Universidade Federal do Rio Grande do Sul

Instituto de Informática

Curso de Pós-Graduação em Ciência da Computação

RUMO A UM MODELO DE IMPLEMENTAÇÃO DE ESPECIFICAÇÕES NÃO PROCEDURAIS DE SISTEMAS TRANSACIONAIS EM BANCO DE DADOS

por

HUBERT AHLERT

RP-190 Junho/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

RESUMO

O tema discutido nesta proposta de tese é a construção automática de sistemas, particularmente a implementação de especificações não procedurais voltadas a sistemas transacionais em banco de dados. Como formas de viabilizar uma solução, são avaliadas alternativas usando métodos de tradução de especificações (compilativo), métodos de execução da especificação (interpretativo) e métodos híbridos que incorporem características compilativas e interpretativas. Como contribuição espera-se o desenvolvimento de técnicas de tradução e/ou interpretação de especificações.

ABSTRACT

The subject being discussed in this thesis proposal is the automatic systems construction, particularly the implementation of non-procedural specifications for transaction-driven database systems. As a means of obtaining a solution, some alternatives are evaluated using specification translation methods (compilative), specification execution methods (interpretative) and hybrid methods including both interpretative and compilative characteristics. It is expected that this work will lead to the development of techniques for translation and/or interpretation of specifications.

PALAVRAS-CHAVE:

Programação Automática, Construção Automática de Sistemas, Ferramentas para Construção de Sistemas, Redes de Petri, Proceduralização de Especificações Declarativas.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Desde os idos tempos da programação diretamente em código de máquina e em assembler, os pesquisadores de informática vêm, progressivamente, procurando mecanismos para minimizar as intervenções humanas no processo de programação dos computadores. Tencionam modificar o paradigma de construção de programas e sistemas no sentido de evitar que o projetista indique, passo-a-passo, como o computador deve realizar as tarefas que lhe são conferidas, ou seja, procuram fazer com que a máquina incorpore um maior grau de conhecimento sobre técnicas de análise e de programação, permitindo, assim, aumentar o grau de automação. A idéia é ter-se representado, na máquina, conhecimentos (experiências sobre análise e programação) suficientes para permitir a delegação, ao computador, de um maior nível de decisão sobre as tarefas que implementa.

Assim, o processo de desenvolvimento de sistemas vem acompanhando os constantes avanços tecnológicos sendo registrados na área de software, ou seja, fazendo uso também das melhorias incorporadas a novos lançamentos em software (linguagens, aplicativos, utilitários, ferramentas). Isto exige eventuais reformulações nas metodologias e, também, a concepção de novas ferramentas de suporte a estas metodologias. Como consequência, o ciclo de vida do software pode sofrer modificações para adaptar-se à nova realidade imposta por um ou outro paradigma de construção de sistemas. O ciclo de vida do software se mantém em sintonia com o perfil das ferramentas que lhe dão suporte e é este perfil que dita o grau de automação a ser obtido nas tarefas de desenvolvimento do sistema.

Investigações da engenharia de software, com resultados práticos em produtos como ferramentas CASE e linguagens de quarta geração, vêm investindo esforços numa constante busca da automação de tarefas de analistas e programadores. Dentre estes esforços, cabe mencionar também os experimentos sobre o paradigma de programação automática /BAL 85a/. Este modelo prega uma construção automatizada dos sistemas mediante sucessivos

refinamentos de especificação, controlados por computador, até a obtenção do código executável do programa. Em relação a este tema, um estudo anterior ao presente trabalho foi apresentado em /AHL 90a/.

Sob o escopo de um ambiente que implemente as idéias de construir, de forma automática, um sistema, deve-se considerar um elenco de conhecimentos sobre técnicas e métodos que podem ter origens em diversas áreas da informática. Além da própria engenharia de software, que se preocupa com metodologias e ferramentas de apoio ao projeto de sistemas, conhecimentos na área de banco de dados e inteligência artificial devem ser buscados.

Para a presente proposta de estudos, o tema é abordado sob o enfoque de técnicas e ferramentas que permitam obter uma implementação de especificações não procedurais, de forma direta, sem necessidade de transcrições manuais para um conjunto de procedimentos em alguma linguagem de programação.

Considerando que uma linguagem de programação é também uma especificação formal, só que com maiores niveis de detalhes de implementação, e que as atuais técnicas de construção de programas já possuem sedimentados métodos compilativos e interpretativos para a geração de código executável (linguagens compiladas x linguagens interpretadas), a questão passa a ser a de estrapolar estes métodos a um nível de abstração maior, ou seja, assegurar que estes métodos consigam também mapear definições não algorítmicas em procedimentos que mostrem as alternativas de implementar estas definições. Nestes termos, pretende-se estudar as formas de mapear as especificações não procedurais para correspondentes procedimentos executáveis dentro do escopo de sistemas de informações do tipo transacional.

No capítulo 2 são revisados conceitos do paradigma de programação automática. Também é apresentada a proposta de uso, como ponto de partida do projeto, de uma linguagem de especificação desenvolvida no CPGCC-UFRGS para o processo de modelagem conceitual de sistemas.

No capítulo 3 é delimitado o escopo de um problema que mereça investigações e estudos mais aprofundados. Na intenção de dar uma solução para este problema, nos capítulos 4 e 5 são discutidas as ações a serem tomadas como trabalho de tese.

No capítulo 6 é feita uma avaliação dos atuais métodos de implementação de Redes de Petri (como classe de especificações não procedurais) e, no capítulo 7, justificado porque estas soluções não se aplicam adequadamente ao problema da tese.

A proposta de tese é discutida no capítulo 8, culminando com a proposição de uma arquitetura de construção de sistemas que sintonize com esta proposta sendo apresentada no capítulo 9.

2. UM AMBIENTE PARA CONSTRUÇÃO AUTOMÁTICA DE SOFTWARE

Para situar o tema de investigação, escolhido como assunto de tese ora sendo proposto, dentro do contexto da construção automática de sistemas a partir de uma especificação, neste capítulo tenciona-se relatar a linha de pesquisa que levou a estabelecer o tema e os resultados preliminares de trabalhos anteriores nesta área.

2.1 A PROGRAMAÇÃO AUTOMÁTICA

A programação automática, segundo /BAL 85a/, pode ser vista como o processo de construção de programas/sistemas onde, a partir de uma especificação informal ou semi-formal submetida a sistemas transformacionais (mecanismos de apoio à transformação de especificações), é obtido automaticamente o código executável, com a mínima intervenção humana. Este paradigma pressupõe sucessivos refinamentos, controlados por computador, realizando o mapeamento entre especificações através da aplicação de regras de transformação até a obtenção de código executável. Cada nova versão da especificação é obtida da versão anterior submetendo-a a regras de transformação que comprovadamente mantenham a equivalência entre as especificações.

Segundo Alencar e Lucena /ALE 89/, em um método de desenvolvimento de programas por transformação, a partir de uma versão inicial do problema (especificação original) e pela aplicação de transformações, tanto em direção ao nível mais baixo de abstração (nível de implementação) quanto pela aplicação de transformações dentro de um mesmo nível (em busca, por exemplo, de versões mais eficientes do algoritmo), é derivada uma versão eficiente da especificação de solução do problema, preservando a correção da definição original.

Balzer /BAL 81/ refere-se a este modelo como uma **implementação transformacional**. Baseado neste paradigma, Balzer /BAL 85a/ propõe um ciclo de desenvolvimento conforme ilustrado na figura 1.

Publicações das décadas de 60/70, de autores como Hoare /HOA 69/ e Dijkstra /DIJ 68/,/DIJ 76/ com pesquisas ligadas a prova matemática da correção de programas, também já induziam a técnicas de construção de programas/sistemas que usam o computador como instrumento para facilitar a tarefa de programação. Dijkstra, por exemplo, procurou associar as idéias de prova de correção (análise, verificação) à derivação (síntese, desenvolvimento) de programas.

A prática de programação convencional, por si só, é um processo de transformação. Utiliza uma especificação informal ou semi-formal e, a partir de um agente representado pelas técnicas de programação sendo manipuladas pelo programador, em um processo altamente dependente do fator humano, produz uma especificação formal representada pelo programa fonte.

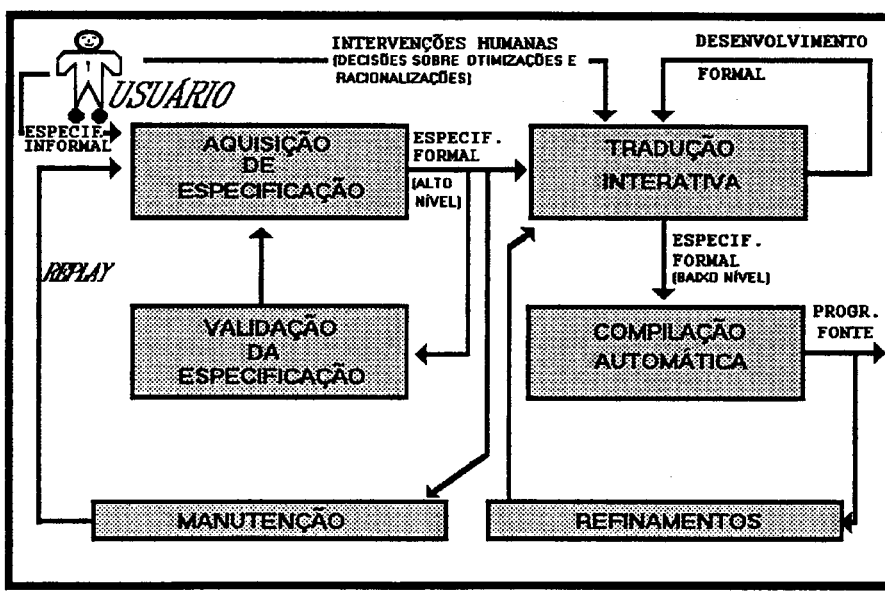


Fig. 1 - Ciclo de vida no paradigma de programação automática (Adaptado de /BAL 85a/)

Na busca de um ambiente de programação automática, a idéia básica gira em torno da minimização de intervenções humanas no processo de construção dos programas com a consequente minimização de erros de programação e fazendo com que o desenvolvimento transformacional de software seja uma rotina automática, controlada pelo próprio computador. Com isto, as tarefas de analistas e programadores podem, afinal, ser automatizadas.

Sob este ponto de vista, deve-se considerar que as ferramentas de programação automática precisam incorporar conhecimento suficiente sobre técnicas de programação para servir de ponte na lacuna existente entre o escopo de linguagens de especificação e o escopo de uma linguagem convencional de programação.

Em virtude do alto grau de heurística que deve estar associado às ferramentas de programação automática, uma alternativa de implementação que merece uma investigação mais minuciosa envolve o uso de sistemas especialistas e técnicas de inteligência artificial.

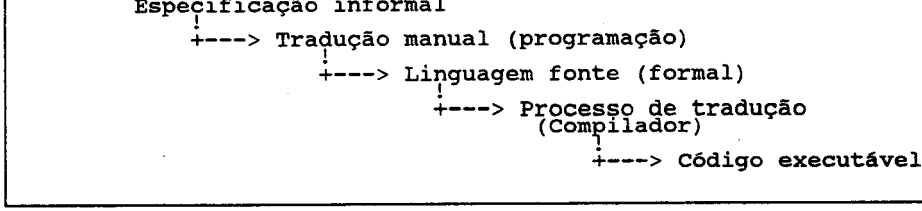
Talvez as pesquisas de Balzer, e outros autores que defendem o uso de sistemas transformacionais na geração automática de sistemas, não tenham obtido resultados práticos muito satisfatórios (pouca literatura recente sobre o assunto). Serviram, no entanto, para abrir um novo horizonte de investigações e reforçaram a idéia de que a área de inteligência artificial pode trazer grandes contribuições para a construção de ferramentas de apoio à engenharia de software que levem a melhorias nas tarefas de análise e programação.

