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

DENISE BANDEIRA DA SILVA

Interface para um Sistema Gerenciador de Transações Longas de Banco de Dados

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Cirano Iochpe
Orientador

Prof. Dr. Clésio Saraiva dos Santos
Co-orientador

Porto Alegre, novembro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silva, Denise Bandeira da

Interface para um Sistema Gerenciador de Transações Longas de Banco de Dados / Denise Bandeira da Silva. – Porto Alegre: Programa de Pós-Graduação em Computação, 2003.

62 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Cirano Iochpe; Co-orientador: Clésio Saraiva dos Santos.

1. Interfaces Gráficas. 2. Linguagens Visuais. 3. Transações Longas. I. Iochpe, Cirano. II. Santos, Clésio Saraiva dos. III. Título.

AGRADECIMENTOS

Agradeço ao Professor Cirano Iochpe pelas idéias inseridas neste trabalho assim como pela sua paciência e sinceridade no momento de nossas reuniões.

Ao professor Clésio Saraiva dos Santos pelos elogios constantes a esta aluna que tanto lhe quer de coração.

Aos meu pais, Mariza e Zaldir, sem o apoio deles, eu jamais teria concluído mais esta etapa de minha formação.

Agradeço ao Amigo Rafael Heitor Bordini a uns amigos muito “Legais” que encontrei aqui no Pós: bordini, soraia, lpn, gersonc, krug e werner. Aos demais colegas e amigos, foi muito bom conviver com vocês.

Aos funcionários do Instituto de Informática, em especial, ao Lulu e à Lulu, à Eli.

Ao CNPq, pelo auxílio financeiro, sem o qual a realização deste trabalho não sería possível.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Existe uma certa gama de aplicações que não pode ser implementada através do modelo convencional de transações, são aplicações que tem um tempo de duração mais longo do que aquelas convencionalmente modeladas. Em uma transação Atômica, ou todo o trabalho é realizado por completo ou nada é feito, mas, quando se trata de atividades de longa duração, isto pode significar a perda de trabalho executado durante horas ou, até mesmo, dias. Pelo mesmo motivo, transações longas não devem executar isoladamente, porque isto impede que outras transações tenham acesso aos dados sendo manipulados.

No âmbito do projeto TRANSCOOP, vêm sendo realizados vários estudos sobre modelos de transações não convencionais. Dentre eles, encontra-se o Modelo de Contratos, que prevê um mecanismo de controle seguro para gerenciar aplicações distribuídas que apresentam atividades de longa duração. Para experimentar e avaliar as idéias inseridas neste modelo está sendo desenvolvido um protótipo. Este sistema é provido de uma interface gráfica interativa, baseada em Manipulação Direta, e suporta a definição de transações longas de banco de dados de acordo com o Modelo de Contratos.

O objetivo deste trabalho é descrever a arquitetura de um protótipo para o Modelo de Contratos, definindo a função de cada um de seus módulos, mais especificamente o módulo Interface, e a comunicação entre eles. Para a definição de uma interface adequada foram considerados aspectos de outras áreas da ciência, pois a área de interfaces homem-máquina é multidisciplinar.

Palavras-chave: Interfaces Gráficas, Linguagens Visuais, Transações Longas.

A long database transaction management system interface

ABSTRACT

Some types of applications cannot be implemented through the conventional transaction model, such applications take a longer time to execute than the conventional ones. In an atomic transaction, or all the work is carried out or nothing is done, but, when long duration transactions are involved, this could mean the loss of work executed during hours or days. For the same reason, long-lived transactions cannot execute in an isolated way, because this does not allow other transactions to have access to the data being manipulated.

This work is part of the TRANSCOOP project, which examines many non-conventional transaction models. One of them, the ConTract Model, provides a control mechanism to manage distributed applications and long-lived transactions in a safe way. A prototype is under development to test and evaluate the ConTract Model ideas. This system provides an interactive graphical interface based on direct manipulation and supports the definition of long-lived database transactions according to the ConTract Model.

The objective of this work is to describe the architecture of a prototype for the ConTract Model. Each one of the prototype modules are described, with especial attention to the Interface and Execution modules and the communication between them. As the interface project is a multidisciplinar field, some aspects of other sciences were studied to provide an interesting interface definition.

Keywords: graphics interface, visual languages, long database transaction.

1 INTRODUÇÃO

Sistemas de Gerência de Banco de Dados (SGBD) baseados no modelo convencional de transações (HAERDER; REUTER, 1983) obtém bom desempenho quando as aplicações modeladas são de curta duração e manipulam um pequeno volume de informações. Tais aplicações aceitam as exigências do modelo tradicional, que enfatiza as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (BERNSTEIN; HADZILACOS; GOODMAN, 1987; AHLERT, 1994).

Entretanto, existe uma certa gama de aplicações computacionais que não pode ser implementada através deste modelo, pois algumas das exigências acima citadas limitam seu uso. Aplicações de sistemas distribuídos, por exemplo, implicam na descentralização da execução de tarefas, consequentemente, o modelo de transações utilizado para modelá-las deve permitir a definição de transações que executam em nodos remotos de uma rede de banco de dados. A maioria dos sistemas computacionais atuais necessita comunicar-se mutuamente a fim de melhorar sua eficiência (rapidez de desenvolvimento e execução de tarefas) (BAPTISTA, 1994). O modelo convencional de transações não prevê mecanismos suficientes para a definição de aplicações distribuídas, o que deve ser totalmente implementado pelo programador da aplicação. Aplicações de longa duração, por sua vez, não devem ser modeladas como transações Atômicas, pois isto exigiria, em caso de falha, que o trabalho antes efetuado fosse totalmente desconsiderado. Da mesma maneira, transações longas não devem executar Isoladamente, porque isto impede, por longo tempo, que outras transações tenham acesso aos dados sendo manipulados. Neste sentido, para melhor suportar os requisitos de processamento das aplicações relacionadas, novos modelos de transações vêm sendo propostos.

No âmbito do projeto TRANSCOOP, vêm sendo realizados vários estudos sobre modelos de transações não convencionais. Dentre eles, encontra-se o Modelo de Contratos (WACHTER; REUTER, 1993), prevendo um mecanismo de controle para gerenciar aplicações distribuídas que apresentam atividades de longa duração de maneira segura (BANCILHON; KIM; KORTH, 1985). É um modelo de programação que garante persistência, consistência, recuperação em caso de falhas, sincronização e cooperação. O Modelo de Contratos foi definido por (REUTER, 1989) e outros trabalhos o descrevem com mais detalhes (WACHTER; REUTER, 1993; WACHTER, 1991; REUTER; SCHWENKREIS; WAECHTER, 1992).

A idéia básica do modelo constitui-se em elaborar aplicações de longa duração a partir da composição de transações ACID (BERNSTEIN; HADZILACOS; GOODMAN, 1987) e prover um mecanismo independente que controle o seu fluxo de execução. Segundo (WACHTER; REUTER, 1993), um contrato é a execução segura de uma seqüência arbitrária de ações pré-definidas (*steps*) de acordo com uma descrição explícita do fluxo de controle (*script*).

Algumas das características suportadas pelo modelo são:

- reusabilidade de código
- mecanismo para definir e controlar transações em termos estáticos e dinâmicos
- construção de atividades complexas através de várias atividades menores
- recuperação progressiva
- contexto persistente
- liberação de resultados intermediários antes do término completo da execução de um contrato
- especificação de atividades compensatórias para desfazer resultados já atingidos
- política de acesso a objetos compartilhados
- resolução de conflitos
- suspensão e retomada de execuções de contratos
- facilidades para mostrar o estado atual da aplicação

Os *steps* constituem a parte algorítmica da aplicação, são as unidades de trabalho elementares e devem ser executados como unidades ACID. Internamente, um *step* não contém nenhum paralelismo e seu código pode ser implementado em qualquer linguagem de programação convencional.

O *script* descreve a estrutura da atividade complexa. O fluxo de controle entre os *steps* pode ser modelado através dos construtores: seqüência, salto, laço e alguns construtores de paralelismo. Através de um *script* também é possível descrever o encapsulamento de *steps* em unidades ACID maiores. Neste caso, a execução de um contrato deve considerar estes conjuntos de *steps* como transações convencionais.

O objetivo deste trabalho é criar uma interface que permita a **definição** e o **monitoramento da execução** de contratos. Para tanto foi necessária a definição da arquitetura de um protótipo que implemente as características do Modelo de Contratos. O protótipo foi definido e constitui-se de dois módulos principais: o **módulo Interface** e o **módulo Gerente de Contratos**. O módulo Interface, objeto de estudo deste trabalho, permite ao usuário definir, compilar e executar contratos. Para tanto, ele é subdividido em outros dois módulos: **Interface de Definição** e **Interface de Execução**.

Através da **Interface de Definição** é possível estabelecer-se todas as condições para a execução segura de um contrato. Além disseo, ela é provida de uma operação para compilação de contratos, a qual gera um código intermediário que será interpretado pelo módulo Gerente de Contratos.

A **Interface de Execução**, por sua vez, é responsável pela apresentação de informações sobre o andamento da execução dos contratos, tais como: quais *steps* executaram com sucesso, quais *stes* falharam, etc.

A execução dos contratos é realizada pelo módulo Gerente de Contratos e não será detalhada neste trabalho. Entretanto, como todos os serviços da Interface, relacionados à execução de contratos, são solicitados ao Gerente de Contratos, foi preciso definir um protocolo de comunicação entre os dois módulos. O protocolo é composto por um conjunto de mensagens preestabelecidas trocadas entre os módulos do sistema.

A definição do protótipo possibilitou a visualização de alguns elementos novos que deveriam pertencer ao modelo. Foram propostas, então, algumas variações para os tipos de comandos apresentados no modelo.

Todas estas funcionalidades do protótipo precisavam ser apresentadas ao usuário de uma forma adequada, considerando que o perfil de usuário para este tipo de sistema é bem variado. O programador de *steps*, por exemplo, é um usuário mais especializado, alguém que deve possuir conhecimento das áreas de banco de dados e programação. Para este usuário, a utilização de uma linguagem textual não seria um grande inconveniente. Apesar de que o estudo e aplicação de linguagens visuais (GLINERT, 1990) para consultas e atualizações a banco de dados vêm se tornando uma área de pesquisa largamente explorada (KIM; KORTH; SILBERSCHATZ, 1988; LEONG; SAM; NARASIMHALU, 1989; WU; HSIAO, 1989; MELLO, 1994).

Entretanto, não é necessário que o programador do *script* se envolva em detalhes muito específicos, pois as ações (consultas e atualizações) a serem tomadas em relação às bases de dados já estão contidas nos *steps* e todo o mecanismo de controle da execução distribuída dos *steps* faz parte do próprio sistema. Assim sendo, a definição da transação, por exemplo, pode envolver a execução de diversas atividades (consultas ou atualizações a bases de dados) em nodos distintos de uma rede de computadores. Para expressar a seqüência com que estas tarefas devem ser executadas poderia optar-se por uma linguagem textual. Porém, foi constatado, através de pesquisas envolvendo outras áreas da ciência, que linguagens orientadas a texto apresentam maior grau de complexidade no que diz respeito a sua assimilação e aplicação. Logo a linguagem textual aplica-se a usuários especializados, técnicos da área com experiência na utilização de outras linguagens da mesma classe.

