **Especificação de uma Ferramenta para Análise de Requisitos em Sistemas de Informação de Escritórios Baseada no Uso de Hipertexto.** Porto Alegre: CPGCC - UFRGS, 1989. 117p. Projeto de Diplomação.

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

[FUG 91] FUGINI, M.G.; NIERSTRASZ, O.; PERNICI, B. Application development through Reuse: the Ithaca Tools Environment. TSCHRITZIS,D. (ed.) **Objects Composition.** Genebra: Université de Genève, 1991. p.99-114.

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

[GAN 91] GANDRIEAU, M.-A.; DURIN, B. Identification and classification of reusable elements in space domain. REBOOT Workshop on REUSE - ESPRIT Project 5327. Grenoble, Sept. 26-27, 1991.

[GIB 90] GIBBS,S.; TSICHRITZIS, D. et al.; Class management for software communities. **Communications of the ACM**, New York, v.33, n.9, p.90-103, Sept. 1990.

[GIR 90] GIRARDI, M.R. **Uma Ferramenta de Apoio à Reutilização de Software no Desenvolvimento Orientado a Objetos**. Porto Alegre: CPGCC - UFRGS, 1990. 228p. Tese de Mestrado.

[GOL 86] GOLDBERG, A.T. Knowledge-based programming: a survey of program design and construction techniques. **IEEE Transactions on Software Engineering**, New York, v.12, n.7, p.752-68, July 1986.

[GRE 86] GREENSPAN, S.J.; BORGIDA, A.; MYLOPOULOS, J. A Requirements modeling language and its logic. In: . New York: Springer-Verlag, 1986. p.471-502.

[GRI 81] GRIETHUYSEN, J.J. et al (eds.) **Concepts and Terminology for the Conceptual Schema**. New York: ANSI, 1981. (ISO TC97/SC5/WG3).

[HEI 88] HEIJMINK, F. et al. Development of Tools for Designing OIS. In: **Information Technology for Organizational Systems**. Brussels: Elsevier, 1988. p.66-73.

[HEL 91] HELD, G. (ed.) **GRAPES Language Description**: Syntax, Semantics and Grammar of GRAPES-86. Berlin: Siemens-Nixdorf Informationssysteme AG, 1991. 304p.

[HEN 90a] HENDERSON-SELLERS, B. Three methodological frameworks for object-oriented systems development. In: INTERNATIONAL CONFERENCE TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES & SYSTEMS, 3., 1990, Sydney. **Proceedings...** Sydney: TOOLS Pacific, 1990. p.118-31.

[HEN 90b] HENDERSON-SELLERS, B.; EDWARDS, J.M. The Object-oriented systems life cycle. **Communications of the ACM**, New York, v.33, n.9, p.142-59, Sept. 1990.

[HOP 88] HOPPEN, N.; OLIVEIRA, J.P.M.; LIMA, L.M.R. Metodologia para modelagem de escritório baseada no nível de estruturação das atividades. In: REUNIÃO ANUAL DA ASSOCIAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO, 12., set. 26-8, 1988, Natal. **Anais...** Natal: 1988. p.247-63.

[HOR 87] HORNICK, R.K.; ZDONIK, B.Z. A shared, segmented memory system for an Object-Oriented Database. **ACM Transactions on Office Information Systems**, New York, v. 5, n.1, p. 79-95, Jan. 1987.

[KIM 87] KIM, W. et al. Composite objects support in an Object-Oriented Database System. **SIGPLAN NOTICES**, New York, v.22, n.12, p.118-25, Oct. 1987.

[KON 82] KONSYNSKI, B.R.; BRACKER, L.C.; BRACKER JR, W.E. A Model for specification of office communications. **IEEE Transactions on Communications**, New York, v.30, n.1, p.27-36, Jan. 1982.

[KON 88] KONSTANTAS, D.; NIERSTRASZ, O.; PAPATHOMAS, M. An Implementation of Hybrid - a concurrent, object-oriented language. In: **Active Object Environments**. Genève: Universitè de Genève, 1988. p.32-50.

[KOR 90] KORSON, T.; McGREGOR, J.D. Understanding object-orientation: a unifying paradigm. **Communications of the ACM**, New York, v.33, n.9, p.40-60, Sept. 1990.

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

[LAD 88] LADDEN, R.M. Survey on issues to be considered in the development of an object-oriented development methodology for ADA. **ACM Software Engineering Notes**, New York, v.13, n.3, p.24-31, July 1988.

[LEE 88] LEE, R.M. Bureaucracies as deontic systems. **ACM Transactions on Office Information Systems**, New York, v.6, n.2, p.87-108, Apr. 1988.

[LEC 88] LECLUSE, C.; RICHARD, P.; VELEZ, F. O_2, an Object-oriented data model. **ACM SIGMOD Record**, New York, v.17, n.3, p.424-33, Sept. 1988.

[LIU 89] LIU, L.-C.; HOROWITZ, E. A Formal model for software project management. **IEEE Transactions on Software Engineering**, v.15, n.10, p.1280-93, Oct. 1989.

[LOU 91] LOUCOPOULOS, P.; THEODOULIDIS, B.; PANTAZIS, D. Business rules modelling: conceptual modelling and object-oriented specifications. In: **Object Oriented Approach in Information Systems**. Amsterdam: North-Holland, 1991. p.323-342. Proceedings of the IFIP TC8/WG8.1 Working Conference - Quebec City, Canada, Oct. 28-31, 1991.

[LUT 88] LUTZE, R. Customizing cooperative office procedures by planning. **SIGOIS BULLETIN**, New York, v.9, n.2,3, p.63-77, Apr./July 1988.

[MAI 87] MAIOCCHI, R.; PERNICI, B. Verification and refinement of office procedures. In: IEEE COMPUTER SOCIETY OFFICE AUTOMATION SYMPOSIUM, Apr. 27-9, 1987, Gaithersburg. **Proceedings...** Washington: IEEE, 1987. p.206-16.

[MAI 91] MAIOCCHI, R.; PERNICI, B.; BARBIC, F. Automatic deduction of temporal information. Udine: University of Udine - Dipartimento de Matematica e Informatica, 1991. 58p. (Research Report).

[MEY 90] MEYER, B. Lessons for the design of the Eiffel libraries. **Communications of the ACM**, New York, v.33, n.9, p.68-88, Sept. 1990.

[MYL 80] MYLOPOULOS, J,; BERNSTEIN, P.A.; WONG, H.K.T. A Language facility for designing database-intensive applications. **ACM Transactions on Database Systems**, v.5, n.2, p.185-207, Jun. 1980.

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

[NAV 88] NAVATHE, S.B.; AHMED, R. TSQL: A Language interface for history databases. In: **Temporal Aspects in Information Systems**. Amsterdam: North-Holland, 1988. p.109-122.

[NEI 84] NEIGHBORS, J.M. The Draco approach to constructing software from reusable components. **IEEE Transactions on Software Engineering**, New York, v.10, n.5, p.564-73, Sept. 1984.

[NIE 90] NIERSTRASZ, O. A Guide to specifying concurrent behavior with Abacus. **Object Management**. Genève: Universitè de Genève, 1990. p.267-93.

[NIX 87] NIXON,B.; CHUNG, L. et al. Implementation of a compiler for a semantic data model: experiences with Taxis. In: ACM ANNUAL CONFERENCE ON MANAGEMENT OF DATA, May 22-29, 1987, San Francisco. **SIGMOD Record**, v.16, n.3, p.118-31, 1987.

[PAP 90] PAPATHOMAS, M.;& KONSTANTAS, M. Integrating concurrency and object-oriented programming. In: **Object Management**. Genève: Universitè de Genève, 1990. p.229-43.

[PER 89] PERNICI, B. et al. C-TODOS: An Automatic tool for office system conceptual design. Communications of the ACM, New York, 7(4):378-419, Oct. 1989.

[PER 90] PERNICI, B. Class design and meta-design. In: TSICHRITZIS,D. (ed.) Objects management. Geneve, University of Geneve - Centre Universitaire d'Informatique, 1990. p.117-31. (Technical Repport).

[PIN 90] PINTADO, X. Selection and exploration in an object-oriented environment: the affinity browser. SICHRITZIS,D. (ed) Object Management. Universitè de Genève, 1990. p.79-105.

[PRI 90] PRICE, R.T.; GIRARDI, R. A Class retrieval tool for an object oriented environment. In: INTERNATIONAL CONFERENCE TECHNOLOGY OF

OBJECT-ORIENTED LANGUAGES & SYSTEMS, 3., 1990, Sydney. **Proceedings...** Sydney: TOOLS Pacific, 1990. p.26-36.

[PRO 89] PROFROCK, A.-K. et. al. ITHACA: an Integrated toilkit for highly advanced computer applications. In: **Object Oriented Development**. Genève: Université de Genève, 1989. p.321-44.

[RAM 89] RAMSEY, N. Developping formally verified Ada programs. **Software Engineering Notes**, New York, v.14, n.3, p.257-65, May 1989.

[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.220-30, June 1990.

[SEG 88] SEGEV, A.; SHOSHANI, A. Modeling temporal semantics. In: **Temporal Aspects in Information Systems**. Amsterdam: North-Holland, 1988. p.47-57.

[SRI 88] SRIPADA, S.M. A Logical framework for temporal deductive databases. In: VERY LARGE DATABASES CONFERENCE, 14., Los Angeles, 1988. **Proceedings**. p.171-82.

[STA 84] STANDISH, T.A. An Essay on Software reuse. **IEEE Transactions on Software Engineering**, New York, v.10, n.5, p.494-7, Sept. 1984.

[SU 91] SU, S.Y.W.; CHEN, H.-H.,M. A Temporal knowledge representation model OSAM*/T and its query language OQL/T. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 17., Sept. 3-6, 1991, Barcelona. **Proceedings....** Barcelona: Industria Grafica, 1991. p.431-42.

[SUB 90] SUBRAMANIAN, R. PENDS: An Approach to modeling and retrieving 'pending' information. **SIGOIS Bulletin**, New York, v.11, n.2/3, p.158-83, 1990.

[SUT 91] SUTCLIFFE, A.G. Object oriented systems analysis: the abstraction question. In: **Object Oriented Approach in Information Systems**. Amsterdam: North-Holland, 1991. p.23-37. Proceedings of the IFIP TC8/WG8.1 Working Conference - Quebec City, Canada, Oct. 28-31, 1991.

[TAK 90] TAKAHASHI, 1990.

[WIR 90] WIRFS-BROCK, R.; JOHNSON, R.E. Surveying concurrent research in object-oriented design. **Communications of the ACM**, New York, v.33, n.9, p.104-124, Sept. 1990.

ANEXO 1

UM AMBIENTE PARA DESENVOLVIMENTO DE PROTÓTIPOS DE BANCOS DE DADOS DEDUTIVOS TEMPORAIS

Nina Edelweiss* , Antônio Carlos da Rocha Costa**

SUMÁRIO

Este trabalho apresenta as principais características de um Ambiente para Desenvolvimento de Banco de Dados Dedutivos Temporais. São apresentados os modelos lógicos adotados para a modelagem dos dados, das restrições de integridade e do tempo na base de dados.