O tema **"programação automática"** foi amplamente discutido em publicações anteriores do autor /AHL 90a/ /AHL 91a/. Nestes trabalhos, foi possível concluir que a programação automática, baseada em sistemas transformacionais, destaca sua principal diferença, em relação à programação convencional, no processo de tradução que é realizado automaticamente através de refinamentos sucessivos na especificação (em diferentes niveis de

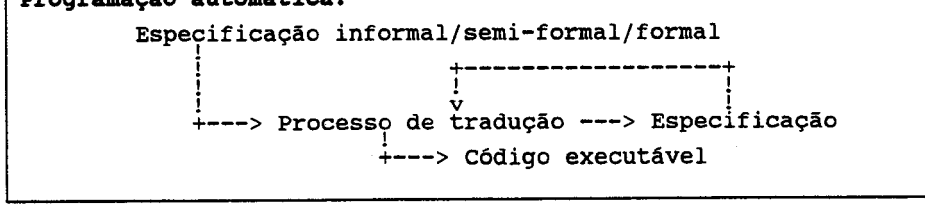
abstração) e, eventualmente, com possibilidade do uso de uma linguagem informal ou semi-formal (próxima a linguagem humana) como ponto de partida. Na programação convencional, o processo de tradução se limita a uma compilação de uma linguagem de programação (especificação formal com detalhes de implementação).

Assim, pode-se resumir as diferenças entre a programação convencional e a programação automática confrontando os seguintes diagramas representativos das transformações que ocorrem nos processos de compilação/tradução.

Programação convencional:



Programação automática:



2.2 O MODELO DE PARTIDA

Em um ambiente de construção automática de sistemas, como em qualquer outra metodologia de análise e projeto, a fase de análise de requisitos e modelagem do sistema assume um papel de fundamental importância pois dela depende a correta compreensão do sistema alvo. É a partir dela que se estabelecem as propriedades do sistema a implementar.

Como ponto de partida para modelagem do sistema, segundo os propósitos da presente proposta, torna-se necessária uma linguagem com as seguintes características:

*ser independente de implementação;
*permitir especificação completa das funcionalidades do sistema (estados + transformações);
*ser formal para evitar ambiguidades;

A linguagem escolhida para ser usada no módulo de modelagem conceitual do sistema é uma forma híbrida que combina Diagramas E/R, Diagramas de Fluxo de Dados (DFD) e Redes de Petri de alto nível. É conhecida como abordagem ER/T (Entidade-Relacionamento/Transação) e resulta de uma dissertação de mestrado /PER 90/ desenvolvida no CPGCC da UFRGS.

2.2.1 Visão geral da Abordagem ER/T

A abordagem ER/T integra dois tipos básicos de modelos de sistemas de informações: Modelo de Dados e Modelo de Funções. Permite estabelecer um conjunto de classes de modelos que descrevem dados e funções em diferentes niveis de abstração ou detalhe. O que diferencia cada modelo é a quantidade de informações relativas ao sistema que o modelo agrega. Apesar de estar previsto um nível informal (modelo ER/T informal) na metodologia original proposta em /PER 90/, o **modelo de partida** para a presente proposta é um modelo ER/T formal (denominado semi-formal em /PER 90/).

Os elementos da abordagem ER/T usados na construção dos modelos são:

.conjuntos de entidades (retângulo);
.conjuntos de relacionamentos (losângos ligados por setas às entidades);
.processos (círculos);
.fluxos de dados (setas orientadas ligando elementos da abordagem ER/T);
.agentes externos (2 retângulos semi-sobrepostos);
.lugares de controle (retângulos pontilhados);
.anotações complementares (anotações associadas aos processos).

A orientação das setas nos fluxos de dados tem significado próprio (inclusão, exclusão, alteração, teste), conforme mostra a figura 2. Esta semântica associada à orientação dos fluxos foi adaptada a partir dos conceitos de **ramos alteradores** e **ramos restauradores** das Redes de Petri descritas em /HEU 89a/.

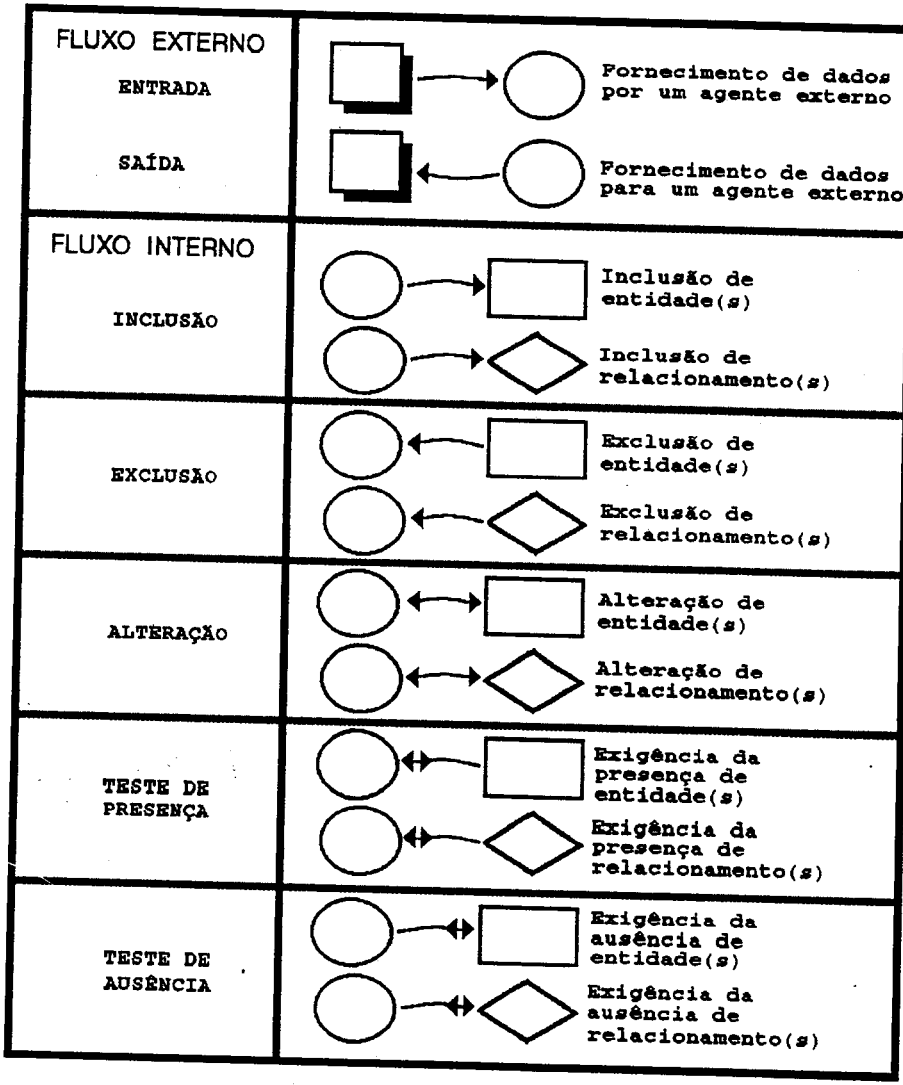


Fig. 2 - Tipos de fluxos da abordagem ER/T
(Extraído de /PER 90/)

Para a representação diagramática dos modelos, a abordagem ER/T sugere a construção de diagramas detalhando, cada qual, um processo individualmente. Neste diagrama é desenhado o processo e todos os objetos associados aos seus fluxos (entidades, relacionamentos, agentes externos, etc). Isto impõe uma **característica transacional** ao modelo, colocando uma transação (processo) como unidade atômica.

Os processos especificam as propriedades dinâmicas do sistema indicando como os dados de entrada são transformados em dados de saída. A ocorrência de uma transformação (transação) ocasiona o consumo do conjunto de fluxos de entrada e de exclusão, a produção de fluxos de saída e de inclusão e a alteração dos fluxos de alteração a partir de regras de habilitação, assemelhança das encontradas nas Redes de Petri /HEU 89a/. A figura 3 ilustra uma transação modelada na abordagem ER/T.

As regras de habilitação de uma transação podem ser sumarizadas em:

1- Para fluxos de exclusão/alteração/teste de presença: presença das entidades (ou relacionamentos);
2- Para fluxos de inclusão/teste de ausência: ausência das entidades (ou relacionamentos);
3- Dados de entrada presentes nos agentes externos;
4- Restrições de integridade do E/R (1:1, 1:N, N:M) não violadas;
5- Restrições das anotações dos processos não violadas.

Para modelar a sincronização de transações em função do tempo (relógio) ou da ocorrência de outras transações (sequência de transações) são utilizados os lugares de controle (depósito de dados de sinalização).

Para os processos que realizam o consumo dos fluxos de entrada (tratamento de entrada de dados), a abordagem ER/T convenciona que o procedimento de tratamento de inconsistência

padrão (inclusão para já existente, alteração para inexistente, etc) está implícito no desenho do processo, sendo desnecessária sua representação explícita no diagrama.

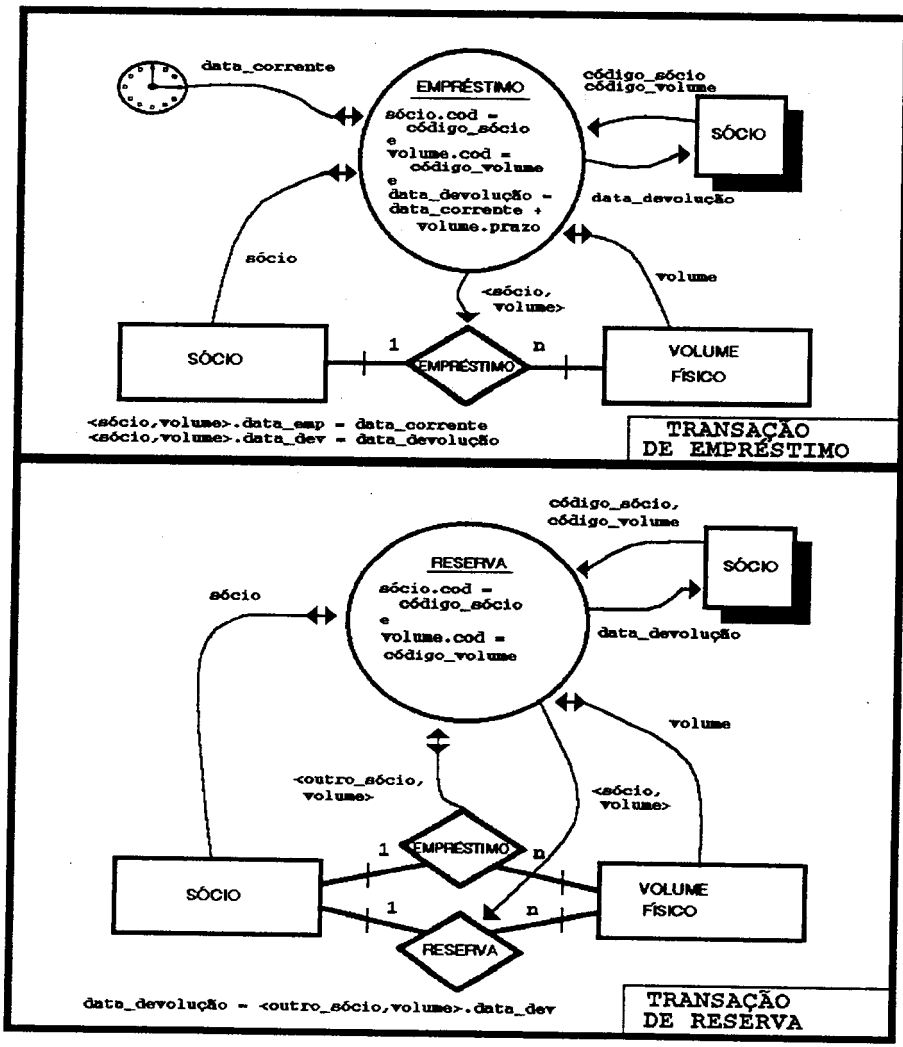


Fig. 3 - Transação no modelo ER/T
(Extraído de /PER 90/)

2.2.2 O Modelo ER/T como representante de classe de linguagens de especificação

As técnicas de descrição formal (TDF) tem o objetivo de permitir a definição do comportamento do sistema de forma completa, precisa, consistente e sem ambiguidades. Dentre estas técnicas existe uma classe de linguagens de especificação que se baseiam na representação de transições de estados, descrevendo as pré e pós condições de habilitação de uma transição (transação na terminologia de banco de dados).