A tecnologia visual é empregada em grande escala nas interfaces dos sistemas existemtes atualmente, principalmente nas áreas de CAD (Computer Aided Design) e sistemas de informação de escritórios (FREITAS, 1992). Através de metáforas do mundo real, as interfaces gráficas de usuário aproximam as atividades exercidas naturalmente pelos usuários às atividades realizadas através dos computadores. Ao contrário da interação através de uma linguagem textual, a linguagem visual oferece uma interface mais intuitiva e de fácil e rápido aprendizado.

O estudo de interfaces com o usuário compreende uma área multidisciplinar, envolvendo aspectos de psicologia, aprendizagem, comunicação, artes visuais, educação, ergonomia, ciência da computação, entre outras. Durante este trabalho foram realizadas pesquisas em algumas destas áreas com o intuito de prover o projeto de uma interface mais adequada para o protótipo do Modelo de Contratos.

O restante deste trabalho está organizado como segue. No Capítulo 2, descreve-se o Modelo de Contratos, base para o desenvolvimento do protótipo. No Capítulo 3, é apresentada a arquitetura do protótipo, descrevendo cada um dos Módulos que a constituem e a comunicação entre eles. No Capítulo 4, é apresentado um estudo sobre os aspectos de outras áreas da ciência que envolveram o projeto da interface. No Capítulo 5, é detalhado o projeto da interface. E, por fim, são relatadas as conclusões do trabalho e sugeridas possíveis extensões no Capítulo 6.

2 O MODELO DE CONTRATOS

O Modelo de Contratos provê uma base formal para a definição e controle de atividades complexas de longa duração do mesmo modo que transações convencionais controlam as atividades de curta duração. "Um Contrato é a execução invulnerável e consistente de uma sequência de ações pré-definidas (chamadas *steps*) de acordo com uma descrição de fluxo de controle definida explicitamente (chamada *script*)" (WACHTER; REUTER, 1993). O programador deve fornecer ao sistema todas as informações necessárias à execução da aplicação, informações sobre os *steps* e o relacionamento entre eles. Todos os aspectos relacionados ao controle da execução são tarefas do Gerente de Contratos (*The Contract Manager*).

O *script* contém a definição explícita do fluxo de controle da aplicação. A Figura 2.1 mostra a representação gráfica de um *script*. A modelagem é de uma aplicação que organiza eventos. Cabe a ela a reserva de um local acessível para a realização do evento e de hotel e transporte para os participantes. Os círculos representam os *steps* e os arcos o fluxo de execução entre eles. As bordas tracejadas ao redor dos *steps* (S_4, S_5) e (S_6,S_7) representam o encapsulamento desses *steps* e, duas transações, T_1 e T_2.

A Figura 2.2 mostra a representação textual para o *script* relacionado à mesma aplicação. Ele é composto por quatro seções distintas, a saber:

- **CONTEXT_DECLARATION** seção onde são declaradas as variáveis de contexto utilizadas pelo contrato;
- **CONTROL_FLOW_SCRIPT** seção que contém a definição explícita do fluxo de controle da aplicação;
- **TRANSACTIONS** seção destinada à definição das transações que encapsulam os *steps*;
- **SYNCRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS** seção onde são definidas as invariantes relacionadas a cada *step* e os *steps* de resolução de conflito.

A sintaxe completa da linguagem de definição de contratos textual é apresentada em (WACHTER; REUTER, 1991).

Os *steps* são unidades de trabalho elementares e devem ser executados como unidades ACID. Internamente, um *step* não contém nenhum paralelismo e seu código pode ser implementado em qualquer linguagem de programação convencional. Quanto à quantidade de trabalho de que um *step* é constituído, deve ser considerado que esta porção de trabalho possa ser recuperada caso ocorra alguma falha no sistema. A Figura 2.3 apresenta

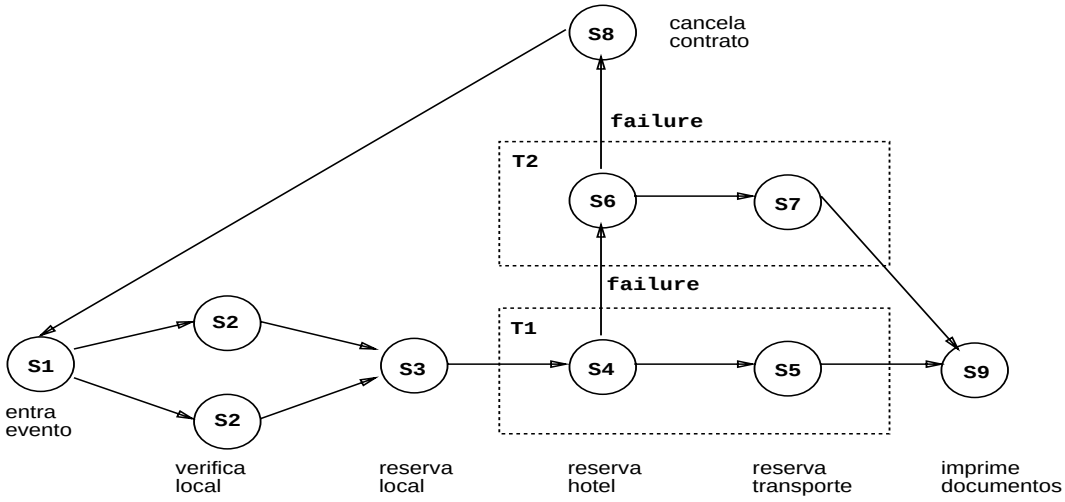

Figura 2.1: Representação Gráfica do SCRIPT

```
1   CONTRACT Organização_Eventos
2   CONTEXT_DECLARATION
3       Bool ok;
4       Int l_custo, n_partic, custo;
5       Array[30] local, buffet, hotel, agencia, data, transporte;
6   CONTROL_FLOW_SCRIPT
7       S1:entra_evento(data, l_custo, n_partic);
8          AltParForEach(
9             Select nome, custo
10            From LOCAIS
11            Where capacidade > n_partic & custo <= l\_custo
12            )
13            S2:verifica_Local(nome, data, custo);
14         EndAltParForEach
15       l_custo = l_custo - custo;
16       S3:reserva_local(local, data, ‘‘encontro_calcadistas’’);
17       S4:reserva_hotel(hotel, data, n_partic);
18       If (ok)
19       Then S5:reserva_transporte(transporte, data, n_partic);
20       Else Begin
21            S6:reserva_hotel(hotel, data, n_partic);
22             If (ok)
23             Then S7:reserva_transporte(transporte, data, n_partic);
24             Else S8:cancela_evento(local, buffet, data, agencia, hotel, transporte);
25            End
26       S9:imprime_documentos(agencia, data, local, buffet);
27 END_CONTROL_FLOW_SCRIPT
28 SYNCRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS
29       S1:EXIT_INVARIANT(l_custo > ‘‘valor_mínimo’’)
30         CONFLICT_RESOLUTION: S0:chamar_administrador();
31       S3:EXIT_INVARIANT(l_custo > ‘‘valor_mínimo’’)
32         CONFLICT_RESOLUTION: S0:chamar_administrador();
33       S4:EXIT_INVARIANT(l_custo > ‘‘valor_mínimo’’)
34         CONFLICT_RESOLUTION: S0:chamar_administrador();
35       S5:EXIT_INVARIANT(l_custo > ‘‘valor_mínimo’’)
36         CONFLICT_RESOLUTION: S0:chamar_administrador();
37       S6 :EXIT_INVARIANT(l_custo > ‘‘valor_mínimo’’)
38         CONFLICT_RESOLUTION: S0:chamar_administrador();
38 END_SYNCRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS
40 END_CONTRACT Organização_Eventos
```

Figura 2.2: Representação Textual do SCRIPT

o código de um *step*, escrito na linguagem “C”, que faz a reserva de um local para a realização de um evento. Os *steps* podem ser programados em qualquer linguagem, desde que se tenha acesso ao código executável do mesmo. No *script* gráfico, a execução deste *step* é referenciada pelo círculo contendo a inscrição S_3 e, no *script* textual, a linha número 12 mostra a chamada ao *step* S_3.

```
EXEC SQL INCLUDE SQLCA;

#define IGUAL 0
#define OK 1
#define NOK 0

EXEC SQL DECLARE agenda_locais TABLE
      ( data           char(7) NOT NULL,
        ocupação       char(30)NOT NULL
       ) ;

reserva_local(nome_local, data, agencia)
char nome_local[30], data[7], agencia[30];
{
    EXEC SQL BEGIN DECLARE SECTION;
        char datai[7];
        char ocupação[30];
    EXEC SQL END DECLARE SECTION;

    printf(‘‘n Reservando %s como local para o dia %s’’, nome_local, data);
    strcpy(datai, data);
    if (memcmp(nome_local, ‘‘PLAZA’’,5)==IGUAL)
        EXEC SQL CONNECT plaza SESSION 2;
    else
        if (memcmp(nome_local, ‘‘UFRGS’’,5)==IGUAL)
            EXEC SQL CONNECT ufrgs SESSION 2;
    else
        if (memcmp(nome_local, ‘‘GERMANIA’’,8)==IGUAL)
            EXEC SQL CONNECT germânia SESSION 2;
    else
        if (memcmp(nome_local, ‘‘FIERGS’’,6)==IGUAL)
            EXEC SQL CONNECT fiergs SESSION 2;
    strcpy(ocupação, agencia);
    EXEC SQL SET_INGRES (SESSION=2);
    EXEC SQL UPDATE agenda_locais
        SET ocupação = :ocupação
        WHERE data = :datai;
    EXEC SQL DISCONNECT;
    EXEC SQL SET_INGRES (SESSION=1);
    return(OK);
}
```

Figura 2.3: Código do STEP Reserva Local

A programação dos *steps* é totalmente independente da programação do *script*. Com isto, a vantagem é que o programador de *steps* não precisa se preocupar com os aspectos relacionados à sincronização, paralelismo, distribuição e recuperação de *steps*. Estes problemas são explicitados durante a programação do *script*. Outra vantagem que a programação em dois níveis apresenta é a possibilidade de várias pessoas diferentes poderem programar os *steps*, pois estes executam funções que não dependem do conhecimento da aplicação como um todo. Uma conseqüência natural deste modelo de programação é a reutilização dos *steps* em várias transações longas (contratos) independentes.

O Modelo de Contratos permite que sejam associadas aos *steps* pré– e pós–condições com o objetivo de controlar o acesso aos objetos compartilhados por mais de uma aplicação, são as chamadas invariantes de entrada e de saída. A invariante de entrada é testada antes da execução do *step* iniciar, caso a avaliação do predicado que compõe a invariante

seja verdadeira, a execução do *step* é iniciada. Caso o resultado seja falso, um *step* de resolução de conflitos pode ser executado. Portanto, juntamente com as invariantes podem ser definidos *steps* que desempenham atividades alternativas, as quais tem como objetivo resolver o conflito ocorrido no acesso concorrente aos objetos. Na Figura 2.2, a partir da linha 24, é possível verificar a definição de algumas invariantes e *steps* de resolução de conflitos a elas associados.

2.1 Controle de Aplicações Distribuídas

A seguir serão descritos alguns mecanismos importantes para o controle de aplicações distribuídas providos pelo modelo de Contratos.

Suspensão

Como as transações modeladas tem um longo tempo de duração, a idéia de permitir a suspensão da execução de um contrato e a sua retomada após um determinado período de tempo é bastante apropriada. No exemplo apresentado neste capítulo, o cliente pode decidir interromper a reserva de um local (*step* S_3) para a realização do evento até que existam reservas de hotel e transporte disponíveis para o mesmo período.