ABSTRACT

This work presents the major features of a Prototype Development Environment for Deductive Temporal Data Bases. The logical model adopted for the modelling of data, integrity restrictions and time representation in the data base are presented.

* Engenheira Eletricista (UFRGS,1968), Mestre em Ciência da Computação (UFRGS, 1975); áreas de interesse: especificações formais, sistemas de informação; Departamento de Informática, CPGCC, UFRGS - Porto Alegre, RS - CEP 90.000

** Engenheiro Eletrônico (UFRGS,1977), Mestre em Ciência da Computação (UFRGS,1980); área de interesse: inteligência artificial; Departamento de Informática, CPGCC, UFRGS - Porto Alegre - RS - CEP 90.000

Trabalho parcialmente financiado por FINEP e SID-INFORMATICA (Projeto ESTRA).

1. INTRODUÇÃO

O objetivo deste trabalho foi a criação de um ambiente para o desenvolvimento de protótipos de Bancos de Dados Dedutivos Temporais. O sistema que gerencia este ambiente foi totalmente implementado, tendo sido utilizada a linguagem de programação em lógica PROLOG [STE86]. Pode ser executado em computadores PC-compatíveis.

2. REPRESENTAÇÃO DAS INFORMAÇÕES

Como o objetivo era implementar um Banco de Dados Dedutivo, foi buscado um modelo de dados lógico para representar as informações que facilitasse, o mais possível, a implementação deste conceito. Na hipótese de que um modelo lógico simples é capaz de cumprir esse requisito, foi escolhida, como base do modelo, a lógica das **classes** e das **relações** [TAR54].

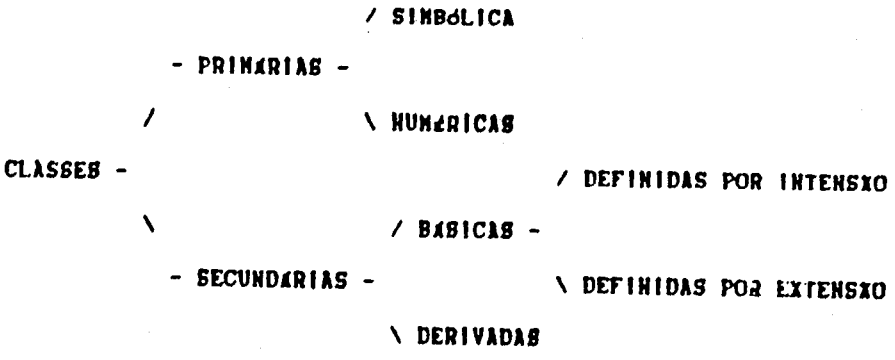
No modelo adotado devem ser definidas determinadas **classes** de dados e as relações que podem ocorrer entre entidades aceitas nestas classes. As informações propriamente ditas são armazenadas em forma de **tuplas** definidas para as relações. Por exemplo:

classe: nome
entidades válidas: maria
joao

classe: valor
entidades válidas: valores numéricos inteiros

relação: idade(nome,valor)
tuplas definidas: maria,18
joao,20

O sistema admite o seguinte conjunto de classes:



As classes primárias servem de base para a definição de novas classes pelo usuário - as classes secundárias. A classe primária simbólica contém todos os símbolos reconhecidos pelo PROLOG, e servirá de base para a definição das classes secundárias básicas definidas por extensão, com tipo simbólico. As classes primárias numéricas dividem-se em inteiras e reais e são a base das classes secundárias básicas definidas por intensão, com tipo inteiro e real.

A definição de uma classe secundária básica de tipo simbólico deve ser feita sempre por extensão: a criação da classe somente associa o seu nome ao tipo das entidades que nela serão aceitas. Estas entidades deverão ser fornecidas posteriormente, uma a uma. Nas classes numéricas, entretanto, a sua criação já define que as entidades aceitas serão todos os valores numéricos daquele tipo (inteiro ou real), pois elas deverão ser sempre definidas por intensão.

Poderão ser formadas novas classes - as classes secundárias derivadas - a partir de classes já definidas. Para isto são utilizadas operações entre classes. As operações disponíveis no sistema são as seguintes:

- união de duas classes;
- intersecção de duas classes;

- complemento de uma classe;

- complemento relativo de uma classe em relação a outra.

As informações a respeito das classes definidas e das entidades aceitáveis em cada uma delas ficarão assim armazenadas na base de dados:

a) classes secundárias básicas definidas por intensão:

```
classe(NomeInteiro,inteiro).
entidade(NomeInteiro,X) :- not var(X),integer(X).
classe(NomeReal,real).
entidade(NomeReal,X) :- not var(X),Positivo(X),
                        (integer(X),!;float(Posit)).
```

Como pode ser observado acima, nestes dois tipos de classes o sistema armazena, em um fato, o nome da classe e o tipo de suas entidades. Em se tratando de classes definidas por intensão, o sistema também armazena a regra de formação das entidades aceitáveis em cada uma delas. Isto é feito no mesmo momento em que a classe for definida.

b) classes secundárias básicas definidas por extensão:

```
classe(NomeClasse,simbolico).
entidade(NomeClasse,Entidade).
```

Neste caso, o usuário deverá fornecer as entidades aceitáveis na classe em um momento posterior e independente ao da definição da classe. Cada entidade aceitável será armazenada em um fato, associada ao nome da classe a que pertence.

c) classes secundárias derivadas:

Quando for definida uma classe derivada, o sistema armazenará o nome da classe juntamente com seu tipo (derivada), a operação que vai ser efetuada para obter a nova classe e o(s) nome(s) da(s) classe(s) de origem. Além disso, o sistema também armazenará a regra de formação das entidades válidas nesta classe, de acordo com a operação efetuada. Por exemplo, no

caso de uma classe definida através de uma operação de união de outras duas classes, as informações armazenadas seriam as seguintes:

```
classe(Nome,derivada,uniao,(Classe1,Classe2)).
entidade(Nome,X) :- entidade(Classe1,X) ; entidade(Classe2,X).
```

O sistema admite as seguintes estruturas para as relações :

```
                          / SIMBÓLICAS
            - PRIMÁRIAS -
           /              \ NUMÉRICAS
RELAÇÕES -                            / TOTAIS
           \              / BÁSICAS -
            - SECUNDÁRIAS -           \ PARCIAIS
                          \ DERIVADAS
```

As relações primárias operam sobre as classes primárias e são relações pré-definidas do PROLOG (==, <>, >=, etc). As relações secundárias são aquelas definidas pelo usuário.

As relações secundárias básicas são definidas diretamente entre classes. Podem ser totais ou parciais. Uma relação básica total entre duas classes define como aceitáveis todas as tuplas que envolvam uma entidade de cada uma das classes, na ordem da definição (produto cartesiano). Uma relação básica parcial entre duas classes aceita todas as possíveis tuplas entre entidades das duas classes, definidas pelo usuário, uma a uma.

Uma relação secundária derivada é aquela que é obtida a partir de relações já definidas, através de operações entre relações. As operações que estão implementadas são as seguintes:

- união de duas relações;
- intersecção entre duas relações;
- complemento de uma relação;

- composição de duas relações;

- inversa de uma relação.

As informações a respeito das relações e das tuplas aceitáveis em cada uma delas são armazenadas na base de dados através dos seguintes fatos:

a) relação básica total :

```
relacao(Nome,total,Classe1,Classe2).
tupla(Relação,Entidade1,Entidade2) :-
    entidade(Classe1,Entidade1),entidade(Classe2,Entidade2).
```

Como pode ser observado acima, o nome da relação é armazenado em um fato, juntamente com o seu tipo (total) e com o nome das duas classes que relaciona. No mesmo momento da definição de relação, o sistema armazena a regra que define as tuplas válidas nesta relação, que são todos os possíveis pares de entidades aceitáveis nas duas classes em questão.

b) relação básica parcial :

```
relação(Nome,parcial,Classe1,Classe2).
tupla(Relação,Entidade1,Entidade2).
```

Neste caso as tuplas serão fornecidas pelo usuário por extensão. As entidades das tuplas deverão ser aceitáveis nas classes correspondentes.

c) relação derivada :

Quando a relação definida for do tipo derivada, o sistema armazenará, além do nome e do tipo desta relação, qual a operação que foi utilizada para a sua formação, e o(s) nome(s) da(s) relação(ões) que serviu (ram) de base, além da regra que define as suas tuplas válidas. Por exemplo, informações armazenadas para uma relação derivada obtida a partir da operação de união de duas relações:

```
relação(Nome,derivada,uniao,(Relação1,Relação2)).
```

```
tupla(Relação,Entidade1,Entidade2) :-
        tupla(Relação1,Entidade1,Entidade2)
    ; tupla(Relação2,Entidade1,Entidade2).
```

Nota-se que foi adotado utilizar somente relações binárias, relacionando, portanto, somente duas classes. A representação de relações n-árias deve ser feita com o auxílio de n relações binárias envolvendo n+1 classes (as n classes originais, mais uma classe de entidades abstratas, representativas de cada tupla da relação n_ária [KOU79]).

3. TRATAMENTO DO TEMPO

Para o tratamento do tempo na base de dados foi utilizada a idéia colocada em [KOU86]. Foi assumido que todas as atualizações da base de dados (inclusão ou retirada de informações) seriam feitas na ordem temporal de sua ocorrência no mundo real. Não é possível, portanto, fazer alguma atualização referente ao passado. Isto possibilitou a utilização das funções embutidas no sistema operacional para fornecerem o dia (dia, mes e ano) e a hora (hora, minuto e segundo) atuais, que são então associados à modificação efetuada.

O tempo não foi incorporado diretamente às tuplas, mas através da definição de fatos específicos. A cada inclusão de uma informação à base de dados, será também incluído um fato associando os dados desta informação ao dia e à hora daquele instante. A cada solicitação de retirada será incluído um fato semelhante. O efeito da operação de remoção será, portanto, não uma retirada mas o acréscimo de novos conhecimentos a respeito do fim do tempo de validade desta informação. A informação propriamente dita, conforme mostrada no item anterior, será incluída somente no momento de sua primeira definição, ficando presente a partir deste momento. O fato de ela estar, em algum momento posterior, definida ou não, será determinado

pela informação correspondente aos momentos de definição ou remoção. Para a manipulação das informações contidas na base de dados foi necessária a definição de atributos que indicassem a sua existência ou não no instante considerado.

O instante de definição de informações é representado na base de dados através dos seguintes fatos:

```
cRclasse(NomeClasse,Dia,Hora).
cRentidade(Classe,NomeEntidade,Dia,Hora).
cRrelacao(NomeRelacao,Dia,Hora).
cRtupla(Relacao,Entidade1,Entidade2,Dia,Hora).
```

A remoção de informações é representada pelos fatos:

```
dRclasse(NomeClasse,Dia,Hora).
dRentidade(Classe,NomeEntidade,Dia,Hora).
dRrelacao(NomeRelacao,Dia,Hora).
dRtupla(Relacao,Entidade1,Entidade2,Dia,Hora).
```

4 RESTRIÇÕES DE INTEGRIDADE

Poderão ser incluidas restrições de integridade no Banco de Dados. As restrições de integridade poderão ser estáticas ou dinâmicas. Através destas poderão ser colocadas restrições a valores de tuplas, criadas dependências entre diferentes tuplas ou feitas restrições a transições de estados da base de dados. Como estas restrições serão incorporadas à base de dados, esta se torna uma base de dados semântica.