Da literatura é possível destacar uma série de linguagens que seguem esta classe de TDF.

Por exemplo:

a) **ESTELLE**: (/ISO 86/,/LIN 86/)

Estelle é uma linguagem de especificação baseada em um modelo de transição de estados com aplicação típica em protocolos de comunicação. Uma especificação Estelle é um conjunto de módulos que interagem entre si, através de canais e pontos de interações, e onde o comportamento interno de cada módulo é caracterizado por uma transição de estados não determinística. Permite também uma hierarquização entre os módulos além da especificação de paralelismos. A especificação das transições (TRANS) é composta por cláusulas que indicam a condição da transição (FROM,WHEN,PRIORITY) e a ação (TO,bloco BEGIN/END) a ser tomada como efeito da transição. Uma transição é habilitada se todas as condições forem satisfeitas.

b) **TAXIS**: (/PEC 88/,/BOR 84/)

Taxis é uma linguagem de especificação para projeto de banco de dados interativos baseado em modelo de dados semântico e orientado a objetos. Permite a modelagem tanto das propriedades estáticas quanto das dinâmicas. Tem também características de encapsulamento, generalização/especialização, agregação e classificação. A modelagem da dinâmica do sistema é feita via especificação de transações (TRANSACTION <nome> WITH) indicando,

além de parâmetros, as pré-condições (PREREQUISITES), as ações (ACTIONS) e o tratamento de exceções (FOR EXCEPTION).

c) **SHM+**: (/BRO 81/,/PEC 88/)

SHM+ (Extended Semantic Hierarchy Model), representante de um modelo de dados semântico, permite a modelagem de ações individuais e de transações de banco de dados. Para as ações (ACTION) são especificados os dados de entrada e saída (IN/OUT) as pré e pós condições (PRE-CONDITION/POST-CONDITION) e a operação sobre o banco de dado (DB-OPERATION). Na especificação das transações (TRANSACTION) também são indicadas as pré e pós condições para sua habilitação além dos objetos que a transação acessa (SCOPE).

d) **ER-R**: (/TAN 91/)

O ER-R é uma extensão do modelo ER, acrescentando-lhe regras do tipo situação-ação baseadas em eventos o que possibilita incorporar os aspectos comportamentais (operações) das aplicações. A notação para o diagrama ER-R (ER com regras) é a mesma do modelo ER acrescida do uso de um paralelograma para a representação de uma regra e fluxos de entrada e saída associados ao paralelograma para a representação de correspondentes eventos. É um modelo baseado em pré e pós condições, segundo os autores, próprio para modelar S.G.B.D ativos (com regras e gatilhos). Textualmente o diagrama é especificado por: RULE-NAME, TRIGGERING-EVENT, CONDITION, ACTION e RESULTANT-EVENTS.

Algumas linguagens aqui exemplificadas (ex.: Estelle, Taxis) especificam as pré e pós condições de habilitação de uma transição em uma forma procedural. Para os propósitos do presente trabalho cabe, no entanto, considerar técnicas que adotam uma forma declarativa (não procedural) para representar a transição de estados. Neste elenco figuram as redes de Petri como um exemplo clássico.

A rede de Petri, além da capacidade de modelar graficamente as atividades de um sistema a um nível conceitual e

de forma declarativa e, ainda, enquadrando-se na classe de TDF em discussão (descrevendo relações de causalidade numa lógica baseada nas noções de estado e evento (transição) /BAK 90/), permitem também especificar concorrência e sincronização. Cabe aos mecanismos de implementação das redes manter esta última característica.

Como exemplos do uso das redes, aparecem na literatura:

a)**RPO**: (/GAR 90/)

No modelo RPO (PNO na versão em inglês) as redes de Petri tipo predicado/transição são acrescidas de estruturas de dados e operações do paradigma de orientação a objetos (classes, instâncias, mensagens) a fim de superar as restrições relativas a representação de dados presentes nas redes clássicas. Utilizam as regras de produção como forma natural de representar textualmente o modelo RPO.

b)**PROTOB**: (/BAL 91/)

No modelo PROTOB a relação entre objetos é representada por fluxos de dados (DFD), o comportamento do objeto através das redes de Petri e a descrição da estrutura dos objetos mediante um arquivo texto escrito na linguagem C ou ADA. Nesta abordagem, um sistema é um conjunto de objetos, cada qual com determinado comportamento (rede) trocando mensagens entre si (grafo de fluxo). Na definição do comportamento do objeto, cada transição da rede, com seus lugares causa/efeito, é considerada a representação gráfica de regras de produção "IF-THEN".

b)**ESDM**: (/CHA 91/)

O modelo ESDM (Executable Semantic Data Model) permite a modelagem de aspectos estruturais (dados) e de aspectos comportamentais (funções) de aplicações em banco de dados. Para modelagem estrutural utiliza uma extensão do modelo ER (com entidades, relacionamentos, atributos, agregações e hierarquia de subset). Para modelagem do comportamento utiliza o mecanismo das redes de Petri (tipo transições e lugares). Associado a cada transição pode haver especificado, textualmente, uma lista de

parâmetros (dados de entrada), as pré-condições (condições dos dados no banco de dados para permitir a habilitação) e as pós-condições (modificações dos dados do banco de dados após a transição). O modelo ESDM é posteriormente traduzido para Ingres.

Dentre este elenco de linguagens referenciadas, figura também o modelo ER/T como representante da classe de TDF baseada em pré e pós condições, incorporando também as principais características de uma rede de Petri e, adicionalmente, permitindo a especificação de transações de banco de dados /HEU 91/ que é um dos principais objetivos da presente proposta de tese.

2.2.3 Justificativas do uso da Abordagem ER/T

A motivação que fundamentou a escolha da abordagem ER/T como linguagem de especificação para a modelagem conceitual do sistema no ambiente proposto relaciona-se, basicamente, as seguintes características:

- .abordagem gráfica de relativa simplicidade que facilita o uso , por parte do projetista, e a interação com o usuário; é de fácil assimilação por parte do projetista pois se fundamenta em conceitos já bastante difundidos (E/R, DFD, Redes);
- .tem base formal o que pode viabilizar uma transformação automatizada para código executável;
- .é um modelo completo que permite representar dados e funções;
- .própria para a especificação de sistemas transacionais;
- .É uma especificação não procedural do tipo causa e efeito, refletindo os estados de um sistema pré e pós transformação (transição de estados);
- .preenche o requisito de independência de implementação;

Em vista da combinação destas características, o modelo ER/T mostrou-se interessante como **modelo de partida** para a realização dos experimentos de implementação de especificações.

Não está sendo afirmado, aqui, que este é um formalismo ideal para a modelagem de sistemas mas sim que a abordagem ER/T pode ser considerada como um exemplo para validar as idéias relativas à implementação de especificações. Posteriormente, estas idéias podem ser aplicadas a outros formalismos. Justifica-se, também, a escolha em vista da proximidade de quem desenvolveu a linguagem de especificação.

Como a abordagem ER/T ainda se encontra em uma fase experimental, eventualmente alguns ajustes deverão ser realizados para consolidar a técnica e aumentar a completeza da especificação. Os melhoramentos do modelo ER/T podem se apoiar em características já sedimentadas em outros modelos de modelagem de sistemas/programas.

Embora a abordagem ER/T se apresente como um formalismo completo, determinadas propriedades estão implícitas na forma de realizar a especificação (por exemplo: iterações), ficando a explicitação (detalhamento do funcionamento) a cargo de mecanismos automáticos de dedução durante a implementação. Outras propriedades são apresentadas na forma declarativa (não procedural) exigindo, também, algoritmos específicos de detalhamento (por exemplo: sequenciamento de operações de uma transação). Estas questões serão também alvo de investigações no decorrer da pesquisa.

3. O PROBLEMA

Conforme discutido no capítulo anterior, um ambiente de construção automática de sistemas busca evitar as traduções manuais realizadas, pelo programador, no processo de mapeamento de uma especificação para uma linguagem de programação ou aquelas traduções manuais sucessivas entre diferentes niveis de especificação, cada qual com maior ou menor abstração (a linguagem de programação também é uma especificação formal só que em um nível que incorpora maiores detalhes de implementação), até chegar ao código executável.

Considerando este ambiente e a característica de uma linguagem não procedural presente no modelo de partida da modelagem do sistema, identificou-se o tema "implementação de especificações não procedurais" como um problema que mereceria uma investigação.

Restringindo o escopo de aplicações cobertas pela proposta ao nível de sistemas de informações do tipo transacionais, com ênfase na utilização de banco de dados, o tema central passa a ser:

A BUSCA DE UM MODELO DE IMPLEMENTAÇÃO DE ESPECIFICAÇÕES NÃO PROCEDURAIS DE SISTEMAS TRANSACIONAIS EM BANCO DE DADOS.

4. AS FORMAS DE REALIZAR UMA SOLUÇÃO

Delineado o problema, cabe agora avaliar as formas de realizar uma solução, tendo como premissas básicas:

Ponto de partida: Especificação não procedural (abstrata)

Resultado final perseguido: Implementação (sequência de operações executáveis)

Método de mapeamento desejado: Algum processo automático ou semi-automático de tradução (poucas intervenções humanas)

Analisando a situação sob o prisma de linguagens de programação e ambientes de construção de software, as alternativas óbvias recairiam sobre 2 enfoques básicos: a)uso de algum método compilativo; b)uso de algum método interpretativo.

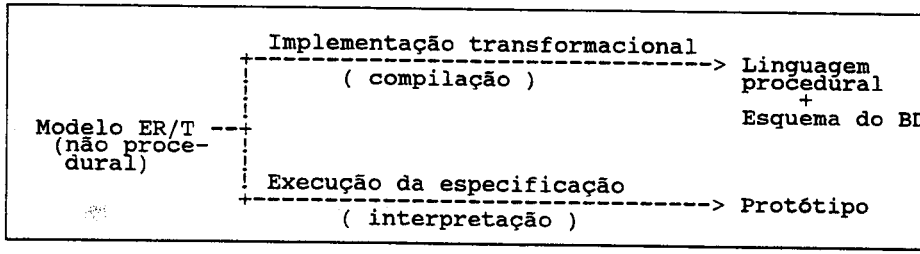
Para o escopo de uma linguagem de especificação mais abstrata (não algorítmica), convem avaliar algumas idéias da literatura onde, por exemplo, Feather /FEA 82a/ sugere mapear cada construção da linguagem de especificação em uma alternativa de implementação, enquadrando-se na idéia de implementação transformacional pregada por Balzer /BAL 81/. Neste caso, pressupõe-se uma comprovada correção no mapeamento entre os diferentes niveis de especificação. Por outro lado, Cohen e Balzer /COH 82/ propõem realizar uma execução das especificações com o intuito de fazer uma validação do comportamento destas.