Migração

Um outro mecanismo previsto pelo modelo é a migração do controle da execução de um contrato (do Gerente de Contratos) para outro nodo da rede caso o nodo original falhe. Isto poderia ser provido por um procedimento de transferência de todo o contexto administrado pelo Gerente de Contratos do nodo interrompido para um nodo com um novo Gerente operando.

Informações de Contexto

E um outro serviço oferecido pelo modelo é a apresentação de informações sobre o estado de execução de todo o contrato, a fim de oferecer ao usuário um *trace* sobre o seu andamento. Isto somente torna-se possível, pois todo o histórico da execução dos contratos permanece armazenado no contexto.

2.2 O Modelo de Transações

Cada *step*, no Modelo de Contratos, é uma unidade de trabalho ACID, a menos que se especifique o contrário na seção **TRANSACTIONS** do script. Nesta seção, é permitido que unidades de trabalho maiores, também atômicas, sejam definidas através do agrupamento de *steps*. Na Figura 2.1, a linha pontilhada traçada em volta dos *steps* S_4 e S_5 significa que ambos formam uma transação atômica e isolada, referenciada como T_1.

As transações, por sua vez, podem ser agrupadas para formar outras transações. O trecho de *script* abaixo mostra a representação textual para a definição de transações.

TRANSACTIONS
T1:(S_4, S_5);
T2:(S_6, S_7);

O Modelo de Contratos também permite que sejam estabelecidos caminhos alternativos para o fluxo de execução da aplicação, dependendo dos resultados obtidos com a execução dos *steps* e das transações. Por exemplo, quando uma determinada transação

falha, de acordo com a semântica da aplicação pode ser desejável que outra seja ativada e execute em seu lugar. Na representação gráfica, Figura 2.1, isto é demostrado pelo arco "failure", significando que outra transação será ativada caso a transação responsável pela reserva do local e transporte (*steps* S_4 e S_5) falhe. O segundo arco "failure" indica que o contrato deve ser cancelado (*step* S_8). Na representação textual isso é feito através da associação da seguinte restrição junto à declaração da transação:

TRANSACTIONS
T1:(S_4, S_5), DEPENDECY (T_1:abort |—> begin:T_2)

2.3 Recuperação

Após a ocorrência de uma falha, um sistema confiável deve recuperar todas as tarefas terminadas ou em andamento, tentando minimizar a perda de trabalho, ou seja, o contrato deve continuar sua execução a partir do último estado válido em que se encontrava antes da falha (*Forward Recovery*). Para garantir esta recuperação é preciso que certas informações sobre o estado de execução do contrato sejam recuperáveis:

- o estado do sistema visto por todas as aplicações, isto é, os bancos de dados de cada aplicação;
- o estado local do contrato, por exemplo, as variáveis dos programas, cursores, etc., utilizados por mais de um *step*;
- o estado global da computação, ou seja, quais *steps* já foram executados, quais não foram, etc.

Para armazenar todas estas informações o Modelo de Contratos introduz o conceito de "contexto persistente". O contexto mantém estáveis todos os estados locais relevantes a uma atividade de longa duração. A cada alteração de um elemento do contexto é criada uma nova versão para o mesmo, permitindo que se tenha toda a história de execução de um contrato. Ao término de cada contrato, todo seu contexto é descartado.

As variáveis utilizadas pelos *steps* são declaradas em uma seção especial do *script*, a **CONTEXT_DECLARATION**, como pode ser visto na Figura 2.2. As demais informações integrantes do contexto são mantidas pelo sistema, assim como todo o seu gerenciamento, os programadores não tem nenhum trabalho de programação adicional em relação a isto.

Em determinados casos de falha, o estado físico da aplicação fica inconsistente em relação ao Banco de Dados e, muitas vezes, atividades compensatórias devem ser executadas para retornar a aplicação para um estado consistente. Entretanto, não é possível modificar o *script* em tempo de execução para definir as atividades compensatórias, elas devem ser previstas durante a definição do *script* de cada contrato.

3 ARQUITETURA DO PROTÓTIPO

Um protótipo do Modelo de Contratos, apresentado no Capítulo 3, encontra-se em fase de desenvolvimento no CPGCC da UFRGS, como parte do projeto TRANSCOOP. O trabalho envolve uma dissertação de mestrado e um trabalho de diplomação do curso de graduação. Conforme a Figura 3.1, que mostra a arquitetura do protótipo, existem dois módulos principais, chamados **Interface** (SILVA; IOCHPE, 1994) e **Gerente de Contratos (GC)**.

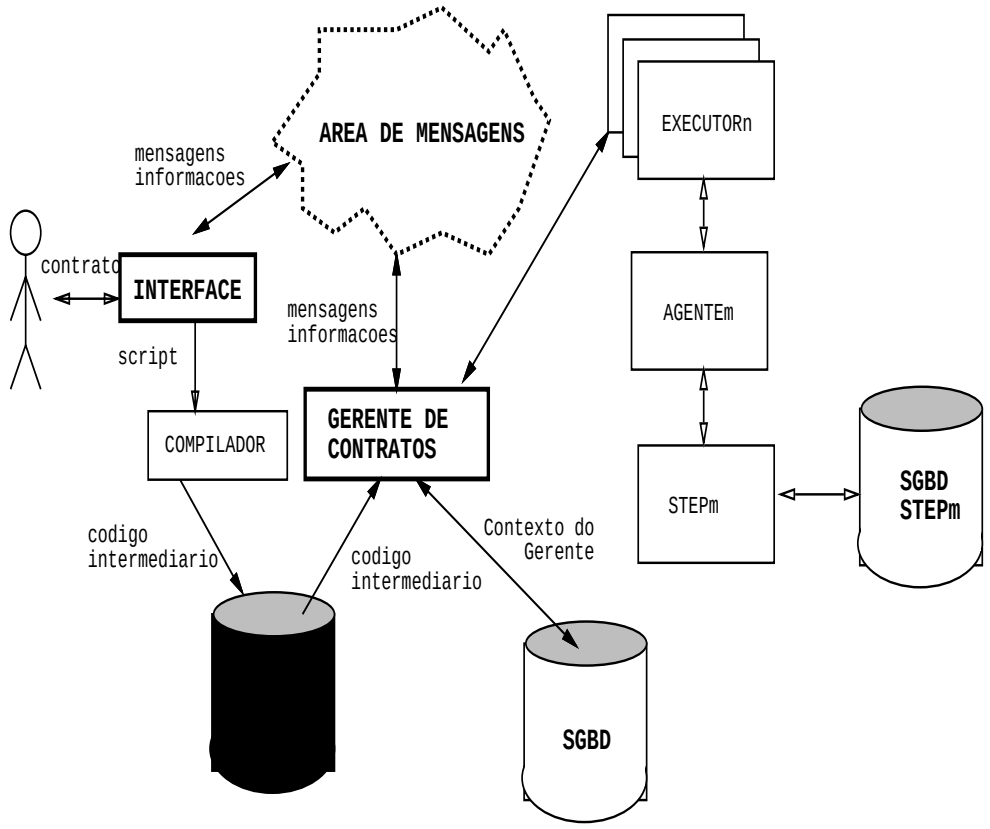

Figura 3.1: Arquitetura do Protótipo

O **Módulo Interface** é objeto de estudo deste trabalho. Através dele o usuário pode definir, compilar e executar transações complexas (o *script*). Durante a compilação é gerado o código intermediário que será interpretado por um executor do **Módulo Gerente de Contratos**, objeto de estudo de um trabalho de diplomação de graduação. O Módulo Interface ainda é responsável pelo monitoramento das execuções de Contratos e *steps*. Através deste monitoramento o usuário pode acompanhar a execução das transações lon-

gas por ele submetidas ao sistema. Com este propósito existe a Área de Mensagens, onde o Módulo Gerente de Contratos escreve os resultados da interpretação dos contratos e o Módulo Interface lê e informa de maneira adequada ao usuário.

O Gerente de Contratos tem a função de realizar os serviços solicitados pela Interface e manter seu próprio contexto de execução seguro no banco de dados. Os serviços atendidos por ele são: **Executa** contrato, **Suspende** execução, **Retoma** execução e **Aborta** contrato. Para cada contrato executado, o Gerente de Contratos cria uma nova instância do Sub–Módulo Executor, o qual fica responsável pela interpretação do código intermediário daquele contrato e pela distribuição de *steps* remotos sobre a rede de bancos de dados através de Agentes.

3.1 Comunicação entre os Módulos Gerente de Contratos e Interface

Como será apresentado no Capítulo 5, o Módulo Interface engloba a definição e o monitoramento da execução de contratos. A arquitetura de processos envolvendo esta definição é mostrada na Figura 3.2. O processo principal, chamado *ProcessoInterface*, é responsável pela criação dos demais processos. Para cada definição de contrato em andamento existe um processo executando, chamado *ProcessoDefinição*. E para o monitoramento de cada execução existe um processo chamado *ProcessoExecução*.

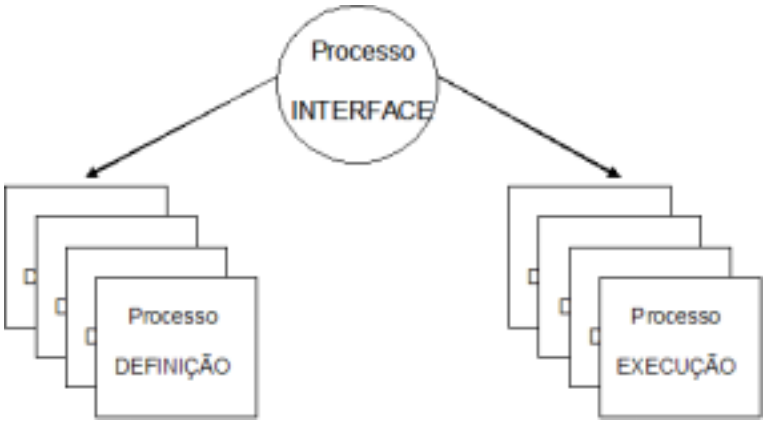

Figura 3.2: Arquitetura de Processos do Módulo Interface

O Módulo Gerente de Contratos (GC) também é composto por um conjunto de processos, como mostrado na Figura 3.3. Cada ocorrência do *ProcessoExecutor* é responsável pela execução de um contrato. Portanto, se existirem, por exemplo, três contratos em execução ao mesmo tempo, também existirão três ocorrências do *ProcessoExecutor*.

Em sua fase inicial de execução, o sistema cria dois processos que executam os dois Módulos principais, Interface e Gerente de Contratos. Como já foi mencionado o Módulo Interface solicita alguns serviços ao GC e é neste caso que ocorre a comunicação entre os dois Módulos. A seguir são apresentados os casos em que ocorre comunicação e qual o tipo de informação trocada.

3.1.1 Definição

Quando a Interface Definição é ativada o *ProcessoDefinição* é criado.