Para a manipulação das restrições foi adotada a idéia apresentada por [CAS87],[KOU79]. As restrições devem ser traduzidas em regras de inferência que, quando satisfeitas, indiquem violação da integridade da base de dados. Estas regras deverão apresentar no lado esquerdo o predicado viola, conforme mostrado abaixo:

viola :- <condição de violação > .

Ao ser solicitada alguma alteração de informações na base de dados (inclusão ou retirada de dados), o gerenciador efetua esta alteração. Em seguida, testa se houve violação de integridade ao ser feita a alteração e, caso isto tenha ocorrido, desfaz a alteração, dando uma mensagem ao usuário. O gerenciador testa a ocorrência de violação de integridade através do predicado viola.

A condição de violação deverá ser escrita em uma linguagem de lógica de 1ª ordem, cujos operadores são aqueles reconhecidos pelo PROLOG, e cujos símbolos predicativos são os que guardam informações na base de dados (classe, entidade, relacao, tupla), além de mais tres (existe, anterior e mensagem). Estes tres ultimos foram criados para auxiliar na escrita das condições de violação, devendo ser apresentados do seguinte modo:

existe(Relacao,Entidade1,Entidade2,Tempo).

anterior(Tempo1,Tempo2).

mensagem(' ..... ').

O primeiro verifica a validade de uma tupla, fornecendo seu instante de definição em Tempo. O segundo verifica se um instante é anterior a outro. Em ambos, o Tempo engloba dia e hora. Através do terceiro, pode-se informar ao usuário qual a restrição de integridade que foi violada.

Na escrita das condições de violação, os nomes de variáveis deverão ser apresentados iniciando por letra maiúscula, enquanto que os nomes que iniciam com letra minúscula correspondem a constantes.

Por exemplo, a restrição de integridade estática que define o salário de um engenheiro como sendo no mínimo NCZ$200,00 seria representada por:

```
viola :- tupla(profissao,Nome,engenheiro),
         tupla(salario,Nome,Valor),
         Valor < 200.0,
```

```
                mensagem('SALARIO DE ENGENHEIRO MENOR QUE O MINIMO').
```

Uma restrição de integridade dinâmica, por exemplo, seria não permitir diminuição de salários, representada da seguinte forma:

```
viola :- tupla(salario,Nome,V1), existe(salario,Nome,V1,T1),
         tupla(salario,Nome,V2), existe(salario,Nome,V2,T2),
         anterior(T1,T2),
         V2 < V1,
         mensagem('SALARIO NAO PODE DIMINUIR').
```

5. UTILIZACAO DO AMBIENTE

O ambiente permite a interação de um usário com a base de dados através de um sistema de menus, ou através de um interface de linguagem natural, ou ainda, através de um interpretador de programas de aplicação. A interface de linguagem natural controla um diálogo entre o usuário e o sistema através de uma linguagem próxima à natural. As regras gramaticais da linguagem que será utilizada no diálogo deverão ser fornecidas pelo próprio usuário, na forma de uma Gramática de Cláusulas Definidas (G.C.D.) [STE86]. O interpretador de programas de aplicação executa instruções escritas em uma linguagem com uma sintaxe elementar de programação em lógica, dotada de predicados operativos pré-definidos. Em qualquer uma destas tres modalidades podem ser definidas ou removidas informações na base de dados, além da possibilidade de efetuar diversos tipos de consultas. Informações detalhadas da arquitetura do sistema que gerencia este ambiente e do modo de utilização podem ser encontradas em [EDE88].

6. CONCLUSÕES

O sistema gerenciador do Ambiente para Desenvolvimento de Bancos de Dados Dedutivos Temporais descrito neste artigo está totalmente implementado, sendo uma extensão daquele descrito em [EDE88], onde o tempo não era considerado. Mostrou-se bastante eficiente em tempo de execução, podendo ser muito útil em um processo de prototipação.

AGRADECIMENTO

Queremos agradecer a J.M.V.Castilho pelas sujestões fornecidas para o tratamento de tempo no sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

[CAS87] CASTILHO,J.M.V. & LEMOS,A.S. A Construção de Protótipos Básicos de Sistemas de Bancos de Dados : Uma Proposta. Pôrto Alegre, CPGCC/UFRGS,1987 - RPn°78.

[EDE88] EDELWEISS,N. & COSTA,A.C.R. Um Ambiente para Desenvolvimento de Protótipos de Bancos de Dados Dedutivos. Pôrto Alegre, CPGCC/UFRGS, 1988 - RP nº 97.

[KOW79] KOWALSKI,R. Logic for Problem Solving. New York, North-Holland, 1979.

[KOW86] KOWALSKI,R. & SERGOT,M. A Logic-Based Calculus of Events. New Generation Computing, 4(1):pags. 214 a 254, Toquio, Ohmsha, Ltda, 1986.

[STE86] STERLING,L & SHAPIRO,E. The Art of Prolog - Advanced Programming Thechniques. Harvard, The MIT Press, 1986.

[TAR54] TARSKI,A. Introduction to Logic and to the Methodology of Sciences. Oxford, Oxford University Press, 1954.

ANEXO 2

INTEGRATION OF TWO OIS SPECIFICATION METHODS

NINA EDELWEISS - JOSÉ PALAZZO M. DE OLIVEIRA

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação
Caixa Postal 1501
90001 - Porto Alegre - RS
BRAZIL
nina@sbu.ufrgs.anrs.br or palazzo@sbu.ufrgs.anrs.br

ABSTRACT

Two OIS specification methods are here presented. The first, T.A.& A. [HOP88], is used during the requirement analysis, resulting in the system's semi-formal specification. The second, YPY[1] [EDE89], is a temporal deductive database development environment. The system's formal specification is based on the data model implemented by the YPY environment, which is used also for the system's prototype construction. The office static aspects are represented by YPY's data model. The dynamic aspects are represented through integrity violation conditions, expressed as first-order logic rules. At last, both methods are used for specifying and later prototyping a real OIS, a video rental shop.

Keywords: Office Information Systems, logic programming, database scheme, formal specifications.

1. INTRODUCTION

An office is an information processing functional unit [NEW80]. There can be found a large number of elements (persons, documents), interacting in usually well defined ways. Communication among persons can be held verbally or through message and documents exchange. Knowledge is shared by all office elements. Activities are concurrent and asynchronous. Office automation has the aim of helping people's work in this environment, through the use of computing tools. These tools can help procedural work or give support to decision processes. An Office Information System (OIS) is the set of all these tools supporting user's work.

---

1 - YPY, in the brasilian's indian language TUPI, means "The Beginning".

After [ADI86], an ideal OIS must be constructed on base of the following components: (1) a database, where the document informations are stored, (2) a comunication channel, like a local network, (3) a set of activities, defining the office workers functions, and (4) a set of users, who know how to use the system. The methods used for OIS specification must consider the particular aspects of these environments. The models used to represent the informations must be semantically rich, to be able to represent the various actions that can be executed on the informations [BRA84], and the various forms these informations may have, as voice, image, text and multimedia documents. A special emphasis shall be given to the dynamic aspects description, which define the valid sequence of activities and the consequent flow of information.

This paper presents two methods which are being developed by our research group. The first, T.A.& A. (Tasks, Activities and Actions) [HOF88] serves as a support for the requirement analysis development phase, giving rise to the OIS's semi-formal specification. The second one, YPY [EDE89], is a temporal deductive database development environment, which can be used for the OIS's formal specification. The resulting specification model is database oriented [NEV80]. The YPY environment is also used to prototype the specification. In section 2, some aspects of OIS specifications are presented. Section 3 presents the T.A.& A. method and section 4, the principal features of the YPY environment. Setion 5 shows how to construct the semi-formal specification of a video rental shop, and section 6 the formal one. Some remarks and conclusions are showed in section 7.

2. SPECIFICATION OF OFFICE INFORMATION SYSTEMS

There are several phases in an OIS implementation. It begins with the requirement collection, assembled by the users and the system's analysts jointly. These requirement analysis usually gives origin to an informal or semi-formal system specification, where the executed tasks, the manipulated objects and the possible transactions on these objects are identified. The T.A.& A. method, presented in section 3, is used for this requirement analysis.

The next phase is the formal specification, constructed on basis of the former informal or semi-formal one. An OIS formal specification must consider the human influence allways present in these systems, showing not only the information structures but also the environment semantics. Concurrency and time control shall also be represented. The formal specification must be implementation independent, not to be influenced by technological advances [ORI82]. Problem-solving environments centered on knowledge bases containing objects, tasks and development strategies are

proper tools for storing OIS specifications [GID84]. The formal specification result is an office conceptual schema: a conceptual model describing the office static and dynamic entities, and an associated formal language. The static entities describe the office structural aspects, such as documents, messages and agents. The dynamic entities describe the static entities evolution in the modelled environment, which gives the information flow defined by the valid sequence of activities to be executed [NEI88, PER88].

The validation of OIS is usually done through execution simulations, by a rapid prototyping. The prototype is used also for system requirements checking and for interfaces definition. The last phase of the OIS implementation, the physical architecture project, is based on the validated specification.

Several specification methods are used in OIS. They present different specification languages, data models, modelling concepts. The prototype construction need is always recognized. Some examples are IRIS [FIS87], ORION [BAN87], C-TODOS [PER88], O2 [LEC88, BAN88], ENCORE [HUR87], MINOS [CHR86]. This paper presents, in section 4, a formal specification method based on the data model implemented by the YPY environment, which is used also for the system's prototype construction.

The methods presented in this paper where developed within a project that aims to support all the OIS development phases, in an integrated manner.

3. THE T.A.& A. METHOD

The T.A.& A. method (Tasks, Activities and Actions) [HOP88] aims to support requirement analysis in OIS. It unfolds the office analysis in three basic hierarchical levels: the tasks, the activities and the actions (figure 1). Office entities like documents, books, messages and texts are all faced as objects. Actions are elementary accesses executed on these objects. There can be two types of actions: automated and human actions. Automated actions represent accesses to the database on which the modelled system is based. They can be executed by automated software functions. Human actions, in which human participation is fundamental, can only be aided by support software functions. Automated actions are primitives of the T.A.& A. method, and can be of two types: (1) actions executed on object classes, like generate a classe, give a class a name, alter a classe's attributes; and (2) actions executed on instances of classes, like generate an instance, change an instance's values, delete an instance, refference an instance, and get an instance's value.