Assim, pode-se considerar como alternativas de realizar uma solução:

1) Implementação transformacional, segundo um modelo compilativo, para gerar um software de produção do sistema;

2) Execução da especificação (simulação, interpretação, prototipação), segundo um modelo interpretativo, para gerar um protótipo do sistema.

Portanto, os experimentos sobre implementação de especificações devem considerar a seguinte situação:



Eventualmente alguma forma híbrida, dentro do escopo destas alternativas, deve ser buscada.

O quadro da figura 4 ilustra uma comparação sucinta entre as duas abordagens.

Fig. 4 - Implementação transformacional vs Execução de especificações

4.1 ABORDAGEM IMPLEMENTAÇÃO TRANSFORMACIONAL

A abordagem de implementação transformacional pressupõe a tradução da especificação (aqui modelada em ER/T) para alguma linguagem alvo (normalmente procedural) com posterior compilação. Na década de 80, inúmeros trabalhos foram publicados destacando o modelo transformacional para chegar-se a código executável a partir da especificação. Uma coletânea comentada de referências bibliográficas sobre o tema, avaliando publicações de autores como Balzer (/BAL 81/,/BAL 82/,/BAL 85a/), Barstow /BAR 77/, Bauer /BAU 89/, Cheatham (/CHE 72/,/CHE 81/), Darlington /DAR 79/, Feather /FEA 82b/, Fickas /FIC 85/, Kant /KAN 81/, Paige /PAI 83/ e Partsch /PAR 83/, foi anteriormente realizada em /AHL 90a/.

Nesta categoria de solução enquadram-se, também, trabalhos descritos em publicações como /RED 89/, /SMI 90/, /HIL 90/ e /SIL 90/.

Em /RED 89/ é apresentado um sistema de suporte à derivação transformacional de programas denominado FOCUS. O sistema utiliza uma linguagem de especificação que combina linguagem funcional com linguagem de programação em lógica.

Smith /SMI 90/, apresenta um software transformacional (KIDS) para desenvolvimento semi-automático de programas a partir de uma especificação formal em linguagem funcional REFINE, gerando código Lisp.

O artigo do Hildum e Cohen /HIL 90/ descreve uma linguagem que permite especificar regras de transformação de programas que viabilizam traduções entre uma variedade de linguagens de programação. As regras de transformação definem o mapeamento de uma sequência de código para outra sequência.

IMPLEMENTAÇÃO TRANSFORMACIONAL vs EXECUÇÃO DA ESPECIFICAÇÃO

MODELO ER/T
(Nao Procedural)

- IMPLEMENTAÇÃO TRANSFORMACIONAL — Linguagem Procedural + Banco de Dados
- EXECUÇÃO DA ESPECIFICAÇÃO ——— Protótipo

IMPLEMENTAÇÃO TRANSFORMACIONAL

- Modelo compilativo
- Automatiza a prática de programação convencional (baseada em processo de transformação de especificação informal em fonte de programa)
- Permite gerar código para ambientes diferentes do de transformação
- Mecanismos de depuração via comandos de interrupção ("trace")
- Modificações de especificações exigem recompilação
- Voltado para produção do sistema

RESULTADO: Um sistema autônomo (autocontido)

EXECUÇÃO DA ESPECIFICAÇÃO

- Modelo interpretativo
- Avaliação prévia do comportamento do sistema a partir da especificação
- Normalmente limitada ao protótipo em um ambiente específico. Pode, porem, incluir módulos de emulação de equipamentos
- Próprio para a execução passo-a-passo para acompanhamento do estado da máquina
- Modificações de especificações afetam somente trechos alterados
- Voltado para prototipação e validação

RESULTADO: Um sistema dependente de ambiente de exec. ("Run-time")

Em /SIL 90/ é apresentado um tradutor, escrito em Pascal, que transforma Redes de Petri a objetos (RPO) na linguagem LIS desenvolvida no LCMI/UFSC. O tradutor recebe a descrição RPO já tratada (na linguagem LRPO /CAN 90/) e gera um conjunto de arquivos compiláveis. Cada transição da rede é traduzida para um procedimento de disparo e existe uma classificação de tipos de transições (por necessidade de comunicação, por conflito estrutural, por número de objetos envolvidos, etc) tendo cada tipo um esqueleto de procedimento de disparo correspondente mantido em um banco de procedimentos.

Sob o enfoque da abordagem de implementação transformacional pretende-se, no projeto ora sendo proposto, investir numa solução do tipo: mapear cada construção particular da especificação ER/T (tipos de fluxos, cardinalidade, dependência funcional, etc) em um esqueleto de código fonte (conjunto de comandos) de alguma linguagem alvo que implemente esta construção.

4.2 ABORDAGEM EXECUÇÃO DA ESPECIFICAÇÃO

Inicialmente o enfoque dado para a abordagem de execução da especificação era a de simplesmente realizar uma avaliação prévia do comportamento do sistema a partir da especificação. Publicações como /COH 82/, /CHE 84/, /BAL 82/ e /BAL 85a/ davam respaldo a esta idéia.

Posteriormente direcionou-se os estudos no sentido de utilizar esta abordagem como uma possível alternativa de obter uma implementação da especificação, conforme discutido anteriormente.

Reforçando esta outra forma de implementação vê-se, por exemplo, que em /CAN 90/ o autor propõe uma linguagem de especificação de sistemas baseada nas Redes de Petri a objetos (RPO) incorporando estruturação e modularidade a estas redes mediante construções adaptadas da linguagem ESTELLE. A linguagem (LRPO) descreve uma especificação na forma de uma estrutura hierárquica de módulos cujo corpo é uma RPO (para descrever o

comportamento do módulo). Adotando a abordagem de execução da especificação, a linguagem LRPO é interpretada através de um simulador de RPO escrito em Lisp.

Para a presente abordagem, que se fundamenta em um método interpretativo, o processo de interpretação pode vir a ser facilitado quando da representação de dados e operações em algum formalismo linguístico. Isto porque este formalismo poderia ser submetido a algum interpretador de linguagem já disponível. Cabe, portanto, averiguar algumas alternativas de representar, na forma textual, os diagramas do modelo ER/T.

Como representar o modelo ER/T:

Como o modelo ER/T tem sua base formal nas Redes de Petri, muitas das investigações feitas sobre as Redes podem também ter aplicabilidade no modelo ER/T. Note-se que as Redes de Petri clássicas não são adequadas à modelagem de dados. A abordagem ER/T procura suprir esta deficiência no sentido de incorporar às Redes uma modelagem de dados através de diagramas Entidade-Relacionamento(E/R).

Um trabalho correlato aparece em /GAR 89/ e /GAR 90/ onde é proposto um simulador (denominado PENELOPE) de Redes de Petri baseadas em objetos (PNO: Petri Net with Objects) para sistemas de manufatura. No modelo PNO, as restrições das redes clássicas quanto à estrutura de dados foram superadas pela utilização dos conceitos do paradigma de orientação a objetos.

Em /BAL 91/ também é proposta a simulação de Redes de Petri que incorporam conceitos da abordagem de orientação a objetos. Apresenta um formalismo gráfico (PROTOB) que integra fluxo de dados (para relação entre objetos), Redes de Petri (para comportamento do objeto) e linguagens de programação de alto nível (para descrição dos objetos em um arquivo texto em C ou ADA).

Innocenti /INN 90/ propõe um formalismo para especificações executáveis (RSF) e um mecanismo de execução simbólica para avaliar o comportamento do sistema. As especificações são feitas na forma de regras de transição (FROM <estados> WITH <condições de disparo> CONS <eventos a consumir> PRODUCE <eventos a produzir>).

Em /BAK 90/, /GAR 89/ e /GAR 90/ é enfatizado que as Redes de Petri tradicionais (predicado-transição) podem facilmente ser traduzidas para um sistema de regras em lógica proposicional em vista de uma grande similaridade entre os modelos (estilos declarativos tipo causa/efeito). Nesta tradução associa-se uma regra a cada transição (lugares de entrada = pré-condição; lugares de saída = pós-condição). Em /CAN 90/ as similaridades entre Redes de Petri e regras de produção também são destacadas (paradigmas equivalentes).

Portanto, para o presente projeto, uma alternativa de mapeamento do modelo ER/T para uma forma textual poderia encontrar acolhida, por exemplo, nas regras de produção da lógica proposicional.

4.3 ÁREAS DA INFORMÁTICA NO APOIO À SOLUÇÃO

A) INTELIGÊNCIA ARTIFICIAL:

A tarefa de programação convencional pressupõe a aplicação de um conjunto de técnicas de programação para traduzir (normalmente de forma manual) uma especificação em uma correspondente linguagem de programação. Quando tenciona-se automatizar esta tarefa, o acervo de conhecimentos necessários sobre a execução da tarefa deve ser incorporado ao sistema que a simulará. Neste contexto, os sistemas especialistas e técnicas de inteligência artificial podem ser uma alternativa interessante.

Em /AHL 91b/, por exemplo, constatou-se que a tecnologia da área de inteligência artificial pode ter um campo promissor quando aplicada a ferramentas da engenharia de software (formalizações, programação em lógica como instrumento de

prototipação, modelos baseados em conhecimento (fatos & regras), etc.). Neste mesmo estudo, analisando as características dos principais paradigmas de programação, observou-se que os mais indicados para o desenvolvimento de sistemas especialistas estão associados às linguagens funcionais (ex.: Lisp (puro), ML, Miranda) e às linguagens de programação em lógica (ex.: Planner, Prolog). Justifica-se em vista de ambas representarem o conhecimento em uma notação direta (Lógicas: orientadas a fatos e regras; Funcionais: orientadas a funções). Dados e procedimentos são tratados de uma forma homogênea facilitando, assim, a representação do conhecimento sobre tarefas humanas que o sistema especialista deve simular via computador.

Em /INN 90/ é salientado que as linguagens declarativas (funcionais e lógicas) são próprias para uma rápida prototipação pela simplicidade e organização no controle da informação. Próprias, portanto, para uma abordagem de execução da especificação.

Convém avaliar também a idéia de combinar as facilidades de linguagens funcionais com as das linguagens em lógica (/BEL 86/,/DEG 86/,/RED 86/,/RED 87/,/RED 89/). Em linguagens funcionais existe, por exemplo, uma uniformidade na manipulação de dados e procedimentos e onde o método (regra) de funcionamento de uma função é descrito através do uso de outras funções (funções descrevendo funções). As expressões em linguagens funcionais estão livres de efeitos colaterais (não há atribuição a variáveis). Já as linguagens lógicas proporcionam facilidades como retrocesso (backtracking) na avaliação de regras (permite estabelecimento de caminhos alternativos) não presente em linguagens funcionais /PIN 88/ e também proporcionam um mecanismo de unificação.

B) BANCO DE DADOS:

Tendo em vista que o modelo ER/T tem sua base formal nas Redes de Petri, características como concorrência, sincronização, não determinismo e restrições de tempo /GAR 89/ estão presentes neste modelo.