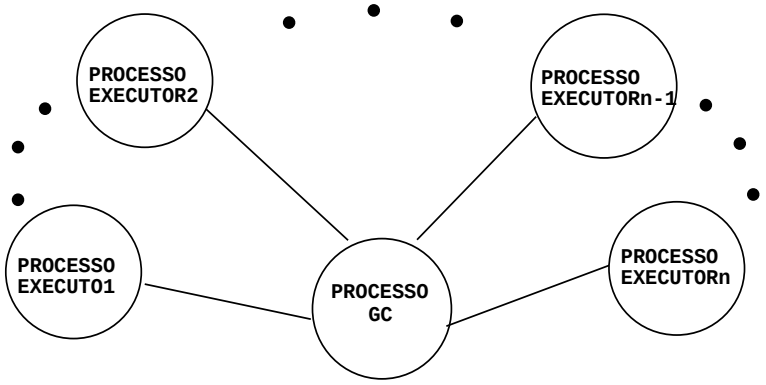

Figura 3.3: Arquitetura dos Processos do Módulo Gerente de Contratos

Definição de Contratos

Durante a definição de contratos não ocorre nenhuma comunicação do *ProcessoDefinição* com o *ProcessoGC* e ele somente é finalizado quando é solicitado o término da Interface Principal.

Compilação

Na fase de compilação, serviço também oferecido pela Interface de Definição, também não ocorre nenhuma comunicação direta entre os dois processos. Mas, indiretamente, ocorre uma comunicação porque é neste momento que é gerado o Código Intermediário, o qual será, posteriormente, interpretado por uma instância do Sub-Módulo Executor do Gerente de Contratos.

3.1.2 Execução

Quando a Interface Execução é ativada é criado o *ProcessoExecução*.

Execução

No momento em que a execução de um contrato é ativada o *ProcessoExecução* envia uma mensagem (**msgInicExecução**)[1] ao *ProcessoGC* solicitando que o *ProcessoExecutor* do contrato indicado na mensagem inicie. A Figura 3.4 mostra os processos ativos neste momento e a Figura 3.5 a notação utilizada.

Durante a execução de um contrato, quando existe a necessidade da execução remota de algum *step*, o *processoExecutor* do contrato cria um novo processo chamado *ProcessoAgente*. Este processo é responsável pelo monitoramento da execução do *step* remoto, ele que faz o interfaceamento entre o sistema de banco de dados remoto e o *step* definido no contrato [2].

Quando um contrato qualquer está em execução, o processo que contém o Sub–Módulo Executor, responsável por sua execução, encontra-se ativo e envia informações sobre o andamento da execução para a área de mensagens do contrato. Tais informações são apresentadas na tabela 3.1. O *ProcessoExecutor* também envia mensagens ao *ProcessoExecução* informando sobre o estado de execução dos *steps* por ele já executados, abortados ou em execução, respectivamente as mensagens: **msgStepSucesso**,

[1] todas as mensagens são descritas a seguir, juntamente com as informações nelas enviadas

[2] A definição do funcionamento interno do Gerente de Contratos não é objetivo deste trabalho, entretanto, algumas idéias, como a existência dos Sub–Módulos Executor e Agente, são sugeridas aqui

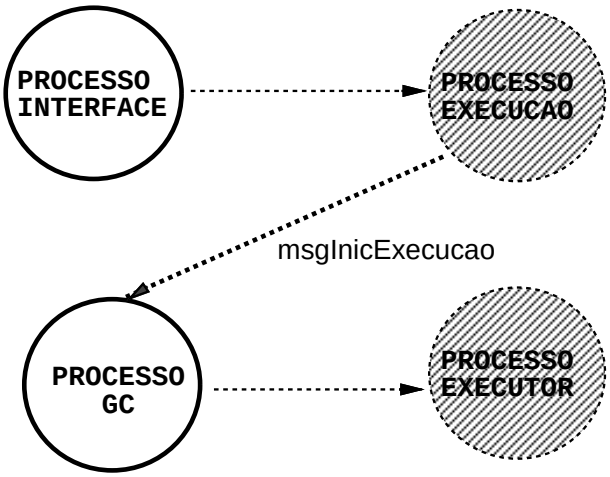

Figura 3.4: Processos Ativos no Momento da Execução

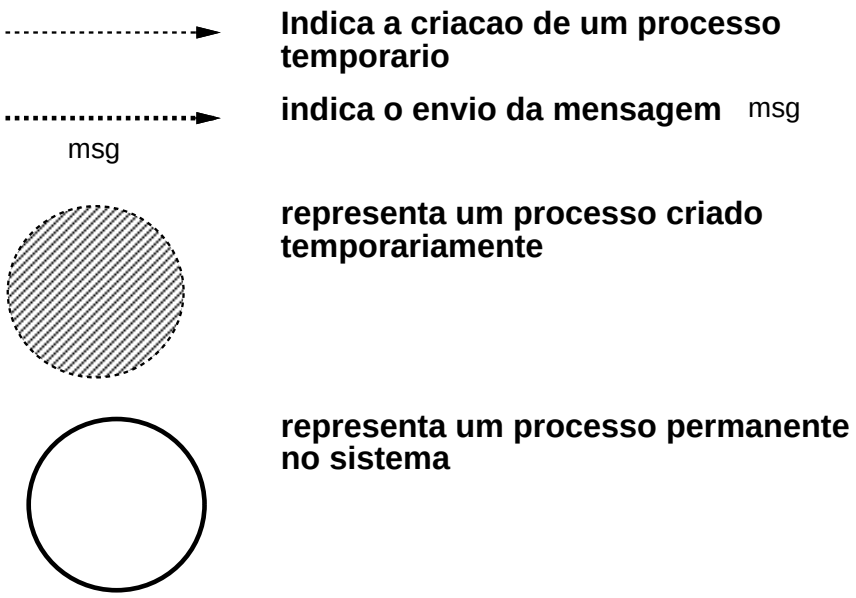

Figura 3.5: Notação utilizada na especificação dos Processos

msgStepAbortado e **msgStepExecutado**. Os *steps* não indicados em nenhuma destas mensagens são considerados como ainda não executados.

Suspensão

Todo contrato em execução pode ser suspenso a qualquer momento através de uma função específica chamada da Interface de Execução. Quando este serviço é solicitado, o *ProcessoExecução* envia uma mensagem (**msgSuspExecução**) ao Gerente de Contratos informando que ele deve suspender a execução do contrato indicado. Este, então, ativa o procedimento responsável pelo tratamento de tal mensagem e, logo após, finaliza o processo que contém o Sub–Módulo Executor do contrato. Porém, a área de mensagens correspondente ao contrato suspenso não é descartada, a Interface continua com o seu controle. A Figura 3.6 mostra a situação dos processos neste momento.

Também é possível, quando todos os contratos estão suspensos ou finalizados, terminar a execução dos dois processos principais, Interface e Gerente de Contratos. Neste

	CONTRATO	STEPS
Hora Início	√	√
Hora Fim	√	√
Não Executado		√
Executando	√	√
Executado	√	√
Suspenso	√	
Abortado	√	
Invariantes de Entrada		√
Invariantes de Saída		√

Tabela 3.1: Informações mantidas sobre os Steps e Contrato

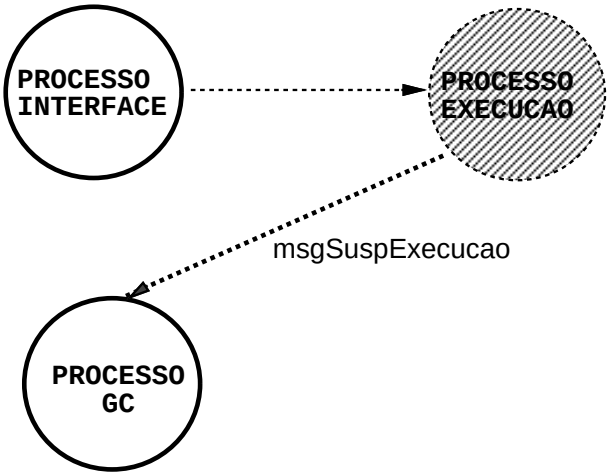

Figura 3.6: Processos Ativos no Momento da Suspensão

caso, a interface transfere a identificação das áreas de mensagens de cada contrato para um meio de armazenamento permanente, envia uma mensagem (**msgFinalizaGC**) ao *ProcessoGC* para que ele termine sua execução e, logo após, termina sua própria execução. Quando o *ProcessoInterface* é reativado, ele restaura todas as áreas de mensagens e envia uma mensagem (**msgReativaExecução**) ao *ProcessoGC* para que ele restaure todas as suas informações de contexto.

Retomada

Um contrato suspenso também pode ser retomado, neste caso, o *ProcessoExecução* envia uma mensagem (**msgResumirExecução**) ao *ProcessoGC* solicitando que ele recupere todas as informações sobre o contrato suspenso e continue sua execução normal. O Gerente cria, então, um novo processo com o Sub–Módulo Executor.

Recuperação

Quando ocorre alguma falha de sistema durante a interpretação de um contrato, após a reativação do sistema, fica ao encargo do *ProcessoGC* a recuperação das áreas de mensagens. Isto porque é ele quem mantém todas as informações sobre os contratos já executados, em execução, etc. e, em caso de recuperação de falha, deve decidir quais *steps*

devem ser desfeitos (*UNDO*), refeitos (*REDO*), etc.

Término Anormal de Execução

Também é possível solicitar o término (*abort*) da execução de um contrato em andamento. Sendo assim, o *ProcessoExecução* envia uma mensagem (**msgAbortarExecução**) ao *ProcessoGC*, solicitando o serviço. Este executa o procedimento responsável por "abortar" contratos, terminando o *ProcessoExecutor* do mesmo e enviando uma mensagem (**msgExecuçãoAbortada**) de volta ao *ProcessoExecução*. A seguir, a Interface transfere a identificação da área de mensagens correspondente àquele contrato para o disco, pois o **ProcessoExecução** ainda encontra-se ativo e pode solicitar informações sobre o estado de execução do contrato.

Término Normal de Execução

É de maneira similar ao caso anterior que a execução de um contrato termina normalmente. Quando o *ProcessoExecutor* detecta que a interpretação [3] do código intermediário de um contrato chegou ao final, o *ProcessoGC* notifica o *ProcessoExecução* (**msgExecuçãoTerminada**).

Os processos do Módulo Interface e dos Sub-Módulos Definição e Execução somente são terminados quando isto é explicitamente solicitado através de alguma função da interface ou em caso de falha de sistema.

3.1.3 Lista de Mensagens

A seguir são descritas as mensagens trocadas entre os processos que constituem o sistema.

msgInicExecução

Enviada do *ProcessoExecução* ao *ProcessoGC*, solicitando o início da execução de determinado contrato.

- identificação do contrato;
- nome do arquivo, contendo o código intermediário a ser executado pelo Gente de Contratos;
- localização da área de mensagens referente ao contrato que será executado.

msgSuspExecução

Enviada do *ProcessoExecução* ao *ProcessoGC*, solicitando a suspensão da execução de determinado contrato.

- identificação do contrato

msgFinalizaGC

Enviada do *ProcessoInterface* ao *ProcessoGC*, indicando que ele deve salvar todas as suas informações de contexto e finalizar sua execução.

[3] Como será visto mais adiante, o código intermediário gerado pelo Módulo Interface será interpretado pelo Gerente de Contratos. No texto é usado o termo "execução" para indicar o resultado da interpretação

msgReativaExecução

Enviada do *ProcessoInterface* ao *ProcessoGC*, indicando que ele deve recuperar todas as suas informações de contexto.

msgResumirExecução

Enviada do *ProcessoExecução* ao *ProcessoGC*, solicitando que ele recupere todas as informações sobre determinado contrato e retome sua execução.