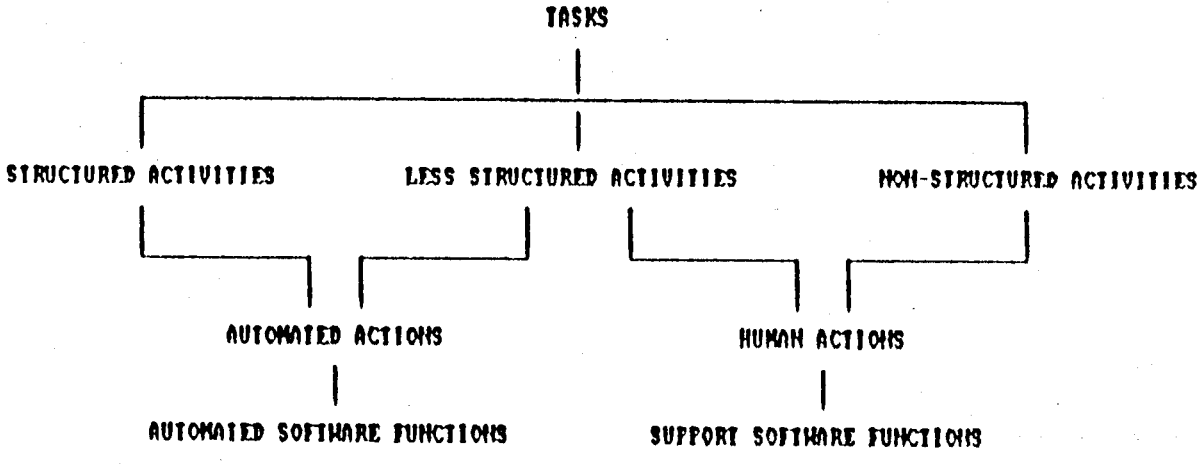


Figure 1 - Hierarchical composition of tasks, activities and actions.

An activity represents a user's transaction. It's composed of (1) an ordered set of actions, and (2) a pre and post-execution conditions set. These conditions model the activities execution sequence in a considered task. There can be concurrent activities in a same task. OIS activities can be classified with respect to the amount of human influence they present, corresponding to their automation possibility: (1) structured activities can be totally automated; (2) less structured activities are those which present necessarily parts of human work, like decision making; and (3) non-structured actions, which cannot be executed by computers. The last two activities types can be supported by computer software functions, like the ones that allow database accesses or simulation and planning tools usage.

The tasks are work portions with a specific meaning in the office. A task is composed of an ordered set of sub-tasks or activities. That means that more than one level of tasks may exist. The modelling of a sub-task must define (1) the decomposition of this one in other sub-tasks or activities, and (2) the set of pre and post-execution conditions for this sub-task.

The T.A.S A. method shall be applied according to the following sequence of steps: (1) tasks identification; (2) object identification; (3) refinement of each task in a set of sub-tasks or activities, simultaneously defining the pre and post-execution conditions for each sub-task and activity; this step shall be repeated until all the sub-tasks are decomposed in activities; (4) refinement of each activity in an ordered set of actions.

4. THE YPY ENVIRONMENT

A system's formal specification can be done defining a database and using its data model [LYN86]. The static modelling constructions, like objects, classes of objects, and relationships are used to define the system's semantic domain. The dinamic modelling constructs, like database operations, are used to define the operations semantic. These concepts are used to specify an OIS using YPY [EDE89], a temporal deductive database development environment. It can be used in PC-compatible computers and is implemented in PROLOG [STE86].

In OIS the time representation associated to the informations is necessary to represent, for instance, event duration, calender, previsions, document and operation life time, and activities' temporal restrictions. The YPY environment implements a temporal database, having associated to each stored information the moment (date and time) of it's definition. When an information is deleted a new information is stored, defining the moment (date and time) of the deletion of this specific information. No information is ever removed from the database, which holds all the past stored.

The YPY's data model is based on the classes and relations logic [TAR54]. Figure 2 shows the data model set of classes. They are classified in primary and secondary classes. There are two kinds of primary classes, depending on the type of elements: (1) the symbolic class, which holds all the PROLOG recognized symbols, and (2) the numerical classes, being implemented the real and the integer types. The secondary classes can be (1) basic, made of elements of the primary classes, or (2) derived, obtained from operations on other classes already defined by the user, like union, intersection and complement. The basic secondary classes must have a type identification, defining their entities original primary class. If this type is symbolic, the classes entities must be defined by the user, one by one, on a extentionally way. If the classes type is integer or real, the entities are already intentionally defined.

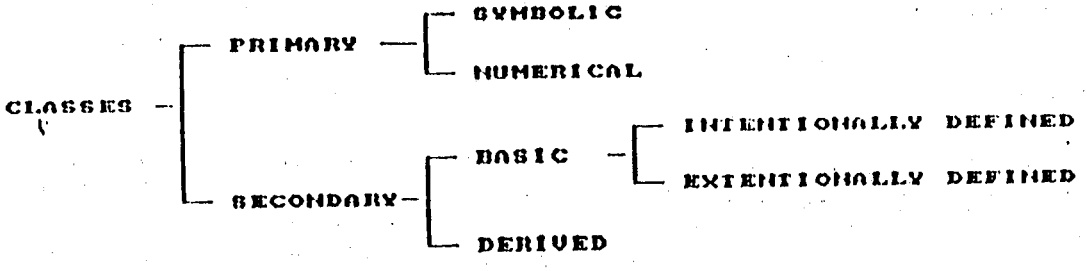


Figure 2 - Classes of YPY's Data Model.

The YPY's data model set of relations is showed in figure 3. All the relations are binary. There are two kinds of relations: (1) basic relations, defined between two classes; and (2) derived relations, obtained by operations over relations previously defined by the user, like union, intersection, complement and composition. The basic relations can be total, in which all possible tuples are defined, or partial, for which the user must define the valid tuples later.

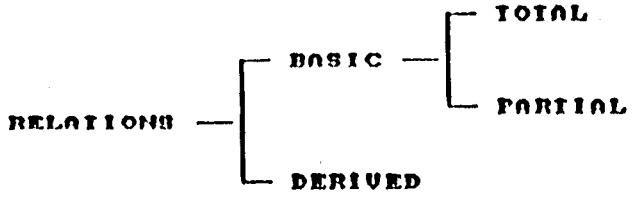


Figure 3 - Relations of YPY's Data Model.

YPY's data model is, in many aspects, similar to that of TAXIS [MIL80], which is an entity oriented data model [MIX87]. As in TAXIS, entities can be generated, deleted and have a unique identity, that means, an entity is different from any other past, present or future entity. In the YPY data model, this also holds for classes, relations and tuples. Only binary relations are possible, like in TAXIS. When an object has a greater number of properties (n), they shall be represented by various binary relations (n+1). As an example, suppose the object client has two properties, name and address. These properties are represented by three relations, associated by a new property common to all three, for instance, code. These relations are relation(code,client), relation(code,name), and relation(code,address). Client, name, address and code are defined as classes.

Using YPY's data model, the system's static properties are defined through classes, entity defined in this classes, binary relations between classes, tuples defined in this relations, and a set of database integrity violation conditions, which restrict the entity and tuple definition possibilities. The dynamic properties are also modelled using integrity violation conditions. The database integrity violation conditions are checked each time an action is executed. If any condition fulfils, an integrity condition was violated, and the last executed action is invalidated. The environment provides for the restoration of the database conditions previous to that actions execution. When compared to methods that model dynamic properties through Petri Nets [RIC87], this can be considered as a complementary approach: while in the nets the valid system's state transitions are represented, YPY's integrity violation conditions represent

the invalid states reached by system's invalid transitions.

The integrity violation conditions are written as inference rules, and are stored in the same database. The rules are written in a first-order predicate logic language, which accepts the YPY data model representations, pre-defined predicates of the YPY environment, and PROLOG operators. There are pre-defined predicates that (1) help to identify which of the rules was violated, giving the user specific messages, (2) verify if an information is defined , and (3) manage time informations. Examples of violation rules:

```
violate :- entity(client_name,X), defined(client_name,X),
          entity(bad_client,X), defined(bad_client,X),
          message('THIS NAME IS IN THE BAD CLIENTS FILE').

violate :- tuple(rented,Tape1,Client),
          exist(rented,Tape1,Client,Time1),
          tuple(rented,Tape2,Client),
          exist(rented,Tape2,Client,Time2),
          before(Time2,Time1), gap(Time2,Time1,Days), Days > 30,
          message('THIS CLIENT HAS ANOTHER TAPE MORE THAN 30 DAYS').
```

Client_name and bad_client are class names, entity(Class_name, Entity_name) is the data model representation for all defined entities; rented is a relation name, Tuple(Relation_name,First_class_entity, Second_class_entity) is the data model representation for all tuples; defined, message, exist, before, gap and message are pre-defined predicates.

The concurrency modelling can also be made through the YPY environment. More than one valid and mutually exclusive transactions are possible in a same moment. Actions from different activities may be executed alternatively. If it is necessary, the user can determine an action execution order through proper integrity violation rules.

The YPY environment can be used (1) through a menu system, which gives offerspossibility a set of tools for defining or removing classes, entities, relations and tuples, and listing various forms of informations; (2) through a specific language, recognized by the environment; and (3) through a more natural language, which grammar the user must give to the environment in the Defined Clausal Grammar (D.C.G.) form [STE86].

5. SEMI-FORMAL SPECIFICATION OF A VIDEO RENTAL SHOP

To illustrate the integrated usage of the presented methods, we will show how to construct the specification of a real office, a video rental

shop. We begin with the requirement analysis, using the T.A.& A. method, supposing the requirement collection already done. Only a subset of all the possible tasks in such an office will be considered: clients registration, tape registration and the renting of a tape. Not all aspects of these tasks will be analysed, just the necessary ones to give an idea of the specification method. Other important tasks in this office are employers and financial control, and publicity routines.

The manipulated objects identified in the video rental shop are (1) tape, with properties code, film name and film type, (2) catalog, formed of objects tape, (3) client, with properties code, name and address, (4) good_client, a file of admited clients, formed of objects client, (5) bad_client, a file holding names of undesired clients, and (6) rentals, a file holding informations over all rented tapes.

Each one of the tasks is now refined, originating sub-tasks and activities. The client's registration task is made in two separate files. The first file holds informations concerning the admitted agency's clients, who can rent tapes. The second hods the names of the undesired clients, to whom the tape renting is not admitted. Those lists are controlled by two mutually exclusive sub-tasks. The sub-task new client registration first examines if this new client's name is not yet in the bad clients list. This would block his registration as an admitted client. If this fact does not occur, the client's name and aditional informations (code, address) are stored in the good clients list.

Activities:

a1. get a value for this client's code;
action: get value;
post-condition: there is no such a value as client's code in anyone of the clients files;

a2. store clients informations in the good clients file;
pre-condition: the name and the code of this client are still not in this file;
action: generate an instance of good_client, with this code, name and address;

The sub-task bad client's registration initially searches this client's name in the good clients file, removing all his informations if found. Only then is this name added to the bad clients file.

Activities:

a3. remove this client's informations of the good clients file;
pre-condition: this client's name is in the good clients file;
action: remove the corresponding instance of good_client;

a4. store this clients name in the bad clients file;
pre-condition: both files of clients do not hold this name;
action: generate an instance of bad_client, with this name.

The tape registration task controls a catalog of all disposable tapes. For each tape the catalog contains a unic code, the film's name and type (adventure, terror, war, musical, etc).