Na área de banco de dados, mecanismos que implementam concorrência e sincronização já estão bastante difundidos e sedimentados. Em um SGBD, a sequência de bloqueio de transações (locks), por exemplo, normalmente é submetida a rotinas de detecção e prevenção de "dead-locks". Portanto, além de conhecimento sobre os mecanismos de gerenciamento das estruturas de dados, um acervo de conhecimentos sobre gerenciamento de transações pode ser buscado desta área de informática.

Da área de banco de dados tornam-se necessários, também, conhecimentos sobre os modelos de dados capazes de suportar um ambiente no qual a especificação de programas/sistemas constitui o conjunto de dados a ser manipulado pelo SGBD. Neste sentido, pesquisas como as de Kerschberg /KER 90/ e Dayal /DAY 88/ sobre banco de dados baseados em conhecimentos (regras, gatilhos) podem, também, trazer grandes contribuições.

Percebe-se, aí, que uma das tendências é combinar conhecimentos advindos das áreas de inteligência artificial e banco de dados, além do acervo sobre técnicas de programação e projeto herdado da área de engenharia de software.

4.4 QUADRO SUCINTO DE ALTERNATIVAS DE SOLUÇÃO

Para sumarizar as alternativas de implementação de especificações aqui discutidas, o quadro abaixo relaciona as opções as serem investigadas:

IMPLEMENTAÇÃO DE ESPECIFICAÇÕES:

* Método interpretativo
  - Repres. via Regras de Produção
  - Repres. via Ling. Funcionais
  - Combinação Lógica x Funcional
* Método compilativo (tradução/transformação)
* Método híbrido (interpretativo + compilativo)

5. EXEMPLO DE MAPEAMENTO DO MODELO ER/T

Conforme visto no capítulo anterior, o diagrama ER/T pode originar uma implementação usando algum método compilativo, interpretativo ou, eventualmente, lançando mão de alguma forma híbrida que se utilize de características de ambos os métodos.

Antes de aplicar um dos métodos, a representação diagramática do modelo ER/T deve ser mapeada para alguma representação linguística.

Para exemplificar a representação do modelo ER/T na forma textual, são discutidas no presente capítulo as alternativas de mapeamento do diagrama que modela a transação de reserva (figura 3) para uma linguagem procedural e para uma linguagem declarativa.

5.1 PARA UMA REPRES. LINGUÍSTICA PROCEDURAL(IMPERATIVA)

A descrição dos procedimentos do exemplo da figura 3 ("transação de reserva") não está, aqui, considerando nenhuma linguagem específica de programação. Os procedimentos são descritos na forma de um algoritmo escrito em "português estruturado" (pseudo-código).

Assim, com base na aplicação das regras de habilitação de uma transformação sumarizadas na seção 2.2.1., a transação de reserva pode considerar uma sequência arbitrária de procedimentos como segue:

PRÉ-CONDIÇÃO: (Causa)

* código_sócio e código_volume informados pelo SÓCIO (REGRA 3);
* teste de presença do sócio (REGRA 1);
* teste de presença do volume (REGRA 1);
* teste de presença do <outro_sócio,volume> em EMPRÉSTIMO (REGRA 1);
* teste de ausência do <sócio,volume> em RESERVA (REGRA 2) (para o fluxo de inclusão é feita a substituição por uma operação que representa o teste de ausência);
* cardinalidade não esgotada para nova inclusão (REGRA 4);

* teste de restrições das anotações:(REGRA 5)
  - sócio.cod = código_sócio e
  - volume.cod= código_volume

CORPO DA TRANSAÇÃO:
* atribuições diversas:
  - data_devolução = <outro_sócio,volume>.data_dev

PÓS-CONDIÇÃO: (Efeito)
* inclusão bem sucedida;
* saída da informação sobre data_devolução bem sucedida.

Obs.: (REGRA n) seguindo o procedimento, referencia as regras de habilitação enumeradas na seção 2.2.1 .

Avaliando esta sequência de operações que implementa a transação de reserva, pode-se generalizar como uma possível Regra de Sequencialização de Transações do Modelo ER/T :

```
1.- apropriação de todas as entradas de dados;
2.- testes de ausência/presença de entidades que dependam
    exclusivamente dos dados(atributos) de entrada;
3.- laço para avaliação exaustiva dos testes de ausência/
    presença dependentes de dados obtidos de outros testes
    (GRAFO de DEPENDÊNCIA). Implementável, por exemplo,
    via uma lista circular:
        3.1 - Se dados suficientes para o teste:
                3.1.1 - executa teste;
                3.1.2 - elimina da lista circular;
                3.1.3 - assinala recuperação do atributo
                        para liberar outros testes;
        3.2 - Passa para próximo da lista;
4.- teste de seleção com base nas anotações;
5.- teste de cardinalidade;
6.- atribuições do corpo da transação;
7.- processa exclusões;
8.- processa inclusões;
9.- processa alterações;
10.- liberação dos resultados de saída.

OBS.: Os testes de presença/ausência incluem também os
      testes para as operações de exclusão/alteração e
      inclusões.
```

Uma outra possibilidade de sequencialização das operações de uma transação ER/T é a de permitir uma priorização dos fluxos por parte do usuário, ao estilo da solução dada para os Diagramas de Navegação, apresentada em /MAR 91/. Durante a modelagem da transação é indicada, via tela específica, a sequência das operações. Neste caso, deverá ser consistida a viabilidade da sequência informada (dados disponíveis para completar a sequência, "dead-locks", etc).

Concluída a tradução do diagrama para uma linguagem procedural (alguma linguagem de programação convencional, por exemplo), o fonte desta linguagem pode então ser submetido ao processo de compilação para a geração de código executável.

5.2 PARA UMA REPRES. LINGUÍSTICA DECLARATIVA(ASSERCIONAL)

As linguagens declarativas representam o conhecimento sobre o programa/sistema em uma notação bastante simples (não algoritmica), preocupando-se somente em indicar quais propriedades o programa/sistema deve prover (resultados esperados), em contraste com as linguagens imperativas que manipulam informações de controle que detalham procedimentos de como chegar a estas propriedades. Como consequência, são indicadas para uma prototipação, já que as especificações feitas através destas linguagens declarativas podem ser executadas, e o seu comportamento validado, segundo um ambiente de desenvolvimento e depuração mais propício e amigável, em vista de: a)dados e procedimentos serem tratados de forma homogênea; b)o projetista se limitar a especificar o que o programa deve fazer e não como deverá fazê-lo.

Podem, portanto, ser consideradas como uma alternativa de mapeamento intermediário para o caso de uma implementação segundo a abordagem transformacional (tradução) ou como linguagem de simulação/prototipação para o caso de uma abordagem de execução da especificação (interpretação).

Para ilustrar como o modelo ER/T pode ser mapeado para uma linguagem declarativa, o exemplo da figura 3 poderia ser especificado da seguinte forma:

Exemplo: Usando uma LINGUAGEM LÓGICA (na forma de Regras de Produção):

SE
código_sócio , código_volume estiverem em externo sócio;
código_sócio estiver em entidade sócio;
código_volume estiver em entidade volume;
código_sócio for diferente de outro_sócio;
outro_sócio , código_volume estiverem em relacionamento empréstimo;
código_sócio , código_volume não estiverem em relacionamento reserva;
contagem(código_sócio em reserva)+ 1 não for maior que limite(código_sócio em reserva);

contagem(código_volume em reserva)+ 1 não for maior que limite(código_volume em reserva);

ENTÃO
data_devolução é igualado a data_dev de outro_sócio , código_volume em empréstimo;
código_sócio , código_volume vão para relacionamento reserva;
data_devolução vai para externo sócio

Note que, como até o presente momento nenhuma linguagem específica foi eleita como linguagem lógica de representação do modelo ER/T, as regras estão especificadas na forma de sentenças em português.

A especificação ER/T, uma vez traduzida para regras de produção, poderia ser executada diretamente usando um interpretador existente de alguma linguagem como Prolog ou OSP5. Outra alternativa seria a de construir um interpretador específico onde, para a sua confecção, linguagens como Lisp ou ML (funcionais) podem vir a ser uma opção interessante.

6. AVALIAÇÃO DE MÉTODOS EXISTENTES DE IMPLEMENTAÇÃO DE REDES DE PETRI

Bibliografia recente sobre a implementação de redes de Petri foi avaliada a fim de determinar as características incorporadas em cada um dos métodos pesquisados para, a partir delas, estabelecer as similaridades e as peculiaridades com o método alvo da presente proposta de tese.

Uma sinopse para cada um dos principais métodos analisados está, a seguir, relacionada.

Método: Rede como regras de produção (/CAN 90/,/GAR 90/)

Forma de Implementação: Interpretativa

Tipo de rede: RPO (Rede de Petri a Objetos)

Características básicas:

DO MODELO:
- Linguagem de especificação (LRPO) baseada em rede de Petri a objeto (rede de Petri + formalizações da estrutura de dados segundo paradigma de orientação a objetos) acrescida de uma estrutura hierárquica de módulos adaptada a partir da ling. Estelle. O comportamento de cada módulo é descrito por uma RPO;

DA IMPLEMENTAÇÃO:
- A rede é tratada como um sistema de regras de produção em vista de similaridades entre os modelos (ambos com pré-condições + ações associadas);
- O interpretador atua como motor de inferência (filtragem: seleção de regras aplicáveis; resolução de conflitos: escolha de regra para disparo; ação: execução da ação (no caso, atribuição de valores a variáveis) associada à regra), executando a evolução dos estados;
- Mecanismo de otimização do processo de filtragem das regras (testes de pré-condição) baseado em compilação das regras para evitar redundância nos testes;
- Implementação não admite concorrência.

Aplicabilidade:

- Sistemas de controle de processo industriais

Método: Implementação rede PROT (/BRU 86/)

Forma de Implementação: Tradutiva

Tipo de rede: PROT (PROcess Traslatable)

Características básicas:

DO MODELO:
- A rede Prot (Rede Traduzível em Processos) modela as interações entre instâncias de diferentes tipos de processos;
- Utiliza uma rede tipo Predicado-Transição (PrT) onde os lugares possuem fichas representando os processos a serem modelados;

DA IMPLEMENTAÇÃO:
- A rede é traduzida para a linguagem ADA (transições e fichas (processos) são implementadas como "TASK" de ADA utilizando esqueletos de procedimentos previamente escritos nesta linguagem);
- Uma "TASK" de processo, quando necessita disparar uma transição, envia dados para a "TASK" de transição e fica aguardando uma mensagem que conclui o disparo ("TASK" ADA trocando mensagens);
- Implementação admite concorrência e sincronização.

Aplicabilidade:

- Sistemas de controle de processo industriais
- Protótipos de sistemas de simulação

Método: Implementação modelo RPO (/SIL 90/)

Forma de Implementação: Tradutiva

Tipo de rede: RPO (Rede de Petri a Objetos)

Características básicas:

DO MODELO:
- Na rede RPO, os dados são tratados como fichas (tokens) representando instâncias de classes de objetos;

DA IMPLEMENTAÇÃO:
- Implementa a rede RPO através da tradução para correspondentes construções na linguagem de programação LIS (Ling. de Implem. de Sist.) que suporta processamento distribuído e foi desenvolvida no LCMI/UFSC. O programa LIS é construído a partir de esqueletos de código que mapeiam os componentes da rede;
- O tradutor, escrito em Pascal, recebe a descrição RPO já tratada (na linguagem LRPO /CAN 90/) e gera um conjunto de arquivos compiláveis;
- Cada transição da rede é traduzida para um procedimento de disparo (conjunto de comandos da linguagem alvo);
- Existe uma classificação dos tipos de transições (por:necessidade de comunicação;conflito estrutural;número de objetos envolvidos) tendo, cada tipo, um esqueleto de procedimento de disparo correspondente mantido em um banco de procedimentos.