- identificação do contrato

msgAbortarExecução

Enviada do *ProcessoExecução* ao *ProcessoGC*, solicitando que a execução de determinado contrato seja terminada.

- identificação do contrato

msgExecuçãoAbortada

Enviada do *ProcessoGC* ao *ProcessoExecução*, informando que a execução do contrato foi abortada.

msgExecuçãoterminada

Enviada do *ProcessoGC* ao *ProcessoExecução* informando que a execução do contrato terminou com sucesso.

msgStepSucesso

Enviada do *ProcessoExecutor* ao *ProcessoExecução*, informando que a execução daquele *step* terminou com sucesso.

- identificação do *step*

msgStepAbortado

Enviada do *ProcessoExecutor* ao *ProcessoExecução*, informando que a execução daquele *step* foi abortada.

- identificação do *step*

msgStepExecutando

Enviada do *ProcessoExecutor* ao *ProcessoExecução*, informando que o *step* encontra-se em execução.

- identificação do *step*

3.1.4 O Código Intermediário

Durante a compilação de um contrato é gerado o Código Intermediário definido especificamente para ser interpretado [4].

[4]Como já foi mencionado, a definição do funcionamento interno do Gerente de Contratos não faz parte deste trabalho, entretanto o Código Intermediário foi projetado com o objetivo de ser interpretado por um Sub–Módulo Executor do Gerente de Contratos

O código intermediário gerado pela compilação de um contrato é composto por quatro seções, a saber: “declaração”, “declaraçãoSteps”, “declaraçãoTrans” e “fluxoControle”. A BNF [5] completa contendo a descrição do Código Intermediário é apresentada na Figura 3.7.

Na seção “declaração” são apresentadas todas as variáveis de contexto utilizadas no contrato e seus tipos particulares. Esta é a primeira seção porque, no início da interpretação, o Sub–Módulo Executor cria uma entrada para cada uma das variáveis declaradas no seu banco de dados local (nodo onde o código está sendo interpretado). Logo, todas as alterações sofridas por estas variáveis devem ser feitas através da execução de transações sobre o banco de dados local. O objetivo é fazer com que o contexto mantenha-se inviolável, pois é através dele que o modelo é capaz de prover a recuperação de falhas com um mínimo de perda possível.

Na seção “declaraçãoSteps” aparece a declaração de todos os *steps* que constituem o contrato. As informações relacionadas aos *steps* são as seguintes:

- número que identifica o *step* para o contrato;
- endereço de rede onde se encontra o programa executável correspondente ao código do *step*. Esse endereço é necessário para a localização e execução do código correspondente ao *step*;
- nome do programa que contém o código executável correspondente ao *step*;
- parâmetros de entrada, valores variáveis ou constantes enviados pelo Executor ao programa que executa o código do *step*;
- parâmetros de saída, nomes de variáveis onde serão retornados os valores resultantes da execução dos *steps*;
- invariante de entrada, expressão booleana composta por variáveis de contexto que será avaliada antes da execução de cada *step*;
- número do *step* de Resolução de Conflito executado caso a invariante de entrada falhe (resultado FALSO);
- invariante de saída, expressão booleana composta por variáveis de contexto que será avaliada após a execução de cada *step*;
- número do *step* Resolução de Conflito executado caso a invariante de saída falhe (resultado FALSO);
- número do *step* compensatório executado quando a atividade realizada por este *step* deve ser desfeita (compensada), após sua conclusão com sucesso.

Na seção seguinte, “declaraçãoTrans”, são apresentadas as declarações de todas as transações que fazem parte do corpo do contrato. É indicada, junto a cada transação, a lista de seus componentes, sendo cada item da lista composto pelo tipo e identificação do componente. Uma transação é descrita deste modo porque ela pode ser definida como sendo um conjunto de *steps*, ou, um conjunto de outras transações, ou, ainda, conjuntos de *steps* e transações.

A última seção, “fluxoControle”, contém a chamada de cada comando que está no corpo do contrato.

[5] Baccus Naur Form: linguagem para descrição de gramáticas.

```
contrato          :: declaração declaraçãoSteps declaraçãoTrans fluxoControle
declaração        :: CTXDCL variáveis CTXDCLEND
variáveis         :: tipoVar tamNomeVar nomeVar [variáveis]
tipoVar           :: BOOLEAN | CHAR | INTEGER | ARRAY
tamNomeVar        :: inteiro
inteiro           :: dígito [inteiro]
dígito            :: 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9
nomeVar           :: identificador
identificador     :: letraSub [alfa]
letraSub          :: letra | '_'
letra             :: a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | x | y | w | z
alfa              :: dígito | letra | '_' [alfa]
declaraçãoSteps   :: STEPDCL steps STEPDCLEND
steps             :: numStep tamNomeProg nomeProg IN parâmetros OUT parâmetros
                     ININV invariante numStepRC OUTINV invariante numStepRC numStepComp [steps]
numStep           :: inteiro
tamNomeProg       :: inteiro
nomeProg          :: identificador
parâmetros        :: constante | variável
constante         :: string | numInteiro | numReal
string            :: STR tamString valor
numInteiro        :: NUMINT valor
numReal           :: NUMREAL valor
variável          :: CTXVAR numVar
numVar            :: inteiro
invariante        :: expressão
expressão         :: fator | fator operando expressão
fator             :: varContexto
varContexto       :: numVar
operando          :: MAIOR | MENOR | MAIORIGUAL | MENORIGUAL | AND | OR
numStepRC         :: inteiro
numStepComp       :: inteiro
declaraçãoTrans   :: TRANSDCL transações TRANSDCLEND
transações        :: transação [transações]
transação         :: numTransação numComponentes listaComponentes
numComponentes    :: inteiro
listaComponentes  :: tipoComponente numComponente
tipoComponente    :: step | transação
numComponente     :: inteiro
fluxoControle     :: CTRLFLOW comandos CTRLEND
comandos          :: step | transação | forEach | altPar | if [comandos]
step              :: numStep
transação         :: numTransação
forEach           :: parAlt ConstCons listaSteps ENDFOREACH
parAlt            :: PARFOREACH | ALPARFOREACH
ConstCons         :: constantes | consulta
constantes        : CONSTANTE lista
lista             :: LPAREN listaPar RPAREN [lista]
listaPar          :: parâmetro [listaPar]
consulta          :: SELECT listaVarContexto FROM tabela WHERE expressão
listaVarContexto  :: varContexto [listaVarContexto]
tabela            :: identificador
listaSteps        :: step [listaSteps]
altPar            :: ap listaSteps
ap                :: ALT | PAR
if                :: IF condição cmdThen [cmdElse]
condição          :: tamComdição expressão [endereçoElse]
cmdThen           :: comandos endereçoFimIf
cmdElse           :: comandos
endereçoFimIf     :: ENDERFIMIF inteiro
endereçoElse      :: inteiro
```

Figura 3.7: BNF do Código Intermediário

3.2 Compilação de Contratos

Durante a fase de definição de um contrato, algumas estruturas de dados são constituídas. Estas estruturas serão utilizadas pela compilação com o objetivo de prover a geração de código.

A primeira estrutura armazena a declaração das variáveis de contexto através das seguintes informações:

- **id** fornece uma identificação única para cada variável;
- **nome** é um identificador, deve ser único dentro da definição de um mesmo contrato;
- **tipo** representa o tipo da variável. Os tipos permitidos são: caracter, inteiro, real e arranjo;
- **prox**, identificação do próximo nodo da lista de variáveis de contexto.

A estrutura lista de *steps* contém todas as informações referentes às definições de cada *step* assim como a sua localização no fluxo de execução do contrato.

- **id**, identificação única para esta definição de *step*;
- **comp**, indica se aquela definição de *step* corresponde a um *step* compensatório;
- **tipo**, determina o tipo do *step* em relação ao seu modo de execução. No Capítulo 5 serão explicitados cada um dos tipos de *steps* permitidos na construção do *script*;
- **VFN**, se o *step* representado neste nodo da lista for o próximo a ser executado depois de um *step* do tipo **IF**, este campo poderá receber os seguintes valores:
  - **VERDADEIRO** se a execução do *step* anterior terminar com sucesso;
  - **FALSO** se a execução do *step* anterior terminar sem sucesso;
  - **NADA** se o *step* anterior não for do tipo **IF**;
- **nome**, identificação do *step*, relacionada à atividade por ele exercida;
- **nomeProg**, nome do programa, contendo o código executável do *step*;
- **nodo**, endereço da rede de computadores onde o *step* deve ser executado;
- **ParEnt**, é a identificação de uma lista onde estão armazenadas as variáveis de contexto, ou constantes, que servem como parâmetros de entrada para o *step*;
- **ParSaída**, é a identificação de uma lista onde estão armazenadas as variáveis de contexto que servem como parâmetros de saída para o *step*;
- **invEnt**, predicado representando a invariante de entrada do *step*;
- **RCEnt**, identificação de um nodo desta estrutura, contendo o *step* de resolução de conflito para a invariante de entrada. Este *step* será executado se a avaliação do predicado armazenado em **invEnt** tiver como resultado o valor FALSO.
- **invSaída**, predicado representando a invariante de saída do *step*;

- **RCSaída**, identificação de um nodo desta estrutura, contendo o *step* de resolução de conflito para a invariante de saída;
- **stepComp**, identificação de um nodo desta estrutura que contém a definição do *step* que realiza a atividade compensatória a do *step* sendo definido;
- **listaAnt**, identificação de uma lista onde estão armazenados os endereços dos *steps*, definidos na **listaSteps**, executados exatamente antes do *step* em questão;
- **ListaSuc**, identificação de uma lista onde estão armazenados os endereços dos *steps*, definidos na **listaSteps**, que devem ser executados exatamente após o *step* em questão;
- **ListaPar**, identificação de uma lista onde estão armazenados os endereços dos *steps*, definidos na **listaSteps**, que devem ser executados em paralelo com o *step* em questão;
- **ListaTrans**, endereço de um nodo da lista onde são definidas as transações, identificando a transação a qual este *step* pertence;
- **forEachConstCons**, este campo pode assumir dois valores de tipos diferentes:
  - **const**, identificação de uma lista onde encontram-se as variáveis ou constantes que servem de parâmetros de entrada para os *steps* tipo PAR_FOR_EACH_CONST e ALT_PAR_FOR_EACH_CONST [6];
  - **cons**, consulta SQL que serve de parâmetro de entrada para os *steps* do tipo PAR_FOR_EACH_CONS e ALT_PAR_FOR_EACH_CONS;
- **prox**, identificação do próximo nodo da lista de *steps*.

As informações apresentadas a seguir descrevem uma estrutura que se destina a armazenar as transações definidas nos contratos.

- **id**, fornece uma identificação única para cada transação;
- **listaComponentes**, identificação de uma lista onde estão armazenados todos os componentes da transação, *steps* ou outras transações;
- **prox**, identificação do próximo nodo da lista de transações.

Também foram definidas algumas estruturas auxiliares, as quais são descritas da seguinte forma:

- **listaVarConst** é uma estrutura do tipo listas de listas que serve para armazenar um número qualquer de parâmetros. Estes parâmetros podem ser variáveis de contexto ou constantes.
- A lista de componentes, chamada **listaComp**, permite a identificação dos componentes de uma transação, sejam eles *steps* ou outras transações. É importante salientar que os componentes das transações são dispostos nesta lista na mesma ordem em que devem ser executados, isto facilitará o processo de geração de código e interpretação do contrato, como será visto mais adiante.