Activities:
a5. get a new code for the tape;
action: get a value;
post-condition: the value does not correspond to any other tape;
a6. store informations of this tape in the tape catalog;
pre-condition: this tape's code does not appear in the tape file;
action: generate an instance of catalog with this tape's code, film name and type.

The task that controls tape rentals is executed by two mutually exclusive sub-tasks: one for the beginning of a rental and the other for it's end. The beginning of a rental sub-task must consider a series of factors, according to the stored informations semantics. A client can rent more than one tape, in the same or in different days. When a tape is rented, the tape's and the client's codes must be stored, toghether with the day of the beginning of the rental as well as other informations not used here.

Activity:
a7. store renting informations in the rented tapes file.
pre-condition: client's code must be in the good clients file and the wanted tape code must be in the tape catalog and this client has no tape more than 30 days and this tape code does not appear in the actually rented tapes file;
action: generate an instance of rentals with the tape's and the client's codes, and the actual date.

The devolution of a tape sub-task defines the end of a rental. It is executed removing the correspondent informations from the rented tapes file. In this example we are not considering the financial control of this agency.

Activity:
a8. remove renting informations from the rented tapes file;
pre-condition: existence of a location with this tape's code and this client's code and existence of the actual date;
action: remove the corresponding instance of rentals.

6. FORMAL SPECIFICATION AND PROTOTYPING OF THE VIDEO RENTAL SHOP

The semi-formally specified tasks, activities and actions are now formally specified using the TPT environment data model. First of all we must identify the primitive automated actions. The actions on object classes in this data model are (1) a basic class generation, associating a type (integer, real or symbolic) to all the elements of that class; (2) a derived class generation; (3) the removal of a defined class, implying in the automatic removal of all entities defined in that class and corresponding derived classes; (4) the definition of a basic relation between two classes, with the simultaneous definition of the type of that relation (total or parcial); (5) a derived relation definition; (6) a relation removal, with the simultaneous removal of all its tuples and derived relations; and (7) references to classes and to relations. The actions executed on instances of classes are (1) entities definition in a class; (2) entities removals from a class; (3) tuple definitions in a relation; (4) tuple removals from a relation; and (5) accesses to entities and to tuples.

The defined classes and their associated types, for this examples are:

- film_code (integer) : film's codes;
- film_name (symbolic) : names of all films;
- film_type (symbolic) : existent film types;
- client_code (integer) : client's codes;
- client_name (symbolic) : names of good clients;
- bad_client (symbolic) : names of bad clients;
- addr_str (symbolic) : names of streets;
- addr_numbr (integer) : corresponding address numbers.

The defined relations, together with the two related classes, are:

- tape (tape_code, film_name);
- tape_type (tape_code, film_type);
- client (client_code, client_name);
- street (client_code, addr_str);
- number (client_code, addr_numr);
- flat (client_code, addr_numbr);
- rented (tape_code, client_code).

To represent the actions of the preceding section, like generate an instance of good_client, more than one tuple must be defined, using the proper relations - client, street, number and flat. The environment provides commands for the generation and delection of entities and tuples. As all the defined informations have associated the actual date information, it is not necessary to generate explicit time informations.

The integrity control in this environment is made after an action is executed. It is, therefore necessary the representation of all possible invalid states reached by the modelled system supposing the pre-conditions be not satisfyed. Each invalid state is represented by a integrity violation rule. This rules are also used to check the post-conditions. The two rules presented in section 4 correspond to this example: the first garantees that a client's name is not in the black list, in which case it accuses a violation and the second one prevents that a client rents a new tape if he already has one rented for more than 30 days.

7. CONCLUSIONS

The specification is one of the most important activities in an OIS development. The prototyping phase and the final physical project are based on this specification. Therefore, the choice of an adequate formalism is fundamental. This formalism shall be able to represent the static and the dynamic characteristics of the system, and shall possess facilities for the specification's validation.

In this paper we presented two specification methods, being developed in our research group: (1) the T.A.& A. method, that can be used in the requirement analysis, resulting in a semi-formal specification; and (2) the YPY environment, used for the formal specification and prototyping phase. The YPY's features here presented are all implemented. The specification's formalizing process using YPY's data model is a natural consequence of the semi-formal specification obtained by the T.A.& A. method.

These methods are part of a project that aims to reach a complete environment for OIS specification and implementation. This environment shall possess tools to support all the development phases, interacting in an integrated way.

BIBLIOGRAPHICAL REFERENCES

[ADIB86] ADIBA,M. Problématique de bases de données multi-media. In: Nouvelles perspectives des bases de données. Adiba et al. Paris, Ed. Eyrolles, 1986. p 11-17.

[BANE87] BANERJEE,J. et. al. Data model issues for object-oriented applications. ACM Transactions on Office Information Systems, New York, 5(1):3-26, Jan. 1987.

[BANC88] BANCILHON,F. et al. The Design and Implementation of O_2, an object-oriented database system. Advances in object-oriented database systems. K.R.Richter (ed), Berlin, Springer-Verlag, 1988. p 1-22.

[BRAC84] BRACCHI,G. & PERNICI,B. The Requirements of office systems. ACM Transactions on OIS, 2(2):151-170, Apr. 1984.

[CHR86] CHRISTODOULAKIS,S. et al. Multimedia document presentation, information extraction, and document formation in MINOS: a model and a system. ACM Transactions on Office Information Systems, New York, 4(4):345-83, Oct. 1986.

[EDE89] EDELWEISS,N. Ambiente para desenvolvimento de bancos de dados dedutivos temporais. SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 4º, Anais. Campinas, São Paulo, abr. 1989, p. 163-173.

[FIS87] FISHMAN,D.H. et al. IRIS: an object-oriented database management system. ACM Transactions on Office Information Systems, New York, 5(1):48-69, Jan. 1987.

[GID84] GIDDINGS,R.V. Accommodating uncertainty in software design. Communications of the ACM, New York, 27(5), May 1984.

[GRI82] GRIETHUYSEN,J.J. et al. Concepts and terminology for the conceptual schema. ISO/TC97/SC5/WG3 Report, 1982.

[HEI88] HEIJINK,F. et al. Development of tools for designing OIS. Information technology for organizational systems. U.-J. Bullinger et al. (Ed.), p 66-73. Elsevier Science Publishers B.V., North-Holland. Brussels, 1988.

[HOP88] HOPPEN,N.; OLIVEIRA,J.P.M.; LIMA,L.M.R. Metodologia para modelagem de escritório baseada no nível de estruturação das atividades. In: REUNIÃO ANUAL DA ASSOCIAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO, 12º, Natal, 26-28 set. 1988. Anais. Natal, 1988. p.247-263.

[HOR87] HORNICK,R.K. & ZDONIK,B.Z. A Shared, segmented memory system for object-oriented database. ACM Transactions on Office Information Systems, New York, 5(1):79-95, Jan.1987.

[LEC88] LECLUSE,C.; RICHARD,P.; VELEZ,F. O_2, an object-oriented data model. ACM SIGMOD RECORD, New York, 17(3):424-33, Sep. 1988.

[LYN86] LYNGBAEK,P. & KENT,W. A Data modeling methodology for the design and implementation of information systems. In: INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED DATABASE SYSTEMS, Pacific Grove, California, 23-26 Sep. 1986. Proceedings. K.Dittrich & U.Dayal (ed.), 1986. p 6-17.

[MIL80] MYLOPOULOS,J.; BERNSTEIN,P.A.; WONG,H.K.T. A Language facility for designing database-intensive applications. ACM Transactions on Database Systems, 5(2):185-207, Jun. 1980.

[NEU80] NEUMAN,U. Office models and office systems design. Integrated office systems - burotics, N.Naffath (ed.), North-Holland Publishing Company, Amsterdam, 1980. p 3-10.

[NIX87] NIXON,B. et al. Implementation of a compiler for a semantic data model: experiences with TAXIS. In: ACM ANNUAL CONFERENCE ON MANAGEMENT OF DATA, San Francisco, 27-29 May, 1987. SIGMOD RECORD, 16(3):118-31, Dec. 1987.

[PER88] PERNICI,B. et al. C-TODOS : An Automatic tool for system conceptual design. Politecnico di Milano, Milão, Itália. Oct. 1988.

[RIC87] RICHTER,G. et al. Generic office frame of reference GOFOR. GMD, St.Augustin, F.R. Germany. (Esprit Project 56: Functional analysis of office requirements). 1987.

[STE86] STERLING,L & SHAPIRO,E. The Art of Prolog - advanced programming thechniques. Harvard, The MIT Press, 1986.

[TAR54] TARSKI,A. Introduction to logic and to the methodology of deductive sciences. Oxford, Oxford University Press, 1954.

ANEXO 3

ESPECIFICAÇÃO FORMAL DO MÉTODO T.A.& A.

NINA EDELWEISS
JOSÉ PALAZZO M. DE OLIVEIRA

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 1501
90001 - Porto Alegre - RS
Fax: +55(512)24.4164 E-mail: nina@sbu.ufrgs.anrs.br

SUMÁRIO

A metodologia T.A.& A. se destina à coleta, organização e análise dos requisitos de um Sistema de Informação de caráter sócio-técnico. Este trabalho apresenta a especificação formal de um de seus métodos, o método T.A.& A. Inicialmente são apresentados os conceitos fundamentais do mesmo, sendo definida uma linguagem para a sua utilização. Em seguida, o método é especificado formalmente, através da utilização do formalismo Z.

ABSTRACT

The T.A.& A. methodology aims to support requirement collection, organization and analysis of a socio-technical Information Systems. This work presents the formal specification of one of its methods, the T.A.& A. method. The fundamental conceipts of the T.A.& A. method are presented and a language to be used in it's applications is defined. Then the method is formally specified. The Z formalism is used in this specification.

1. INTRODUÇÃO

Neste trabalho é apresentada a especificação formal do método T.A.& A. A realização desta especificação foi motivada pela necessidade de avaliar a precisão dos conceitos envolvidos na definição do método, uma vez que tal formalização só é realizável quando estes conceitos estão completamente definidos, de forma clara e precisa. Exemplos de especificação formal de ferramentas de software (linguagens, modelos de dados, métodos de especificação) podem ser encontrados em [MAR89, MOT89, MOT90]. Através da realização desta especificação foi possível completar a definição do método, no que se refere à linguagem a ser utilizada quando de sua

utilização e à seqüência de passos a serem seguidos em sua aplicação.

A especificação formal de um método pode ser utilizada para verificar a completeza e a coerência de uma especificação que estiver sendo construída através da utilização deste mesmo método. A especificação do método deve ser traduzida em regras de inferência e armazenada em uma base de conhecimentos. Os fatos que compõem a especificação devem ser armazenados nesta mesma base de conhecimentos, à medida em que a especificação for sendo construída. A cada novo fato incluído é feita a verificação das regras de definição do método, dando assim origem a uma especificação formalmente verificada. Como exemplos de sistemas que permitem o desenvolvimento de produtos formalmente verificados e que utilizam princípios análogos a este podemos citar TODOS [PER90], OSIRIS [MAI87] e PENELOPE [RAM89].