Aplicabilidade:

- Sistemas de controle de processo industriais
- Sistemas de processamento distribuído em tempo real

Método: Implementação modelo PROTOB (/BAL 91/)

Forma de Implementação: Interpretativa/Tradutiva

Tipo de rede: PROT + Grafo de fluxo

Características básicas:

DO MODELO:
- O modelo PROTOB (Metodologia + ambiente CASE) segue o paradigma de orientação a objetos e utiliza uma representação gráfica (Grafo:dados,controles,atividades,objetos,troca de mensagens) e uma semântica procedural definida na forma textual (arquivo Script: definições em C ou ADA, declarações de "procedures"e funções externas);
- O comportamento de cada objeto é modelado na rede PROT;

DA IMPLEMENTAÇÃO:
- As redes PROT são diretamente executadas por um mecanismo de simulação que possui um motor de inferência que seleciona a transição a ser disparada, resolve conflitos entre transições e seleciona uma tupla de tokens que torna o predicado verdadeiro;
- O comportamento do simulador é similar ao ciclo de reconhecimento de um interpretador de regras de produção (Rede PROT como regra de produção na linguagem OPS5);
- Um tradutor converte o modelo PROTOB em programa executável na linguagem alvo (C ou ADA);
- Implementação admite concorrência.

Aplicabilidade:

- Sistemas de monitoração de processos;
- Protocolos de comunicação.

Método: Implementação rede PPN (/LI 89/)

Forma de Implementação: Interpretativa

Tipo de rede: PPN (Rede de Petri Procedurais)

Características básicas:

DO MODELO:
- A rede PPN é uma extensão da rede de Petri clássica, acrescentando uma estrutura de dados sem individualização das fichas, mecanismos de interconexão de redes e funções para operar sobre os dados;

DA IMPLEMENTAÇÃO:
- A implementação se baseia em: especificação PPN + algoritmo de gerenciamento de disparos (interpretador);
- O interpretador é um algoritmo pré-elaborado, escrito em C, responsável pela busca e seleção da transição a ser disparada em uma estratégia de busca circular de transições habilitadas, por ordem de declaração;
- A combinação do interpretador com a especificação gera um programa executável em C (compilável);
- Implementação não admite concorrência;

Aplicabilidade:

- Protocolos de comunicação

Método: Implementação rede clássica (/NEL 83/)

Forma de Implementação: Tradutiva

Tipo de rede: Rede de Petri clássica (lugar/transição)

Características básicas:

DO MODELO:
- Utiliza a Rede de Petri clássica sem associação de mecanismos especiais de representação de estrutura de dados (representação de dados muito limitada). Acrescenta transições de iniciálização e finalização sem lugares de entrada e saída associados;

DA IMPLEMENTAÇÃO:
- A semântica da rede é expressa através da sintaxe da linguagem procedural XL/1 (uma linguagem concorrente) numa primeira etapa do processo de tradução. Numa segunda etapa é feito um mapeamento da linguagem XL/1 para correspondentes construções na linguagem PL/1 (caso dos programas sem concorrência) e pseudo-PL/1 (para os programas concorrentes);
- Para cada nó da rede (lugar, transição) é selecionado um bloco de código XL/1 pré-definido acrescentando-lhe informações adicionais, peculiares ao lugar/transição;
- O mecanismo de implementação de lugares realiza um loop de testes das transições de saída e remete a ficha para a primeira dentre as transições habilitadas. O mecanismo de implementação das transições transfere fichas dos lugares de entrada para os de saída;
- Implementação da especificação admite concorrência.

Aplicabilidade:

- Muito limitado pela falta de estruturas de dados adequadas.
- Simulações (ex.: autômatos, controle de processos) via Redes de Petri clássicas.

Método: Implementação modelo ESDM (/CHA 91/)

Forma de Implementação: Tradutiva

Tipo de rede: ESDM (Executable Semantic Data Model)

Características básicas:

DO MODELO:
- O modelo ESDM permite modelagem dos aspectos estruturais e aspectos comportamentais de um banco de dados;
- Para modelagem estrutural, utiliza uma extensão do modelo ER, representando entidades, relacionamentos, atributos e hierarquia de subset (visando generalização/especialização);
- Para modelagem comportamental, utiliza Redes de Petri (tipo lugar/transição). Eventos do banco de dados são modelados como transições contendo parâmetros, pré-condições e pós-condições. A execução de uma transição pode ocasionar mudanças nos dados do banco de dados e/ou provocar o disparo de outras transições;
- Para modelagem conceitual completa, a integração entre o submodelo estrutural (ER) e o submodelo comportamental (Rede de Petri) é feita mediante lugares (círculo com rótulo), ou seja, lugares são interfaces entre transições e também entre a rede e o submodelo de dados (conectores de submodelos). Visualmente esta integração entre modelos (dados & funções) fica um pouco prejudicada;

DA IMPLEMENTAÇÃO:
- A implementação do modelo ESDM considera um mapeamento para INGRES. O submodelo estrutural é traduzido para comandos CREATE da linguagem QUEL (ling. de consulta do Ingres). O submodelo comportamental é traduzido para um programa EQUEL (no caso, embutimento de comandos QUEL em programa C);

Aplicabilidade:

- Sistemas de informações em banco de dados

7. AS SOLUÇÕES ATUAIS FRENTE NOSSAS NECESSIDADES

Considerando os quesitos, anteriormente enumerados na seção 2.2, desejáveis para a linguagem de especificação a ser utilizada como modelo de partida para o presente trabalho, enfatizando também o aspecto da NÃO PROCEDURALIDADE NA ESPECIFICAÇÃO e das aplicações em SISTEMAS TRANSACIONAIS DE BANCO DE DADOS como temas básicos das investigações desta proposta, justificou-se o uso da abordagem ER/T como modelo que atenderia a estes propósitos (seção 2.2.3) e verificou-se que o modelo ER/T enquadra-se perfeitamente em uma classe de TDF bastante discutida e aceita atualmente na literatura (seção 2.2.2).

Eventuais substitutos do modelo ER/T para a modelagem dos sistemas, segundo os critérios aqui estabelecidos, devem incorporar características como:

- ter representação gráfica;
- ser não procedural;
- permitir modelar dados e funções de forma homogênea;
- ser independente de implementação;
- ter base formal;
- permitir especificar transações em banco de dados;
- ter, embutidos no diagrama, aspectos como:
  - .manipulação de dados (inclusão,alteração,exclusão e consulta);
  - .pré e pós condições para transição;
  - .tratamento de inconsistências;
  - .tratamento de E/S.

Uma vez caracterizada a linguagem de especificação a ser utilizada para a modelagem do sistema, cabe frisar os requisitos a serem considerados no método de implementação da especificação. São eles:

- Mecanismos de sequencialização de operações para tratar a não procedüralidade da especificação;
- Mecanismos de implementação dos aspectos estruturais do banco de dados (hierarquia entre dados, relacionamento entre dados, etc);
- Mecanismos de implementação dos aspectos comportamentais do banco de dados (tratamento de entrada e saída, operações básicas de acesso e atualização, etc.);
- Mecanismos de implementação de concorrência e sincronização (com bloqueio de transações);
- Mecanismos de implementação de restrições de integridade estáticas (chave primária, cardinalidade, exclusão mútua, dependência funcional) com correspondente tratamento de exceções;

Reafirmando que os objetivos básicos do trabalho aqui sendo proposto se apoiam em premissas como:

- atender as peculiaridades dos sistemas de informação em banco de dados, como:
  - . representação dos dados;
  - . manipulação dos dados (consulta/atualização);
  - . gerenciamento de transações;
  - . gerenciamento de concorrência;
- usar um modelo não procedural para modelagem do sistema;
- tirar proveito da característica de causa/efeito da classe de TDF, anteriormente discutida, para modelar transações em banco de dados;
- incorporar um alto grau de conhecimento nos mecanismos de tradução para resolver problemas como tratamento de restrições de integridade, acessos ao B.D e tratamento de exceções;

justifica-se o porquê das soluções existentes não se aplicarem adequadamente ao problema da tese, pelos seguintes motivos:

a) Rede RPO, PROT e PROTOB:

- o mecanismo de funcionamento das redes se baseia, exclusivamente, em regras de habilitação clássicas (marcas de entrada presentes, marcas de saída ausentes e fórmulas de conexão assumindo condição verdadeira) quando a modelagem conceitual de banco de dados envolve também considerar restrições de integridade nas regras de habilitação de uma transação;
- a estrutura de dados (tokens) utilizada nas redes é inadequada para uma modelagem conceitual de banco de dados (com entidades,relacionamentos e atributos); o relacionamento entre os dados e as restrições de integridade estáticas não são passíveis de serem explicitadas no modelo;

b) Redes de Petri clássicas (lugar/transição):

- o nível da rede é muito elementar para a aplicação em sistemas de informações em banco de dados, principalmente em vista das limitações quanto à representação da estrutura de dados (não tem objetos com atributos) e a represetação das formas de acesso aos dados.

c) Rede PPN:

- possui uma representação de dados inadequada;
- é uma rede muito procedural.

d) Rede ESDM:

- dentre os modelos e implementações estudados é o que mais se aproxima dos propósitos do problema da tese;
- a não proceduralidade na especificação, no entanto, pode ser questionada tendo em vista que grande parte dos procedimentos executados numa transição, e a sequência das operações nestes procedimentos, estão explicitados na linguagem de anotação da transição exigindo pouca heurística (conhecimento dedutivo) dos algoritmos de tradução; constata-se, por exemplo, que:
  - *existe uma equivalência no nível das linguagens de modelagem (anotação da transição) e de implementação;
  - *a entrada de dados é indicada textualmente nos parâmetros da transição e mapeados diretamente para comandos de entrada da linguagem de implementação (é impressa uma mensagem solicitando os dados e, em seguida, é aguardado o seu o fornecimento);
  - *a sequência arbitrária de procedimentos escolhida para modelar a transição é a mesma sequência que será utilizada na implementação desta transição (a tradução não considera aspectos como otimizações ou sequências alternativas); por exemplo, na pré-condição é indicada, como seleção, a sequência para recuperação das tuplas (ou para exclusão, se transição tem lugar de consumo) e, na pós-condição, a sequência de valoração (atribuição) dos atributos para as operações de inclusão e alteração de tuplas;
- o mecanismo de habilitação de transições se dá, exclusivamente, pela presença de tuplas nos lugares de entrada e pela pré-condição satisfeita; não considera lugares de saída (inexistência da tupla como requisito de habilitação) nem tão pouco restrições de integridade estáticas, como dependência funcional de atributos, cardinalidade do relacionamento, dependência de domínio, etc.; o método de implementação pressupõe um tratamento das restrições de integridade sendo realizadas totalmente ao encargo do SGBD; é uma característica que eventualmente poderia ser incorporada no método com relativa facilidade;
- existe dificuldade em delinear transações sobre o banco de dados, ou seja, determinar quais os dados e quais os acessos necessários em uma operação atômica sobre o banco de dados; o método de implementação não prevê mecanismos de gerenciamento de transações;
- o método de implementação utilizado também não prevê mecanismos de tratamento de exceções.