[6] Estes tipos de *steps* serão detalhados no Capítulo 5

Quando um contrato está totalmente definido as estruturas acima contém as informações necessárias para a geração de código. O algoritmo seguido pelo compilador é apresentado a seguir.

- Gera código com as informações sobre cada variável de contexto contida na **listaVarCtx**;
- Gera código com as informações sobre cada step contido na **listaSteps**, conforme definição do código intermediário;
- Gera código com as informações sobre cada transação definida na **listaTrans**;
- Neste momento começa a geração de código para o fluxo de controle da aplicação. Primeiro é feita uma pesquisa sobre a **listaSteps** para verificar qual o *step*, ou *steps*, de todo o contrato que não possui antecessores. Este *step* será o primeiro executado durante a interpretação, logo, a sua chamada deve ser a primeira nesta seção do código intermediário;
- Para cada tipo de *step* definido na **listaSteps**, começando pelo *step* localizado no passo anterior, deve-se gerar o código de sua chamada conforme estabelecido na *BNF* apresentada na seção anterior. Após a geração do código de cada *step*, deve-se identificar qual o próximo *step* a ser tratado. Esta identificação somente varia quando o tipo [7] do *step* for um **IF**, para dos demais o próximo *step* tratado será aquele contido em sua **listaSuc**. Para *steps* do tipo **IF**, deve-se verificar qual *step* de sua **listaSuc** contém o valor **VERDADEIRO** em seu campo **VFN**. Após isto, é chamado recursivamente o procedimento de geração de código para o "lado" verdadeiro do *step* **IF**, e, posteriormente, para o "lado" **FALSO**.

3.3 O Executor e a Interpretação do Código Intermediário

Nesta seção é apresentada uma idéia bastante simples e inicial de como poderia ser realizada a interpretação de um contrato pelo Sub–Módulo Executor do Gerente de Contratos.

Como visto, a primeira ação realizada pelo interpretador é a criação de entradas no banco de dados para todas as variáveis de contexto. Logo após ele cria uma lista de definições de *steps*, com todas as informações contidas na seção "declaraçãoSteps" e, do mesmo modo, uma lista de declarações de transações. Quando a interpretação atinge a seção "fluxoControle", para cada comando encontrado o controle é desviado para uma rotina específica de tratamento daquele comando, e assim segue até alcançar o último comando do código intermediário.

3.4 Requisitos para o Projeto de uma Interface para o Protótipo do Modelo de Contratos

De acordo com o protótipo do Modelo de Contratos apresentado neste capítulo é possível identificar-se alguns requisitos básicos que devem ser atendidos pelo projeto de Interface do protótipo do Modelo de Contratos, os quais são listados a seguir:

[7] Todos os tipos de *steps* permitidos na construção de contratos encontram-se descritos no Capítulo 5

- Codificação e Compilação de *Steps*:

  A interface deverá permitir que o usuário possa editar um texto com o código de um *step* e, a seguir, acessar o sistema de compilação apropriado, de forma que o resultado final desta operação seja a geração de um código executável para o *step*.

- Acesso ao código dos *steps*:

  Aos usuários deve ser permitido o acesso ao código fonte dos *steps*, sempre que for necessário.

- O acesso aos *steps* será realizado por usuários com um nível de conhecimento técnico considerável:

  Esta característica define que os usuários do sistema que deverão acessar o código dos *steps*, os programadores de *steps*, já possuem conhecimentos sobre linguagens de programação e não encontrariam maiores dificuldades no aprendizado de outras linguagens.

- Definição e Compilação de *Scripts*:

  Ao usuário deve ser permitido definir *scripts* através da composição de *steps* já definidos pelos programadores de *steps*.

- Acesso aos *Scripts* poderá ser realizado por usuários sem um conhecimento técnico muito abrangente:

  Os programadores de *script* não são, necessariamente, pessoas ligadas à área técnica, por isso a dificuldade no aprendizado de uma nova tecnologia pode acarretar algumas desvantagens na utilização do sistema.

- Monitoramento da Execução de Contratos (*scripts*):

  Esta etapa da utilização do sistema também será realizada, provavelmente, por usuários não técnicos.

- Flexibilidade no Acesso à Interface:

  Como identificado nos itens anteriores a interface será utilizada tanto por usuários especializados como por usuários leigos, o que implicará na projeto de uma interface que tente promover o máximo de produtividade de ambos.

4 ESTUDO SOBRE INTERFACES

Atualmente, os usuários de computadores não constituem mais um pequeno grupo formado por profissionais da área de informática. O emprego dos computadores nas mais variadas tarefas fez com que esse grupo aumentasse muito. Tornou-se necessário, então, que estes novos usuários pudessem acessar os sistemas de maneira rápida e simples. Desta forma, o estudo dos processos que envolvem a interação homem-máquina receberam enorme importância.

Também a programação de computadores ainda exige o emprego de pessoas altamente especializadas. Mas a introdução das linguagens visuais facilitou está atividade, tornando-a acessível a usuários menos familiarizados. Como exemplos de linguagens visuais podemos citar (EDWARDS, 1988; GLINERT, 1990):

- The Pinball Construction Set
- ARK
- Interface Builder
- VIP
- DataFlow

Pesquisas vêm sendo desenvolvidas com o objetivo de aumentar o desempenho não somente dos programas de computadores mas também da comunicação dos usuários com estes programas. Para que fim serviria um supercomputador com N processadores, altíssima velocidade de processamento e capacidade de memória se os *softwares* desenvolvidos para ele não fossem capazes de utilizá-lo eficientemente. Associa-se a mesma idéia a área de interfaces. Para prover a utilização eficiente de um software, este deve ser capacitado de uma interface bem projetada de acordo com objetivos preestabelecidos.

Interfaces de usuário constituem aquelas partes do sistema de computação que permitem ao usuário acessar as facilidades oferecidas pelo sistema. Geralmente, interfaces homem-máquina não são tão cuidadosamente projetadas como interfaces máquina-máquina. Neste segundo caso, se todos os diálogos possíveis não forem previamente estabelecidos e conhecidos por ambas as partes a comunicação torna-se inviável.

O Estudo de interfaces com o usuário compreende uma área multidisciplinar, envolvendo aspectos de psicologia, aprendizagem, comunicação, artes visuais, educação, ergonomia, ciência da computação, entre outras. Segundo (PEW; AL., 1212; LIANG, 1987) o projeto de uma interface é muito mais um processo artístico do que ciência.

4.1 Aspectos Multidisciplinares

4.1.1 Psicologia

A idéia de que o Homem adapta-se melhor à execução de atividades complexas quando estas são subdivididas em porções menores é um importante aspecto psicológico que influencia na conduta humana (THIMBLEBY, 1990). A execução de tarefas simples geralmente é finalizada com sucesso, gerando um sentimento de segurança e confiança naquele que executou. Ao utilizar um interface que lhe traz este sentimento, o usuário tende a aumentar sua produtividade, pois não necessita desperdiçar tempo preocupando-se com resultados negativos.

4.1.2 Ergonomia

Quanto aos aspectos ergonômicos pode-se dizer que existem, pelo menos, duas metáforas descrevendo a interação homem-máquina: a metáfora *conversacional* e a metáfora *espacial* (HARTSON; HIX, 1989). Na metáfora conversacional a interface é vista como um diálogo entre o usuário e o computador. O usuário expressa ações a serem executadas em alguma linguagem, o sistema executa tais ações e devolve resultados sobre os quais o usuário aplica uma determinada interpretação e expressa novamente outras ações, seguindo desta maneira até alcançar um determinado objetivo. Através deste tipo de interação os objetos de interesse do usuário são referidos abstratamente (FOLEY et al., 1989), por exemplo, por nome, e as ações somente expressas através de comandos (ou uma linguagem específica) (PIMENTA, 1991).

A metáfora espacial difere significativamente da anterior, pois é permitido ao usuário manipular diretamente os objetos de seu interesse, assim ele tem a ilusão de estar agindo no seu espaço de trabalho sem a interferência do sistema. Como exemplo deste tipo de interfaces pode-se citar aquelas que utilizam o conceito de manipulação direta (SHNEIDERMAN, 1983). Nestas interfaces os objetos são visíveis e a manipulação dos mesmos ocorre através de ações reais (como apontar, mover, etc.).

Um exemplo simples de manipulação direta de um objeto gráfico utilizando um pacote gráfico de desenho é apresentado nas Figuras 4.1 e 4.2.

Na figura 4.1 o retângulo não é largo o suficiente para envolver o círculo desenhado no centro. Através de um sistema de manipulação direta o usuário pode "clicar"em um dos pontos do retângulo e diretamente aumentar seu tamanho com a ajuda do *mouse*.

Freqüentemente associado ao conceito de manipulação direta está a idéia de diálogo multitarefa, referindo-se a multiplicidade de tarefas providas em um dado momento em um diálogo. Este tipo de diálogo é chamado de não-sequencial, ou assíncrono, pois o usuário pode escolher a ordem em que deseja realizar as diferentes tarefas. Um exemplo é mostrado na figura 4.3.

Através desta *popup*[1] o usuário deve fornecer as informações sobre um arquivo a ser carregado pelo sistema. Após a apresentação da janela, o usuário pode optar por digitar todo o caminho [2] e o nome do arquivo que deseja abrir, ou selecionar o diretório e arquivo a partir da lista, ou ainda cancelar a operação.

As interfaces que utilizam a metáfora espacial apresentam maior **facilidade de aprendizado ao usuário**[3].

[1] janela aberta somente em algumas situações durante a execução de uma aplicação
[2] *path*, em inglês
[3] *usability*