O método especificado faz parte da metodologia T.A.& A. (Tarefas, Atividades e Ações) [HOP88, OLI89a,b], destinada à coleta, organização e análise dos requisitos de Sistemas de Informação de caráter sócio-técnicos. Sua utilização tem por objetivo dar origem a uma especificação que represente, com a maior fidelidade possível, a realidade destes ambientes, dando ênfase às tarefas neles executadas. Está sendo desenvolvida através de um projeto que envolve o Curso de Pós-Graduação em Ciência da Computação (CPGCC) e o Programa de Pós-Graduação em Administração (PPGA), ambos da UFRGS. O método T.A.& A., integrante desta metodologia, é utilizado na organização dos requisitos, resultando em uma especificação semiformal do sistema.

A Seção 2 deste trabalho apresenta a especificação informal do método T.A.& A. A definição dos passos que devem ser seguidos na sua utilização e de uma linguagem apropriada são vistos na Seção 3. A Seção 4 apresenta a especificação formal do método, através do formalismo Z. Considerações finais constituem a Seção 5.

2. MÉTODO T.A.& A.

Os elementos básicos de um Sistema de Informação de caráter sócio-técnico são os agentes (profissionais que atuam neste ambiente), os dados (informações armazenadas) e o trabalho

desenvolvido [BAR85]. O modelo de um sistema desta natureza deve identificar as tarefas que são executadas, os objetos manipulados e quais as transações possíveis sobre estes objetos. Usualmente a descrição destes aspectos é feita em linguagem natural, cujas características, inerentemente ambíguas, dão origem a documentos ambíguos, inconsistentes e incompletos. O emprego de um formalismo visa a obter descrições mais completas da realidade. O formalismo empregado não deve ser muito complexo, uma vez que esta especificação é dirigida ao usuário, isto é, deve ser apresentada em uma forma facilmente compreendida por pessoas não especialistas em computação, para fins de validação.

Na modelagem é utilizado o paradigma de orientação a objetos [ATK89], já utilizado em outros sistemas destinados ao mesmo tipo de aplicação, tais como OTM [LOC88], IRIS [FIS87], ORION [BAN87], EXODUS [CAR88], O2 [LEC88] e TODOS [PER90]. As entidades do sistema, tanto as concretas como as abstratas, são encaradas como objetos modelados em um banco de dados. As propriedades estáticas do sistema são representadas pelos objetos e as dinâmicas, pelas possíveis transações sobre estes objetos.

Com base nos objetos identificados e analisando as atividades que deverão ser efetuadas com os mesmos sob ponto de vista de apoio informatizado para a execução das tarefas, o método decompõe o trabalho do escritório em três níveis hierárquicos básicos, representando cada um deles um nível específico de descrição da realidade: o nível das tarefas, o das atividades e o das ações. Uma tarefa corresponde a uma aplicação, uma atividade é uma interação entre o usuário e a aplicação, e uma ação constitui uma modificação elementar sobre os objetos. As ações são primitivas do método, sendo aplicáveis a quaisquer objetos. O nível das tarefas pode ser subdividido em múltiplos subníveis, representando cada subtarefa uma parcela independente de trabalho.

Toda tarefa, subtarefa e atividade apresenta um conjunto de pré-condições e um de pós-condições de execução. As pré-condições representam retrições de integridade dinâmica do sistema, modelando as características fundamentais que o sistema deve apresentar para que esta tarefa/subtarefa/atividade possa ser executada (condições de habilitação). As pós-condições representam o estado que o sistema

deve apresentar após a execução desta parcela de trabalho. Esta forma de representação da ordem de execução permite a especificação de processos concorrentes em sistemas abertos, característica importante em ambientes sócio-técnicos e que nem sempre é levada em consideração nos sistemas existentes. Atualmente a modelagem de sistemas abertos tem recebido muita atenção, como demonstram os sistemas UBIK [YON90] e GUTENBERG [CHR90]. O enfoque utilizado no método T.A.& A. é semelhante ao do sistema AMS [TUE88].

A maioria dos sistemas de análise de Sistemas de Informação, como por exemplo o sistema TODOS [PER90], é orientada somente para a definição de problemas estruturados. Alguns destes sistemas permitem a modelagem de parcelas de trabalho não estruturado junto ao estruturado, como POISE [CRO84] e STRATEGIC APPROACH [PAN84]. A característica principal do método T.A.& A., característica esta que o torna uma importante ferramenta para a análise desta classe de ambientes, consiste na possibilidade de integrar, em um mesmo processo de análise e de modelagem, o trabalho estruturado com o não estruturado. As atividades que apresentam parcelas de trabalho não estruturado são modeladas através dos seus interfaces com o sistema, (definidos por suas pré e pós-condições). A definição de execução da atividade fica indefinida, representando trabalho humano.

3. CARACTERÍSTICAS DO MÉTODO

Para a utilização do método T.A.& A. foi definida a seguinte seqüência de passos: (1) identificação das tarefas que se quer especificar; (2) identificação dos objetos que serão manipulados na execução destas tarefas; (3) detalhamento das tarefas em conjuntos ordenados de subtarefas e/ou de atividades, com a definição das pré e pós-condições de cada subtarefa e atividade; este passo deverá ser repetido até que todas estejam desdobradas em atividades; (4) detalhamento de cada atividade em um conjunto ordenado de ações; as atividades para as quais isto não for possível ou desejável são classificadas como "indefinidas", caracterizando uma parcela de trabalho humano.

Foi definida uma linguagem a ser utilizada na representação do modelo, cuja BNF se encontra no Anexo 1 deste trabalho. Na definição desta linguagem foi levada em consideração a necessidade de

validação da especificação por pessoas não fortemente treinadas em computação. Foram escolhidas construções simples, de tal modo que a especificação resultante tivesse forma semelhante a um texto em português. Na apresentação das pré e pós-condições é permitida a opção entre frases em linguagem natural ou sentenças de lógica de primeira ordem, mais exatas, porém mais complexas.

A definição de uma classe de objetos é feita definindo sua estrutura e restrições que delimitam as instâncias válidas na classe. Exemplos de definição de classes de objetos:

```
Objeto FUNCIONÁRIO
   atributos NOME, PROFISSÃO, SALÁRIO, CARGO
Objeto ENGENHEIRO
   subclasse de FUNCIONÁRIO
   atributos ESPECIALIZAÇÃO
   restrições:
     'toda intância desta classe possui PROFISSÃO = ENGENHEIRO'
```

As tarefas, subtarefas e atividades são definidas através de suas pré e pós-condições e de um conjunto ordenado de subtarefas e/ou atividades que caracterizam a sua execução. Seu nome deve iniciar por um verbo na forma infinitiva, seguindo o nome de um objeto, podendo ser complementado por qualquer seqüência de palavras. Uma atividade pouco estruturada é definida como 'indefinida', sendo o trabalho modelado somente através de seus interfaces com o sistema. As ações, pré-definidas na linguagem, são criar instâncias, obter valores de atributos, remover instâncias e atualizar atributos. Exemplos de definição de tarefa e atividade:

```
Tarefa contratar engenheiro
   pré-condição: 'existem os dados do engenheiro'
   execução:
      escolher cargo que vai ocupar
      definir salário
      cadastrar engenheiro
      comunicar engenheiro de seu contrato
   pós-condição:
      'existe instância de ENGENHEIRO com todos os seus dados'
```

Atividade comunicar engenheiro de seu contrato
pré-condição: 'o engenheiro foi cadastrado'
execução: 'indefinida'
pós-condição:.

4. ESPECIFICAÇÃO FORMAL DO MÉTODO T.A.&.A.

O objetivo principal deste trabalho é a especificação formal do método T.A.& A. A especificação formal de um método deve definir precisamente (1) o modelo de representação das informações utilizado pelo método, (2) a linguagem de representação este modelo, e (3) a seqüência de passos que deve ser seguida na utilização do método.

Na especificação formal do método T.A.& A. é utilizado um subconjunto do formalismo Z [ABR74, DEL82], um método de especificação orientado a modelos, baseado em conceitos matemáticos simples como lógica de primeira ordem e teoria dos conjuntos. Este formalismo foi escolhido devido ao seu alto grau de abstração, respaldado por uma notação gráfica [ALV88] ao mesmo tempo clara e concisa. Além disso, é facilmente adaptável à representação através de regras de inferência, permitindo assim sua utilização na verificação formal de especificações. O formalismo utilizado é apresentado no Anexo 2 deste trabalho. São utilizadas, ainda, as seguintes convenções:

nil - entidade indefinida
ø - conjunto vazio
card(F(x)) - função que fornece a cardinalidade de F

4.1 CONJUNTOS

Os conjuntos utilizados no método T.A.& A. são os seguintes:

OBJETO - objetos identificados no ambiente modelado;
TAREFA - tarefas definidas;
SUBTAREFA - subtarefas definidas;
ATIVIDADE - atividades definidas;
AÇÃO = ('criar','obter','remover','atualizar')
- ações pré-definidas;
VERBO - verbos da língua portuguesa, na forma infinitiva
STRING - todas as possíveis seqüências de caracteres alfanuméricos.

Restrição imposta ao conjunto VERBO:

(1) VERBO não contém as palavras reservadas para as ações:

$\forall v \in VERBO . \forall a \in AÇÃO . v \neq a$

4.2 FUNÇÕES DE DEFINIÇÃO DE OBJETOS

OBJETO < -nome-/- > STRING

OBJETO << -Atributos-/- >> OBJETO

OBJETO << -Componentes-/- >> OBJETO

OBJETO << -subclasse-/- > OBJETO

OBJETO << -conjunto-/- > OBJETO

OBJETO << -subconjunto-/- > OBJETO

OBJETO << -restrições-/- > ('verdadeiro','falso')

Condições impostas a estas funções:

(1) Todos os nomes de objetos devem ser diferentes:

$\forall o,o' \in OBJETO . nome(o) \neq nome(o')$

(2) Quando forem definidos componentes para um objeto, este não poderá ter as funções subclasse, conjunto e subconjunto definidas:

$\forall o \in OBJETO . (Componentes(o) \neq \emptyset \rightarrow (subclasse(o) = nil \wedge conjunto(o) = nil \wedge subconjunto(o) = nil))$

(3) Quando um objeto for definido como conjunto de elementos de outro objeto, não podem ser definidas as funções Componentes, subconjunto e subclasse:

$\forall o \in OBJETO . (conjunto(o) \neq nil \rightarrow (Componentes(o) = \emptyset \wedge subconjunto(o) = nil \wedge subclasse(o) = nil))$

(4) Quando um objeto for definido como subconjunto de outro objeto, não podem estar definidas as funções Componentes, conjunto e subclasse:

$\forall$ o $\in$ OBJETO . (subconjunto(o) $\neq$ nil $\Rightarrow$ (Componentes(o) = $\varnothing$
$\wedge$ conjunto(o) = nil $\wedge$ subclasse(o) = nil))

(5) Quando um objeto for definido como subclasse de outro objeto, não poderão estar definidas as funções Componentes, conjunto e subconjunto:

$\forall$ o $\in$ OBJETO . (subclasse(o) $\neq$ nil $\Rightarrow$ (Componentes(o) = $\varnothing$ $\wedge$
conjunto(o) = nil $\wedge$ subconjunto(o) = nil))