O conjunto de limitações constatadas nas atuais soluções, aqui enunciadas, frente ao problema exposto, motivam o desenvolvimento de um método de implementação de especificações que incorpore, de forma efetiva, os requisitos desejáveis para o modelo de implementação em questão.

8. PROPOSTA DE TESE E CONTRIBUIÇÕES

Durante as explanações feitas neste texto, procurou-se defender a idéia de que a implementação de especificações não procedurais, com o mínimo de intervenção do projetista, visando automatizar todo o processo e assim otimizar a produção de software, é um tema que merece investigação. Direcionando o horizonte de pesquisa sobre sistemas de informações, delimitou-se o escopo de aplicações ao nível dos sistemas transacionais em banco de dados.

Sob este prisma, o objetivo básico da tese passa a ser a obtenção de um método completo de implementação de especificações, segundo o problema enunciado nesta proposta, tendo os métodos tradutivos e interpretativos, analisados na literatura, como ponto de partida das investigações. Como peculiaridade principal deste método, espera-se enfatizar os aspectos que ainda não obtiveram uma solução proposta em métodos anteriormente estudados. Assim, como pretensões prioritárias das investigações da tese, tenciona-se realizar experimentos que se situam no entrelaçamento de 3 (três) temas básicos:

- Mecanismos de sequencialização das operações (proceduralização da especificação);
- Mecanismos de gerenciamento de transações em ambientes concorrentes, visivelmente críticos em caso de bloqueio de um conjunto de tuplas/registros;
- Mecanismos de tratamento de exceções;

As implementações previstas se situam no âmbito do módulo de tradução que irá considerar o modelo ER/T numa representação interna (não diagramática) e também algum SGBD existente (por exemplo, INGRES). Abstraem-se, destas implementações, os detalhes de diagramação do modelo ER/T (ferramenta CASE) e os problemas de comunicação com agentes externos (interfaces). A eventual implementação de alguma rotina ou protótipo, durante os trabalhos da tese, se limitará a servir de instrumento para experimentar certas idéias e não como resultado prático final (produto) oriundo da pesquisa.

Como contribuições da tese, espera-se dar uma solução para os 3 temas básicos acima mencionados e obter a especificação completa do método com o devido detalhamento das técnicas usadas para mapear as diferentes construções da linguagem de especificação não procedural em uma representação que permita a implementação/execução direta.

Dentro das perspectivas de trabalhos futuros, a completa especificação do método servirá como ponto de partida para propiciar a futura construção de ferramentas que o implementem, indo de encontro aos objetivos propostos no projeto d&f (data & function) em andamento no CPGCC-UFRGS.

9. A PROPOSIÇÃO DE UMA ARQUITETURA PARA O AMBIENTE

Visando estabelecer um ambiente de programação automática que sintonize com a proposta da tese, foi sugerida uma arquitetura de um sistema de construção de software conforme mostra a figura 5.

Nesta arquitetura, a fase de aquisição de especificação é representada por um módulo de modelagem conceitual onde são descritas/desenhadas as propriedades estáticas e dinâmicas do sistema. Para esta modelagem, uma alternativa interessante pode estar em um **editor gráfico inteligente** que, além de orientar o usuário em determinadas fases da modelagem (condutor de modelagem: como modelar! o que está faltando!), pode gerar automaticamente algumas especificações (desenho de componentes dinâmicos do sistema) como, por exemplo , os procedimentos para as operações básicas de atualização. (Nota: para fins da presente proposta abstrai-se, aqui, os detalhes funcionais deste módulo).

O módulo de execução da especificação é o responsável pela simulação do modelo conceitual a fim de validar o seu comportamento. Este módulo permite obter uma rápida prototipação para avaliar as reais condições de execução do sistema proposto. Permite que o usuário final possa ter uma idéia de como será o ambiente operacional do futuro sistema e serve, também, aos propósitos do projetista como instrumento de depuração do modelo conceitual onde, através de uma opção de execução passo-a-passo, é possível realizar um acompanhamento do estado da máquina (variáveis, interfaces, etc.).

O módulo de tradução representa o processo de mapeamento de um modelo abstrato para um modelo implementável. Este módulo é o responsável pela tradução da especificação original para uma linguagem que incorpore os detalhes de como implementar o sistema em um determinado ambiente computacional. Associado ao processo de transformação de especificação seria interessante manter um histórico de derivações a partir da sequência de transformações aplicadas. Com isso, permite-se que uma nova implementação seja rederivada de uma especificação

modificada e somente os procedimentos envolvidos na alteração sejam reeditados, mantendo os demais inalterados. Isto implicaria na incorporação de um mecanismo de gerenciamento de versões (tema ainda em estudo). A tradução é feita de forma interativa a fim de que o projetista indique o ambiente operacional a ser gerado e alimente o sistema com decisões sobre otimizações e racionalizações do processo de transformação para aqueles detalhes que os algoritmos de refinamento não possam ser autosuficientes.

Uma vez obtida a base de especificação transformada, o processo de compilação é realizado automaticamente, gerando a aplicação (Banco de Dados + Software) em um ambiente específico. As informações sobre este ambiente são obtidos de "driver's" que descrevem as características do equipamento a ser usado como destino da aplicação.

Cabe frisar que, nesta arquitetura, a figura do usuário, frente à interface, é representada pelo projetista de sistemas.

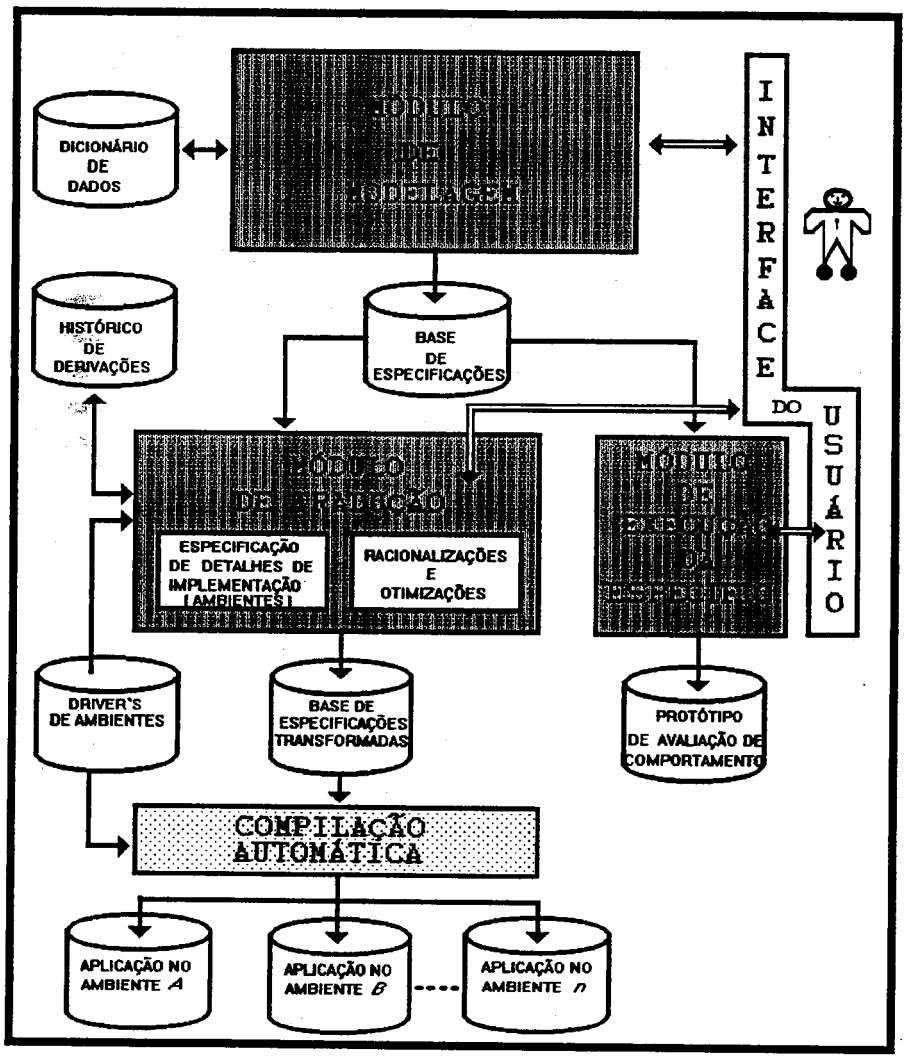


Fig. 5 - Arquitetura de um sistema de construção automática de software

REFERÊNCIAS BIBLIOGRÁFICAS

/AHL 90a/ AHLERT, H.: Estudo de metodologias e ferramentas que envolvem suporte à construção automática de sistemas, TI. n. 188 - CPGCC/UFRGS, mai 1990.

/AHL 91a/ AHLERT, H.: Estudo comparativo e taxonomia de ferramentas de suporte à construção automática de sistemas, RP. n. 153 - CPGCC/UFRGS, abr 1991.

/AHL 91b/ AHLERT, H.: Sistemas especialistas para a engenharia de software, RP. n. 154 - CPGCC/UFRGS, abr 1991.

/ALE 89/ ALENCAR, P.S.C., LUCENA, C.J.P.: Métodos formais para o desenvolvimento de programas, EBAI, jan 1989.

/BAK 90/ BAKO, B., VALETTE, R., CARDOSO, J.: Implementação de um sistema de regras de produção controlado para a aplicação na supervisão de sistemas flexíveis de manufatura (on the implementation of controlled rule-based systems for supervising FMS). 4.Congr. nacional de automação industrial, São Paulo 23-27 julho 1990.

/BAL 81/ BALZER, R.: Transformational implementation: An example, IEEE Trans. Software Eng., vol. SE-7, pp. 3-14, jan. 1981.

/BAL 82/ BALZER, R., GOLDMAN, N., WILE, D.: Operational specification as the basis for rapid prototyping, ACM Sigsoft Software Engineering Notes, vol. 7, no. 5, pp. 3-16 dec. 1982

/BAL 83a/ BALZER, R., CHEATHAM, T.E. Jr., GREEN, C.: Software technology in the 1990's using a new paradigm. Computer, vol 16, no. 11, pp. 39-45 (1983)

/BAL 85a/ BALZER, R.: A 15 years perspective in automatic programming, IEEE Trans. Software Eng., vol. 11, no. 11, pp 1257-1267 (1985)

/BAL 91/ BALDASSARI, M., BRUNO, G., CASTELLA, A.: "PROTOB": an object-oriented CASE tool for modelling and prototyping distributed systems, Software-Practice and Experience, vol. 21(8): 823-844, aug. 1991.

/BAR 77/ BARSTOW, D.: A knowledge-based system for automatic program construction, in: Proc. 5th Int. Join Conf. Artificial Intelligence, cambridge, MA, aug. 1977, pp. 382-388.

/BAU 76/ BAUER, F.L.: Programming as an evolutionary process, em Bauer, F.L. e Samelson, K. (Eds), Languages Hierarchies and Interfaces, pp. 153-182, Springer-verlag, Berlin, 1976.

/BAU 87/ BAUER, F.L., et alii: The Munich project CIP, vol II: The program transformation systems CIP-S, Lecture Notes in Computer Science 252, Springer, Berlin, 1987

/BAU 89/ BAUER, F.L., et alii: Formal program construction by transformations - Computer-aided, intuition-guided programming, IEEE Trans. on Soft. Engineering, vol. 15, no 2, pp. 165-180, feb. 1989.