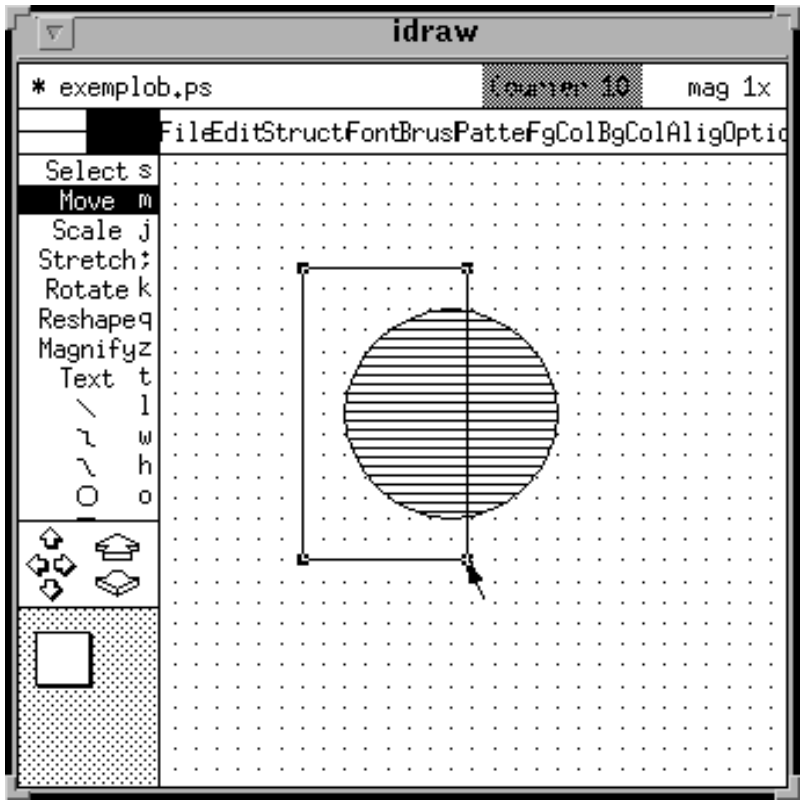

Figura 4.1: Círculo não envolvido totalmente pelo retângulo

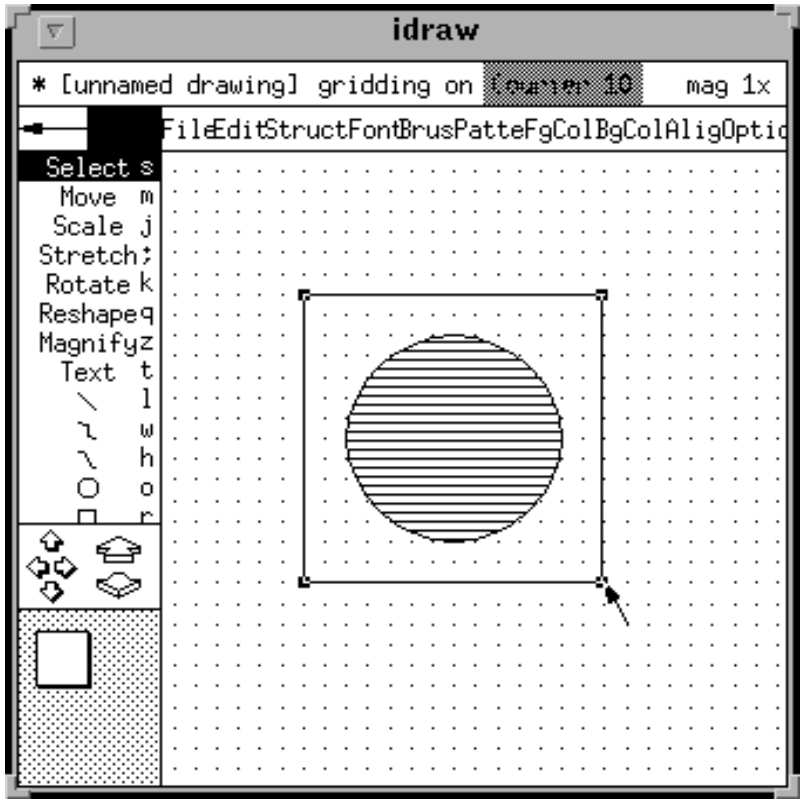

Figura 4.2: Círculo totalmente envolvido pelo retângulo

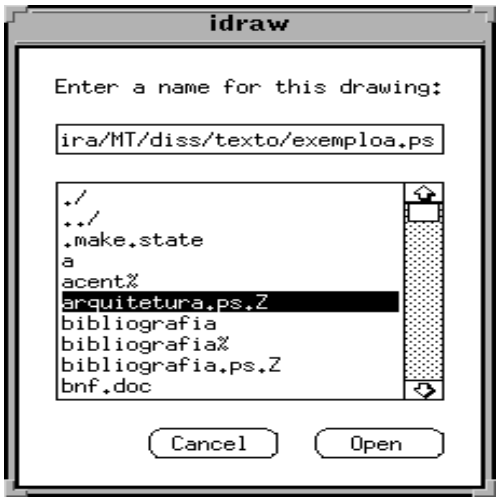

Figura 4.3: Popup para abertura de arquivo

4.1.3 Comunicação

Segundo Edward Sapir e Benjamin Whorf (THIMBLEBY, 1990) (hipótese de Sapir-Whorf), a maneira pela qual o homem percebe o mundo depende da linguagem que lhe está disponível. Ou seja, a linguagem controla o modo com que ele pensa. Por exemplo, um programador da linguagem FORTRAN jamais pensaria em uma solução recursiva para um determinado problema, mesmo que a recursão fosse a solução natural para o mesmo.

4.1.4 Aspectos de Artes Gráficas

"Uma figura equivale a milhares de palavras" (THIMBLEBY, 1990). Esta frase expressa a constatação de psicólogos que tentam demonstrar a vantagem da utilização de figuras e imagens sobre as palavras e números na representação de informações. Por exemplo, erros de edição podem ser visualizados em menos tempo se uma representação gráfica apropriada for utilizada.

4.2 Melhorando a Qualidade das Interfaces

O reprojeto de interfaces de usuário com base em testes iterativos – iterando pelo menos até a terceira versão do sistema – pode melhorar substancialmente a qualidade da interface (NIELSEN, 1992, 1993).

Cada iteração deveria resultar em uma interface melhor do que a obtida na versão anterior, mas isto não é sempre verdade. A Figura 4.4 (NIELSEN, 1993) mostra a natureza genérica do projeto iterativo, apesar de ainda não existir um número suficiente de estudos de casos documentados para estimar esta curva precisamente.

As primeiras iterações irão geralmente resultar nas principais melhoras da interface, pois é nesta fase que os projetistas percebem e consertam os erros mais graves. As últimas iterações apresentam, progressivamente, menor potencial para melhoras porque a maioria das dificuldades já foram superadas. Na verdade não se sabe se este processo é infinito ou não. O que normalmente acontece é a reconcepção da interface após um elevado número de iterações.

A qualidade global de um sistema é formada pela soma de muitos atributos, onde um

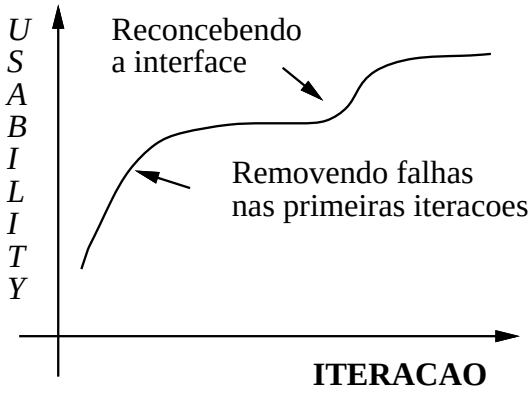

Figura 4.4: Relação entre o projeto iterativo e a qualidade das interfaces

deles é a qualidade da interface. O sistema deve satisfazer as necessidades previstas pela atividade desempenhada pelo usuário e assim permitir que ele produza resultados de alta qualidade. A qualidade pode ser definida em função de vários atributos citenielsen92, a saber:

- Eficiente: minimizar o esforço requerido para se realizar determinada tarefa;
- Conveniente: prover fáceis técnicas de acesso as suas operações;
- Flexível: prover mais de uma alternativa para a execução de determinadas operações;
- Consistente: minimizar o esforço de aprendizagem por parte do usuário, ou seja, a interface deve seguir um padrão já conhecido pelo usuário. Sendo assim os usuários podem facilmente passar do estágio em que não conheciam nada a respeito da interface para o estágio em que produzem algum trabalho;
- Natural: permitir a interação da forma mais natural possível, não exigindo que o usuário tenha conhecimento de terminologia não referente a tarefa que está realizando;
- Diversa: suportar tanto usuários experientes como não experientes, de forma que o usuário experiente obtenha um alto nível de produtividade;
- Complacente com os erros cometidos pelos usuários, permitindo que estes sejam facilmente recuperados e não causem "catástrofes";
- Satisfação: a interface deve facilitar a execução das operações, permitindo que o usuário sinta-se satisfeito ao observar seu sucesso.

Nem todos estes atributos têm o mesmo peso, isto depende muito da aplicação. Por exemplo, em um sistema de telefonia, cada redução de um segundo no tempo de trabalho por chamada para os seus operadores pode ocasionar a economia de milhões de dólares por ano. Entretanto, para aplicações que monitoram processos industriais é necessária uma baixa freqüência de erros. Para cada sistema um determinado conjunto de atributos pode ser importante, então, procura-se avaliar a qualidade da interface através destes atributos somente. Exemplos de como medir a melhora da qualidade das interfaces através de um projeto iterativo podem ser encontrados em (NIELSEN, 1993).

5 INTERFACE GRÁFICA

O presente Capítulo descreve a interface gráfica do protótipo do sistema gerenciador de transações longas de banco de dados apresentado no Capítulo 3. O Projeto da interface procura considerar os requisitos identificados no Capítulo 3 e analisados no Capítulo 4, onde discutiu-se sobre Interfaces.

A interface foi implementada através dos pacotes gráficos XView versão 3.0 (X Window-System-based Visual Environment for Workstations) (HELLER, 1990), um pacote gráfico que suporta aplicações gráficas e interativas, as quais executam sob o sistema X Window System e Slingshot versão 2.0, uma extensão do pacote XView. O modelo de interface será baseado no padrão de interfaces gráficas de usuário OPEN LOOK desenvolvido pela Sun Microsystems e At&T (HELLER, 1990).

É através da interface que ocorre toda a comunicação entre o usuário e o sistema. Ela é composta por um conjunto de janelas, cada uma delas representando um determinado módulo. Existem três módulos principais:

- Interface PRINCIPAL, de onde é possível disparar a execução dos demais módulos;
- Interface DEFINIÇÃO, que permite ao usuário definir os *steps* e a transação de longa duração;
- Interface EXECUÇÃO, a qual permite o monitoramento da execução da transação definida anteriormente.

A comunicação entre os módulos ocorre da seguinte forma: o módulo PRINCIPAL possibilita a criação de diversas instâncias dos Sub-Módulos INTERFACE E EXECUÇÃO, os quais existem independentemente. A Figura 5.1 mostra esta relação:

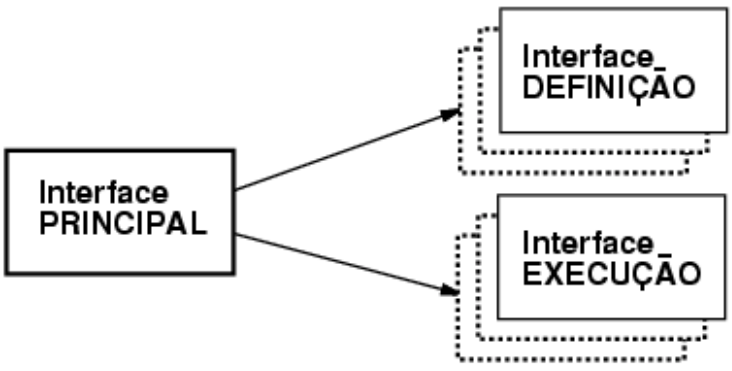

Figura 5.1: Relação entre os Sub–Módulos de Interface

A seguir, cada um dos Módulos da interface de usuário, PRINCIPAL, DEFINIÇÃO E EXECUÇÃO, é descrito detalhadamente.

5.1 Interface Principal

Ao iniciar-se o sistema, uma primeira janela (uma *PopUp Window*) contendo informações genéricas sobre o mesmo é apresentada ao usuário, a mensagem é mostrada na Figura 5.2. Esta janela pode ser fechada pelo usuário a qualquer momento, através do botão "OK", assim como pode ser reaberta através do botão Info da interface PRINCIPAL.