4.3 FUNÇÕES DE DEFINIÇÃO DE TAREFAS, SUBTAREFAS E ATIVIDADES

ident_tarefa
TAREFA < -/---------- > VERBO X OBJETO X STRING''

ident_subtarefa
SUBTAREFA < -/---------- > VERBO X OBJETO X STRING''

ident_atividade
ATIVIDADE < -/---------- > VERBO X OBJETO X STRING''

Execução
TAREFA U << -/---------- > (VERBO X OBJETO X STRING) U
ATIVIDADE U ('indefinida') U
SUBTAREFA (AÇÃO X STRING) U
(AÇÃO X STRING X OBJETO X STRING)

pré-condição
TAREFA U << -/---------- >('verdadeiro','falso')
ATIVIDADE U
SUBTAREFA

pós-condição
TAREFA U << -/---------- > ('verdadeiro','falso')
ATIVIDADE U
SUBTAREFA

Condições impostas a estas funções:

(1) Identificadores de tarefas, subtarefas e de atividades devem ser únicos:

$\forall$ t,t' $\in$ TAREFA . ident_tarefa(t) $\neq$ ident_tarefa(t')

$\forall$ s,s' $\in$ SUBTAREFA . ident_subtarefa(s) $\neq$ ident_subtarefa(t')

$\forall$ a,a' $\in$ ATIVIDADE . ident_atividade(a) $\neq$ ident_atividade(a')

$\forall$ t $\in$ TAREFA . $\forall$ s $\in$ SUBTAREFA .
ident_tarefa(t) $\neq$ ident_subtarefa(s)

$\forall$ t $\in$ TAREFA . $\forall$ a $\in$ ATIVIDADE .
ident_tarefa(t) $\neq$ ident_atividade(a)
$\forall$ s $\in$ SUBTAREFA . $\forall$ a $\in$ ATIVIDADE .
ident_subtarefa(s) $\neq$ ident_atividade(a)

(2) A execução das tarefas e das subtarefas é definida por um conjunto de subtarefas e de atividades:

$\forall$ t $\in$ TAREFA . Execução(t) $\neq$ 'indefinida'
$\forall$ s $\in$ SUBTAREFA . Execução(s) $\neq$ 'indefinida'
$\forall$ t $\in$ TAREFA . Execução(t) = (v,o,s) $\Rightarrow$ v $\notin$ AÇÃO
$\forall$ s $\in$ SUBTAREFA . Execução(s) = (v,o,s) $\Rightarrow$ v $\notin$ AÇÃO
$\forall$ t $\in$ TAREFA .
$\neg$ (Execução(t) = (a,b,c,d) $\vee$ Execução(t) = (a,b))
$\forall$ s $\in$ SUBTAREFA .
$\neg$ (Execução(s) = (a,b,c,d) $\vee$ Execução(s) = (a,b))

(3) A execução das atividades é definida por um conjunto de ações ou é 'indefinida':

$\forall$ a $\in$ ATIVIDADE . Execução(a) = (v,o,s) $\Rightarrow$ v $\in$ AÇÃO
$\forall$ a $\in$ ATIVIDADE . Execução(a) = (v,s,o,s') $\Rightarrow$ v $\in$ AÇÃO
$\forall$ a $\in$ ATIVIDADE . Execução(a) = (v,s) $\Rightarrow$ v $\in$ AÇÃO
$\forall$ a $\in$ ATIVIDADE .
Execução(a) = 'indefinida' $\Rightarrow$ card(Execução(a)) = 1

4.4 CONTROLE DOS PASSOS DE DESENVOLVIMENTO DA ESPECIFICAÇÃO

As seguintes condições se relacionam com os passos sucessivos de desenvolvimento de uma especificação:

(1) Todos os nomes de objetos devem ser definidos antes de ser referenciados:

$\forall$ t $\in$ TAREFA . ident_tarefa(t) = (v,o,s) $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)
$\forall$ s $\in$ SUBTAREFA . ident_subtarefa(s) = (v,o,r) $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)
$\forall$ a $\in$ ATIVIDADE . ident_atividade(a) = (v,o,s) $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)
$\forall$ t $\in$ TAREFA . Execução(t) = (v,o,s)
($\exists$ o' $\in$ OBJETO . nome(o') = o)
$\forall$ s $\in$ SUBTAREFA . Execução(s) = (v,o,r) $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)

$\forall$ a $\in$ ATIVIDADE . Execução(a) = (v,o,s) $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)
$\forall$ a $\in$ ATIVIDADE . Execução(a) = (v,s,o,s') $\Rightarrow$
($\exists$ o' $\in$ OBJETO . nome(o') = o)

(2) Para definir uma subtarefa, ela deve ter sido referenciada anteriormente como integrante da Execução de uma tarefa ou de outra subtarefa:

$\forall$ s $\in$ SUBTAREFA . ident_subtarefa(s) = (v,o,r) $\Rightarrow$
(($\exists$ t $\in$ TAREFA . (v,o,r) $\in$ Execução(t))
$\vee$ ($\exists$ s' $\in$ SUBTAREFA . (v,o,r) $\in$ Execução(s')))

(3) Para definir uma atividade, ela deve ter sido referenciada anteriormente como integrante da Execução de uma tarefa ou de uma subtarefa:

$\forall$ a $\in$ ATIVIDADE .ident_atividade(a) = (v,o,r) $\Rightarrow$
(($\exists$ t $\in$ TAREFA . (v,o,r) $\in$ Execução(t))
$\vee$ ($\exists$ s $\in$ SUBTAREFA . (v,o,r) $\in$ Execução(s)))

4.5 CONDIÇÕES FINAIS DA ESPECIFICAÇÃO

Ao final da especificação deverão ser verificadas as seguintes condições (testes de completeza da especificação):

(1) Todo objeto definido através de um nome deve ter pelo menos uma das funções entre classes de objetos definida:

$\forall$ o $\in$ OBJETO . nome(o) $\neq$ nil $\Rightarrow$ (Atributos(o) $\neq \emptyset \vee$
Componentes(o) $\neq \emptyset \vee$ subclasse(o) $\neq$ nil $\vee$
conjunto(o) $\neq$ nil $\vee$ subconjunto(o) $\neq$ nil)

(2) Todas as subtarefas e atividades referenciadas deverão ter sido definidas:

$\forall$ t $\in$ TAREFA . Execução(t) = (v,o,r) $\Rightarrow$
(($\exists$ s $\in$ SUBTAREFA . ident_subtarefa(s) = (v,o,r))
$\vee$ ($\exists$ a $\in$ ATIVIDADE . ident_atividade(a) = (v,o,r)))
$\forall$ s $\in$ SUBTAREFA . Execução(s) = (v,o,r) $\Rightarrow$
(($\exists$ s' $\in$ SUBTAREFA . ident_subtarefa(s') = (v,o,r))
$\vee$ ($\exists$ a $\in$ ATIVIDADE . ident_atividade(a) = (v,o,r)))

(3) Todas as tarefas deverão ter sua execução e suas pré-condições definidas:

$\forall$ t $\in$ TAREFA . (Execução(t) $\neq \emptyset \wedge$ pré_condição(t) = nil)

(4) Todas as subtarefas e atividades deverão ter sua execução, pré e

pós-condições definidas:

$\forall$ s $\in$ SUBTAREFA . (Execução(s) $\neq$ ø $\wedge$ pré_condição(t) $\neq$ nil $\wedge$ pós_condição(t) $\neq$ nil)

$\forall$ a $\in$ ATIVIDADE . (Execução(a) $\neq$ ø $\wedge$ pré_condição(a) $\neq$ nil $\wedge$ pós_condição(a) $\neq$ nil)

5. CONSIDERAÇÕES FINAIS

Neste trabalho é especificado formalmente, através do formalismo Z, o método T.A.& A., integrante de uma metodologia de desenvolvimento de Sistemas de Informação sócio-técnicos. Através deste método é feita a organização dos requisitos do ambiente a modelar, resultando em uma especificação semiformal.

Só é possível especificar formalmente um método quando sua definição está completa. Um método está completamente definido quando apresenta [ROL88] (1) um modelo para a representação das informações, (2) uma linguagem de representação deste modelo, (3) uma seqüência de passos a serem seguidos em sua aplicação, e (4) ferramentas disponíveis para a sua utilização. A especificação formal de um método se baseia nos três primeiros itens. O método T.A.& A. apresentava um modelo para a representação das informações bem definido, baseado em tarefas, atividades e ações. Foi necessário definir a linguagem a ser utilizada para representar este modelo e os passos a serem seguidos durante a utilização do método. Diversas ferramentas estão sendo desenvolvidas, estando algumas em fase de validação.

A especificação de um Sistema de Informação passa por diversas fases durante o seu desenvolvimento. A cada momento, um estado de uma especificação é definido por (1) um conjunto de objetos, (2) um conjunto de relações entre estes objetos, e (3) um conjunto de condições que devem ser satisfeitas pelos objetos e pela especificação como um todo. Uma das importantes aplicações da especificação de um método é a sua utilização no controle de uma especificação. Através da modelagem das possíveis transições desde o momento inicial da especificação até o seu término, a especificação do método utilizado garante a completeza e a coerência da especificação realizada.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABR74] ABRIAL,J.R. Data Semantics. In: KLIMBIE,J.W.; KOFFEMAN,K.L. Data Base Management. Amsterdam, Horth Holland, 1974. p.1-59.

[ALV88] ALVARES,L.O.C. Contribuition à l'Étude du Pilotage de la Modelisation des Systèmes d'Information. Grenoble, Université Joseph Fourier, 1988. 225p.

[ATK89] ATKINSON,M. et al. The Object-Oriented Database Manifesto. Rapport Technique Altair 30-89, 21/08/89. 18p.

[BAN87] BANERJEE,J. et al. Data model issues for object-oriented applications. ACM Transactions on Office Information Systems, New York, 5(1):3-26, Jan.1987.

[BAR85] BARBIC,F.; CERI,S.; BRACCHI,G. Modeling and integrating procedures in Office Information Systems Design. Information Systems, New York, 10(2):149-68, Apr.1985.

[CAR88] CAREY,M.J; DEWITT,D.J.; VANDERBERG,S.L. A Data model and query language for EXODUS. SIGMOD RECORD, New York, 17(3):413-23, Sept.1988.

[CHR90] CHRYSANTHIS,P.K.; STEMPLE,D.; RAMAMRITHAM,K. A Logically distributed approach for structuring office systems. SIGOIS Bulletin, New York, 11(2,3):11-22, 1990.

[CRO84] CROFT,W.B.; LEFKOVITZ,L.S. Task support in an office system. ACM Transactions on Office Information Systems, New York, 2(3):197-212, Jul.1984.

[DEL82] DELOBEL,C.; ADIBA,M. Bases de Donnés et Systèmes Relationnels. Paris, Dunod Informatique, 1982. 449p.

[FIS87] FISHMAN,D.H. et al. IRIS: an Object-oriented Database Management System. ACM Transactions on Office Information Systems, New York, 5(1):48-69, Jan.1987.