/BEL 86/ BELLIA, M., LEVI, G.:The relation between logic and functional languages: a survey, in: The Journal of Logic programming, 3: 217-236 (1986)

/BOR 84/ BORGIDA, A.,MYLOPOULOS, J.,WONG, H.K.T.: Generalization/specialization as a basis for software specification.IN: On Conceptual Modelling. Perspectives from Artificial Intelligence, Database, and Programming Language, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, Eds Springer-Verlag, New York, 1984, pp. 87-114.

/BRO 81/ BRODIE, M.L.: On modelling behavioural semantics of databases, 7th VLDB conf., Cannes, France, 1981

/BRU 86/ BRUNO, G., MARCHETTO, G.: Process translatable Petri nets for the rapid prototyping of process control systems, IEEE Transactions on Software Engineering, vol. SE-12, no. 2, 346-357.

/CAN 90/ CANTÚ, E.: Uma abordagem para a representação, simulação e implementação de sistemas baseada na Rede de Petri a Objetos, Florianópolis, SC, 1990, Dissertação de Mestrado.

/CHA 91/ CHAO, C.M., KUNG, C.: Rapid prototyping of conceptual database design on a relational database management system, Proc. 10Th Int. Conf. on the Entity Relationship Approach, 1991.

/CHE 72/ CHEATHAM, T.E., WEGBREIT, B.: A laboratory for the study of automatic programming, in: Proceeding of AFIPS Spring Join Comp. Conf., (Atlantic City, N.J., May 16-18, 1972), vol. 40, AFIPS Press, Reston, VA., pp 11-21.

/CHE 81/ CHEATHAM, T.E., HOLLOWAY, G.H., TOWNLEY, J.A.: Program refinement by transformation, in: Proceedings of 5th International Conference on Software Engineering (San Diego, Calif., mar. 9-12, 1981), IEEE, New York, pp. 430-437.

/CHE 84/ CHEATHAM, T.E.: Reusability through program transformation, IEEE Trans. on Soft. Engineering, vol. SE-10, no. 5, sep. 1984, pp. 589-594.

/COH 82/ COHEN, D., SWARTOUT, W., BALZER, R.: Using symbolic execution to characterize behavior, ACM Sigsoft Software Engineering Notes, pp. 25-32, dec 1982.

/COU 90/ COUSINEAU, G., HUET, G.: The CAML primer, Inria, 1990, (Rapports Techniques, 122).

/DAR 79/ DARLINGTON, J.: Program transformation: an introduction and survey, Comput. Bull., dec. 1979, pp. 22-24.

/DAY 88/ DAYAL, U., BUCHMANN, A.P., MCCARTHY, D.R.: Rules are objects too: a knowledge model for an active object-oriented database system, in: /DIT 88/, 129-143

/DEG 86/ DEGROOT, D., LINDSTROM, G.: Logic programming: functions, relations and equations, Prentice-Hall, Englewood Cliffs, NJ (1986).

/DIJ 68/ DIJKSTRA, E.: A constructive approach to the problem of program correctness, BIT, vol. 8, 1968.

/DIJ 75/ DIJKSTRA, E.: Guarded commands, non determinancy and formal derivation of programs, Comm. ACM, vol. 18, no. 8, pp. 453-457, 1975.

/DIJ 76/ DIJKSTRA, E.: A discipline of programming, Prentice-Hall, 1976.

/FEA 82a/ FEATHER, M.: Mappings for rapid prototyping, ACM Sigsoft Software Engineering Notes, pp. 17-24, dec 1982.

/FEA 82b/ FEATHER, M.: A system for assisting program transformation, ACM Trans. Program. Lang. Syst., vol. 4, no. 1, 1982

/FIC 85/ FICKAS, S.: Automating the transformational development of software, IEEE Trans. on Soft. Eng., vol. 11, no. 11, pp. 1268-1277, (1985).

/GAR 89/ GARNOUSSET, H.E., et alii: Efficient tools for analysis and implementation of manufacturing systems modelled by Petri net with object: a production rules compilation-based approach, IECON'89, in Proceedings of the 15th Annual Conference of the IEEE Industrial Society, Philadelphia, Pennsylvania, vol. 3: 543-549

/GAR 90/ GARNOUSSET, H.E., et alii: A manufacturing system simulator based on the Petri net with objects mode, European Simulation Conference - ESM, junho 90.

/HAR 86/ HARPER, R., MITCHELL, K.: Introduction to standard ML, Laboratory for foundations of Computer Science, University of Edinburgh (1986).

/HEU 89a/ HEUSER, C.A.: Modelagem conceitual de sistemas, Buenos Aires, Editorial Xapelusz S.A, 1989

/HEU 91/ HEUSER, C.A., PERES, E.M.: ER-T diagrams: an approach to specifying database transactions, Proceedings of the 10th International Conference on the ER approach, San Francisco, oct. 1991

/HIL 90/ HILDUM, D., COHEN, J.: A language for specifying program transformations, IEEE Trans. Soft. Engineering, vol. 16, no. 6, jun 1990.

/HOA 69/ HOARE, C.A.R.: An axiomatic basis for computer programming, Comm. ACM 12(10), pp. 576-580, 1969.

/INN 90/ INNOCENTI, M.D.,et alii: "RSF": a formalism for executable requirement specifications, IEEE Trans. Soft. Engineering, vol. 16, no. 11, nov 1990.

/ISO 86/ ISO DP9074: Estelle: a formal description technique based on extended state transition model, out 1986.

/KAN 81/ KANT, E., BARSTOW, D.R.: The refinement paradigm: the interaction of coding and efficiency knowledge in program synthesis, IEEE Trans. Soft. Eng., vol. SE-7, no. 5, sep. 1981, pp. 458-471.

/KER 90/ KERSCHBERG, L.: Expert database systems: knowledge data management environment for intelligent information system, Information System 15, no. 1, 1990, pp. 151-160.

/LI 89/ LI,P., VON THUM,M., DILLON, T.S.: Semiautomatic implementation of communication protocols from a Petri net base specification language description. Proc. of the 2Th Inter. Conf. on Formal Description Techniques, 1989.

/LIN 86/ LINN,R.J.Jr.: The features and facilities of Estelle. Protocol specification, testing and verification. V editado por M. Diaz, North-Holland, Amsterdam (Holanda), 1986, pp 271-296.

/MAR 91/ MARTIN, J., McCLURE, C.: Técnicas Estruturadas e CASE, tradução de Lúcia Faria Silva, São Paulo, Makron, McGraw-Hill, 1991.

/MEL 89/ MELO, W.L.M.: Uma proposta de um editor diagramático generalizado, Porto Alegre, CPGCC-UFRGS, 1989, Dissertação de Mestrado.

/NEL 83/ NELSON,R.A., HAIBT,L., SHERIDAN, P.D.: Casting Petri nets into programs. IEEE Transactions on Software Engineering, vol. 19, no. 5, 590-602.

/PAI 83/ PAIGE, R.: Transformational programming - aplication to algorithms and systems, in: Proc. of 10th ACM Symp. on Principles of Programming Languages, (Austin, Tex., jan. 24-26, 1983), ACM, New York, pp. 73-87.

/PAR 83/ PARTSCH, H., STEINBRUGGEN, R.: Program transformation system, Comput. Surveys, vol. 15, no. 3, sep. 1983, pp. 199-236.

/PEC 88/ PECKHAM,J.,MARYANSKI,F.: Semantic data models, ACM computing surveys, vol 20, no. 3, sep 1988.

/PER 90/ PERES, E.M.; Abordagem ER/T: uma proposta para a integração da modelagem conceitual de propriedades estáticas e dinâmicas de sistemas de informação, Porto Alegre, CPGCC da UFRGS, dez 1990 (Dissertação de Mestrado)

/PIN 88/ PINGALI, K., EKANADHAM, K.: Accumulators: new logic variable abstractions for functional languages, Ithaca, Cornell University, 1988, (Technical Report 88-952).

/RED 86/ REDDY, U.S.: On the relationship between logic and functions languages, in: Logic Programming: functions, relations and equations, D. Degroot and G. Lindstrom, Eds, Prentice-Hall, 1986, pp. 3-36.

/RED 87/ REDDY, U.S.: Functional logic languages, Part I. in: Graph Meduction, Springer-Verlag, 1987, pp. 401-425 (Lecture Notes in Computer Science, vol 279)

/RED 89/ REDDY, U.S.: Transformational derivation of program using the Focus system, ACM Sigplan Notices, vol. 24, no. 2, feb 1989.

/REG 90/ REGO, A.M.: Uma linguagem gráfica de definição de dados para um modelo E/R estendido, Porto Alegre, CPGCC-UFRGS, 1990, Dissertação de Mestrado.

/SIL 90/ PEREIRA E SILVA, R.: Uma proposta para a implementação de modelos baseados em Rede de Petri a Objetos, Florianópolis, SC, 1990, Dissertação de Mestrado.

/SMI 90/ SMITH, R.D.:"KIDS": a semiautomatic program development system, IEEE Trans. Soft. Engineering, vol 16, no. 9, sep 1990.

/TAN 91/ TANAKA, A.K.,et alii: ER-R: an enhanced ER model with situation-action rules to capture application semantics, Proc. 10Th Int. Conf. on the Entity Relationship Approach, 1991.

/ZAV 91/ ZAVE, P.: An insider's evaluation of PAISLey, IEEE Trans. Soft. Engineering, vol 17, no. 3, mar 1991.

Relatórios de Pesquisa

RP-189: "Análise da adequação dos recursos de paralelismo em algoritmos para resolução de equacões não lineares", junho 1992.
M.A.O.CAMARGO; D.M.CLAUDIO; P.O.NAVAUX

RP-188: "Especificação formal de um sistema de Banco de Dados: um exemplo completo", junho 1992.
J.M.V.CASTILHO; S.LOH; A.R.MARTINI

RP-187: "Leitor CIF em C++versão 1.0: manual do programador, junho 1992.
C.A.R. CRUSIUS

RP-186: "PROJETO CIPREDI: Desenvolvimento da Biblioteca "SEA OF GATES" tecnologia 1.2 um", junho 1992.
B.A. GÜTSCHOW; D.A.C. BARONE

RP-185: "PROJETO SILOG: Ambiente e ferramentas para desenvolvimento automatizado de sistemas de informação com uso de especificações lógicas", junho 1992.
J.M.V. DE CASTILHO; S. LOH

RP-184: "PREVIEW: sistema de animação, junho 1992.
A.E.F. SCHMIDT; S.R. MUSSE

RP-183: "Vinculação de Estímulos do Ambiente de Simulação", junho 1992.
S.R. MUSSE

RP-182: "Rotinas de comunicação para a placa GAPP", junho 1992.
C..A.F. DE ROSE; G.G.H. CAVALHEIRO; R. MENNA BARRETO

RP-181: "Um modelo para representação de atividades em aplicações de escritórios", maio 1992.
D.D.A. RUIZ

RP-180: "O Simulador de Arquiteturas Hipercúbicas iPSC /2", maio de 1992.C.A.F. DE ROSE

RP-179: "GRAFITE: pacote gráfico para ensino de computação gráfica", maio 1992.
L.M. FRANCISCO; S.D. OLABARRIAGA

RP-178: "Comportamento de interfaces inteligentes", maio 1992.
A.S. FRAINER

RP-177: "Divisão e conquista: uma técnica para paralelização de algoritmos", maio 1992.
T.A. DIVÉRIO; D.M. CLAUDIO; P.O.A. NAVAUX

RP-176: "COTONET - Um compilador de netlists formato SPICE", março, 1992.
D.L.N. DE FREITAS; R.A.L. REIS