[HOP88] HOPPEN,N; OLIVEIRA,J.P.M.; LIMA,L.M.R. Metodologia para modelagem de escritório baseada no nivel de estruturação das atividades. In: REUNIÃO ANUAL DA ASSOCIAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO, 12., Natal, 26-28 set., 1988. Anais. Natal, 1988. p.247-263.

[LEC88] LÉCLUSE,C.; RICHARD,P.; VELEZ,F. O2, an Object-oriented data model. SIGMOD RECORD, New York, 17(3):424-33, Sept.1988.

[LOC88] LOCHOVSKY,F.H. et al. OTM: Specifying office tasks. In: CONFERENCE ON OFFICE INFORMATION SYSTEMS, Palo Alto, California, Mar. 23-25, 1988. Proceedings. ACM SIGOIS, 1988. p.46-54.

[MAI87] MAIOCCHI,R.; PERNICI,B. Verification and refinement of office procedures. In: IEEE COMPUTER SOCIETY OFFICE AUTOMATION SYMPOSIUM, Gaithersburg, Apr. 27-29, 1987. Proceedings. Washington, IEEE, 1987. p.206-16.

[MAR89] MARTINS,J.F.T.; CARRANO,R.M.; MEIRA,S.L. Semântica denotacional para SQL. Revista Brasileira de Computação, Rio de Janeiro, 5(1):53-71, 1989.

[MOT89] MOTZ,R.; FONSECA,D. Especificação formal do modelo relacional usando METOO. In: SIMPÓSIO BRASILEIRO BANCOS DE DADOS, 4., Campinas, 15-17 abr., 1989. Anais. Campinas, R.Vieira Gráfica e Ed., 1989. p.185-96.

[MOT90] MOTZ,R.; FONSECA,D. Estudo formal da estrutura de um modelo de dados orientado a objetos. Revista Brasileira de Computação, Rio de Janeiro, 5(4):25-36, 1990.

[OLI89a] OLIVEIRA,J.P.M.; RUIZ,D.D.A. Modelagem da semântica e dinâmica em formulários eletrônicos. In: SIMPÓSIO BRASILEIRO BANCOS DE DADOS, 4., Campinas, 15-17 abr., 1989. Anais. Campinas, R.Vieira Gráfica e Ed., 1989. p.263-72.

[OLI89b] OLIVEIRA,J.P.M.; RUIZ,D.D.A. Dynamic modeling of office works forms, nets and views. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, 15., Santiago, 10-14 jul. 1989. Actas. Santiago, CLEI, 1989. p.297-307.

[PAN84] PANKO,R.R. 38 Offices: analyzing needs in individual offices. ACM Transactions on Office Information Systems, New York, 2(3):226-34, Jul.1984.

[PER90] PERNICI,B.; ROLLAND,C. Automatic Tools for Designing Office Information Systems - the TODOS Approach. Politecnico di Milano/Université de Paris I, Feb. 1990.

[RAM89] RAMSEY,M. Developping formally verified Ada programs. Software Engineering Notes, New York, 14(3):257-65, May 1989.

[ROL88] ROLLAND,C; FOUCAULT,O.; BENCI,G. Conceptions des Systèmes d'Information - la Méthode REMORA. Paris, Eyrolles, 1988. 351p.

[TUE88] TUENI,M; LI,J.; FARES,P. AMS: A Knowledge-based approach to tasks representation, organization and coordination. In: CONFERENCE ON OFFICE INFORMATION SYSTEMS, Palo Alto, Mar. 23-25, 1988. Proceedings. New York, ACM, 1988. p.78-87.

[YON90] YONG,P. Structure and action in distributed organizations. SIGOIS Bulletin, New York, 11(2,3):1-10, 1990.

ANEXO 1
BNF DA LINGUAGEM DE REPRESENTAÇÃO DO MODELO T.A.& A.

```
<objeto> ::=   Objeto <nome_objeto> : <tipo_objeto>
             | Objeto <nome_objeto>
                   atributos <lista_objetos>
                   [ componentes <lista_objetos> ]
                   [ restrições: <expressão_lógica> ]
             | Objeto <nome_objeto>
                   componentes <lista_objetos>
                   [ restrições: <expressão_lógica> ]
             | Objeto <nome_objeto>
                   subclasse de <nome_objeto>
                   [ atributos <lista_objetos> ]
                   [ restrições: <expressão_lógica> ]
             | Objeto <nome_objeto>
                   conjunto de <nome_objeto>
                   [ atributos <lista_objetos> ]
                   [ restrições: <expressão_lógica> ]
             | Objeto <nome_objeto>
                   subconjunto de <nome_objeto>
                   [ restrições: <expressão_lógica> ]
```

```
<nome_objeto> ::= <nome>
<tipo_objeto> ::= Inteiro | Real | String
<lista_objetos> ::= <nome_objeto> | <nome_objeto> , <lista_objetos>
<lista> ::= <elemento> | <elemento> , <lista>
<elemento> ::= <nome> | <número_inteiro> | <número_real>
<tarefa> ::=    Tarefa <verbo> <nome_objeto> <resto>
                    pré-condição: <expressão_lógica>
                    execução: <subtarefa_atividade>1
                    pós-condição: [ <expressão_lógica> ]
<subtarefa> ::= Subtarefa <identificação_da_subtarefa>
                    pré-condição: <expressão_lógica>
                    execução: <subtarefa_atividade>
                    pós-condição: [ <expressão_lógica> ]
<identificação_da_subtarefa> ::= <verbo> <nome_objeto> <resto>
<subtarefa_atividade> ::= <identificação_da_subtarefa>
                    | <identificação_da_atividade>
                    | <subtarefa_atividade> ; <subtarefa_atividade>,
<atividade> ::= Atividade <identificação_da_atividade>
                    pré-condição: <expressão_lógica>
                    execução: <corpo_exec>
                    pós-condição: [ <expressão_lógica> ]
<identificação_da_atividade> ::= <verbo> <nome_objeto> <resto>
<corpo_exec> ::= <ações> | indefinida
<ações> ::= <ação> | <ação> ; <ações>
<ação> ::= criar <nome_objeto> [ com <lista> ]
        | obter <variável> de <nome_objeto> sendo <expressão_lógica>
        | remover <nome_objeto> [ <lista> ]
        | remover <variável>
        | atualizar <nome_objeto> de <nome_objeto> para <elemento>
        | atualizar <variável> para <elemento>
<nome> : formado por letras e do caracter "_".
<expressão_lógica> : escrita em uma linguagem de lógica de primeira
    ordem, envolvendo nomes e atributos de objetos; ou uma
    seqüência qualquer de caracteres entre um par de duplas aspas.
<número_inteiro> : valor numérico inteiro.
<número_real> : valor numérico real.
<verbo> : classe gramatical portuguesa; deve ser apresentado na
    forma infinitiva.
<resto> : seqüência qualquer de palavras.
```

ANEXO 2
FORMALISMO EMPREGADO NA ESPECIFICAÇÃO

O formalismo utilizado na especificação do método T.A.& A. é um subconjunto da linguagem Z [ABR74, DEL82, ALV88]. Numa especificação em Z devem ser definidos (1) conjuntos de entidades, (2) relações binárias entre estes conjuntos e (3) fórmulas lógicas para representar condições que se aplicam aos conjuntos ou às relações.

1. CONJUNTOS DE ENTIDADES

A descrição de um sistema inicia pela descrição dos conjuntos de objetos que o compõe - objetos, lugares, pessoas. Cada um destes objetos, concreto ou abstrato, constitui uma entidade. Cada objeto possui uma identidade única, fato este que permite diferenciar uma

entidade de outra. Objetos de igual natureza são agrupados em conjuntos de entidades os quais, em Z, são equivalentes aos da matemática.

2. RELAÇÕES BINÁRIAS ENTRE ENTIDADES

Uma relação binária entre dois conjuntos, não necessariamente distintos, representa uma associação existente entre entidades destes dois conjuntos. Esta associação é representada por uma função de mapeamento. As funções podem ser parciais ou totais, mono ou multivaloradas. O conjunto domínio da função define se ela é total (a função é definida para todos os elementos do domínio) ou parcial. O conjunto imagem define a cardinalidade da função como monovalorada ou multivalorada - se a avaliação da função fornecer somente um valor ela é monovalorada. Para a definição destas funções pode ser utilizada a seguinte representação gráfica [ALV88]:

```
------> função monovalorada total
----/-> função monovalorada parcial
----->> função multivalorada total
---/->> função multivalorada parcial
```

As seguintes convenções para representação dos elementos são usualmente empregadas:
- nomes de entidades em letras minúsculas;
- nomes de conjuntos em letras maiúsculas;
- nomes de funções monovaloradas em letra minúscula;
- nomes de funções multivaloradas iniciam com letra maiúscula;
- representação da cardinalidade de uma função multivalorada através de um par de valores inteiros entre parêntesis, separado por vírgula, sendo o primeiro a cardinalidade mínima e o segundo, a máxima.

Assim, por exemplo, a representação gráfica

```
            f
E1 << -/--------------- > E2
                        F
```

representa uma função f entre os conjuntos E1 e E2, total e monovalorada, sendo a função inversa dada por F, parcial e multivalorada.

Uma extensão do formalismo Z é a definição de associações compostas [ALV88]. Nestas, um ou mais argumentos da função podem ser opcionais. Entretanto, pelo menos um dos argumentos deve ser obrigatório. A representação dos argumentos opcionais é feita através de um expoente zero no nome do conjunto correspondente.

3. FÓRMULAS LÓGICAS

As fórmulas lógicas em Z obedecem às regras clássicas [illegible] fórmulas ou expressões lógicas. São também chamadas de [illegible] (quando o enfoque utilizado é o de sistemas especialistas) [illegible] restrições (no enfoque de bancos de dados).

Relatórios de Pesquisa

RP-200: "Primitivas de acesso ao Banco de Dados do Projeto AMPLO",
Outobro/92.
LIA GOLDSTEIN GOLENDZINER

RP-199: "Semântica formal de linguagens de programação", setembro 1992.
A.S. CASTRO VERA

RP-198: "Métodos e linguagens de especificação formal", setembro 1992.
A.S. CASTRO VERA

RP-197: "LINGUAGENS VISUAIS: uma abordagem para a programação de computadores, especificação de interfaces gráficas, visualização de software e acesso a bancos de dados", setembro 1992.
C.M.D.S. FREITAS

RP-196: "LIBTEX: uma biblioteca para síntese de texturas em imagens de computação gráfica. Agosto 1992.
M. WALTER

RP-195: "A cross-indexed guide to the texture literature". Julho 1992.
M. WALTER

RP-194: "PGX-Pacote de rotinas gráficas para o AMPLO em ambiente Xwindow". Julho 1992.
D. G. FRIDMAN; F. HESSEL; A. E. NEUJAHR

RP-193: "Operating systems and concurrent programming". Julho 1992.
W. PANDIKOW

RP-192: "Estrutura de probabilidade com aritmética intervalar", Julho 1992.
H. KORZENOWSKI, M.A. CAMPOS

RP-191: "Um misto de visão clássica e moderna de topologia", Julho 1992.
B.M. ACIÓLY; D.M. CLAUDIO