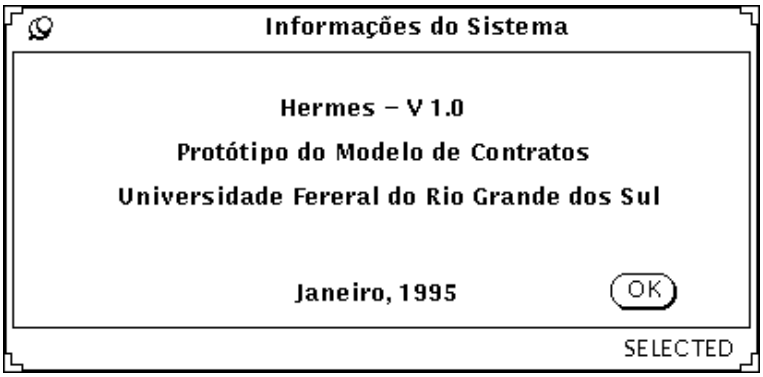

Figura 5.2: Janela de Informações do Sistema

O Módulo PRINCIPAL do sistema de interface, mostrado na Figura 5.3, é responsável pelo gerenciamento dos dois Sub–Módulos, DEFINIÇÃO e EXECUÇÃO. Como pode ser visto, existem ícones associados à janela da interface PRINCIPAL, os quais representam os outros dois módulos. Para iniciar a execução de qualquer um deles deve-se "clicar" duas vezes sobre o respectivo ícone e, consequentemente, a interface referente ao módulo é apresentada. Se, durante sua execução, o usuário fechar a janela principal da interface, automaticamente aparecerá um ícone na janela da interface PRINCIPAL, representando a execução suspensa do módulo. Para retomá-la basta clicar duas vezes sobre o ícone que a representa.

A interface também é provida de um "Sistema de Ajuda", estruturado em forma de hipertexto, que ensina o usuário a utilizar o *software*. Através do botão Ajuda é possível acessar o nodo raíz deste hipertexto e, a partir dele, pode-se alcançar qualquer outra tela de ajuda desejada. Ao mesmo tempo, o "Sistema de Ajuda" também é sensitivo, ou seja, se o cursor estiver posicionado sobre algum objeto da interface e o botão de "Ajuda" for pressionado, o sistema de hipertexto é acionado, mas não a partir do nodo raíz e sim a partir daquele que contiver o texto relacionado ao objeto apontado.

A interface também é provida de um "Sistema de Ajuda", estruturado em forma de hipertexto, que ensina o usuário a utilizar o *software*. Através do botão Ajuda é possível acessar o nodo raíz deste hipertexto e, a partir dele, pode-se alcançar qualquer outra tela de ajuda desejada. Ao mesmo tempo, o "Sistema de Ajuda" também é sensitivo, ou seja, se o cursor estiver posicionado sobre algum objeto da interface e o botão de "Ajuda" for pressionado, o sistema de hipertexto é acionado, mas não a partir do nodo raíz e sim a partir daquele que contiver o texto relacionado ao objeto apontado.

O botão Fim serve para o encerramento da interface, neste caso, todas as instâncias de interfaces dos módulos DEFINIÇÃO ou EXECUÇÃO que se encontram executando são

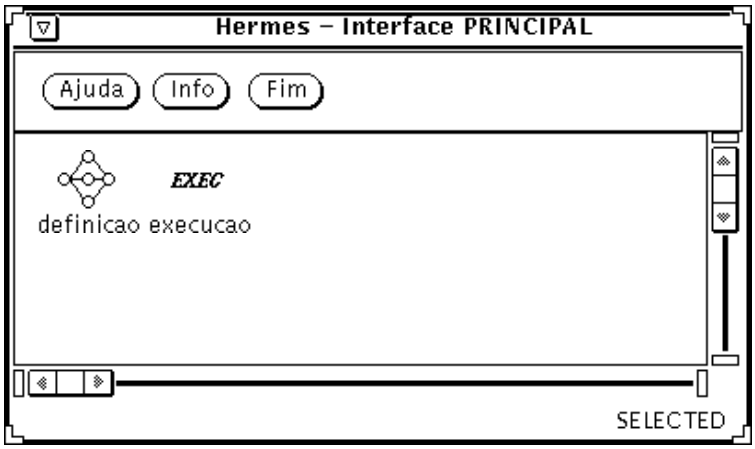

Figura 5.3: Interface PRINCIPAL

suspensas, mas apenas as interfaces, pois a atividade que cada um destes módulos estava executando continua normalmente. Neste caso é apresentada ao usuário uma janela, como a mostrada na Figura 5.4 perguntado se ele deseja salvar as operações ainda não salvas ou se deseja terminar a interface mesmo assim.

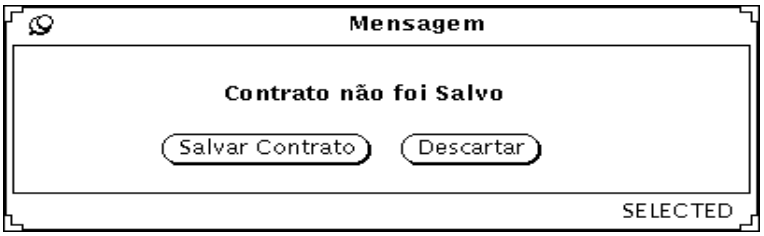

Figura 5.4: Janela de Salvamento

5.2 Interface Definição

Quando, na janela principal, o usuário "clicar"duas vezes (*double click*) no ícone correspondente ao módulo DEFINIÇÃO a interface mostrada na Figura 5.5 é apresentada. É através das facilidades providas por esta interface que o usuário define um contrato.

A interface apresenta três áreas distintas:

- Área de Botões: na parte superior aparecem os botões destinados à execução de todas as funções da interface de definição, a descrição de cada uma destas funções é apresentada a seguir;
- Gerente de Arquivos: a região intermediária funciona basicamente como um gerenciador de arquivos. Nesta região aparecem ícones, representações gráficas de cada um dos arquivos envolvidos na definição de um contrato;
- Editor Gráfico: e a última parte é o editor gráfico provido de uma linguagem visual, a qual permite a definição completa de um contrato.

Agora, as funções da interface do Sub–Módulo DEFINIÇÃO serão apresentadas, assim como suas alternativas de execução.

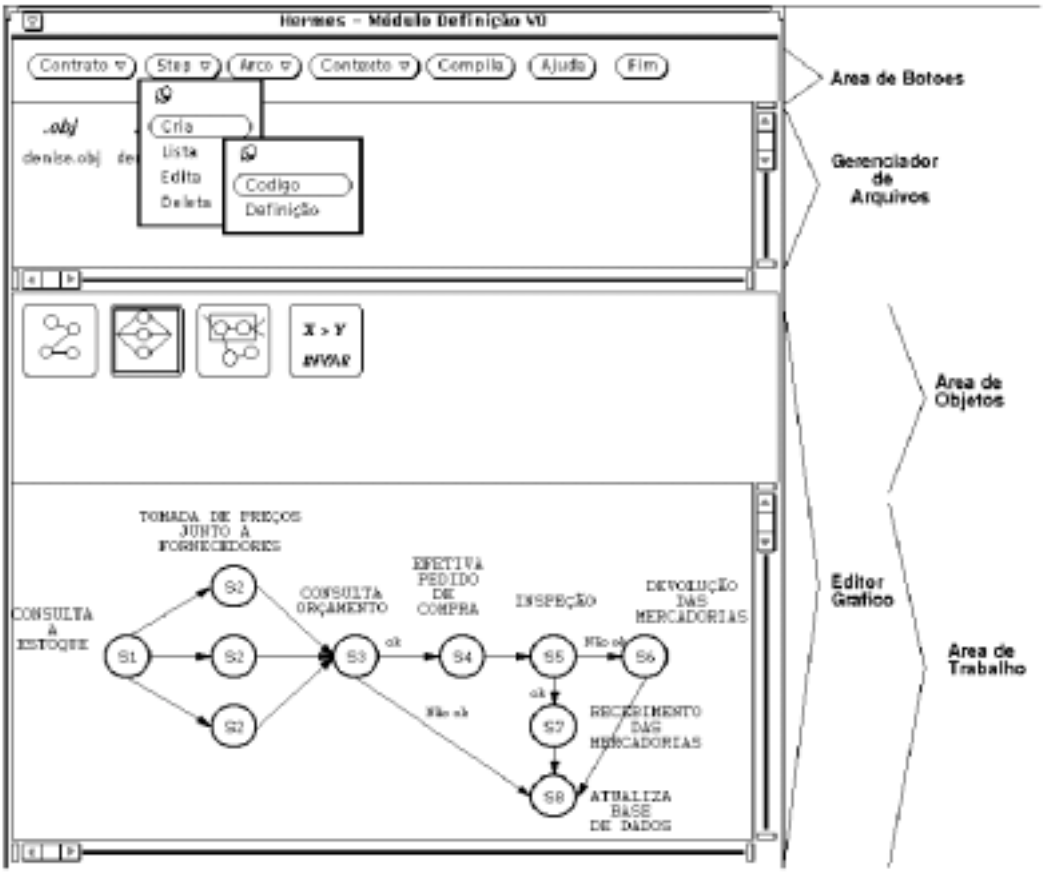

Figura 5.5: Interface Definição

5.2.1 Contrato

O gerenciamento da definição de um Contrato é feito pelas opções contidas no menu Contrato, mostradas na Figura 5.6, as quais são: Carregar, Criar, Salvar, Salvar Como e Deletar.

Figura 5.6: Menu Contrato

A criação de um contrato pode ser feita através das opções apresentadas no menu Criar, mostradas na Figura 5.7: Novo, Cópia e União.

Figura 5.7: Menu Criar

Criar–Novo

Na criação de um contrato novo, as listas, **listaVarCtx**, **listaSteps** e **listaTrans** mencionadas no Capítulo 3, são constituídas. A **listaSteps** armazena a descrição da ordem parcial de *steps* a ser definida no editor gráfico; a **listaVarCtx** armazena a definição de todas as variáveis de contexto; e a **listaTrans** é destinada a definição das transações e seus componentes.

Criar–Cópia

Na criação de um contrato através de uma cópia, é apresentada ao usuário uma lista de nomes de contratos, como a mostrada na Figura 5.8, da qual ele deve escolher apenas um. Logo após, as estruturas referentes ao contrato escolhido são duplicadas. A opção "Cópia" é útil quando um contrato novo é apenas a extensão de um contrato já existente ou apresenta poucas modificações em relação a ele.

Criar–União

A opção "União" possibilita que um contrato novo seja a união de um ou mais contratos já existentes. Para isto é apresentada ao usuário uma lista com nomes de contratos e ele deve escolher quais contratos deseja utilizar. Posteriormente, as listas principais são criadas e os grafos que representam os *scripts* dos contratos escolhidos são mostrados. O

usuário pode, então, selecionar partes destes *scripts*, copiar para a sua área de trabalho e assim montar o seu próprio contrato.

Na opção União a lista de *steps* é gerada automaticamente, sendo suas entradas compostas apenas pelas definições dos *steps* utilizados no novo contrato. Estas definições são retiradas das listas de *steps* dos contratos já existentes. O mesmo ocorre com a lista de variáveis de contexto, o sistema analisa quais variáveis de contexto de cada contrato estão sendo reutilizadas e acrescenta à lista do contrato em criação.

Quando contextos de contratos diferentes, que estão sendo compostos, possuem variáveis com nomes iguais, o usuário deve esclarecer se deseja que alguma das definições permaneça ou se prefere alterar. As Figuras 5.9 e 5.10 mostram os dois casos onde pode ocorrer esta situação. No primeiro caso, podem existir variáveis de tipos diferentes , mas com nomes iguais. E no segundo as variáveis tem o mesmo tipo e nome. Em ambos os casos, se o usuário optar por alterar os nomes das variáveis, estes nomes também serão alterados na lista de *steps* e de variáveis de contexto.

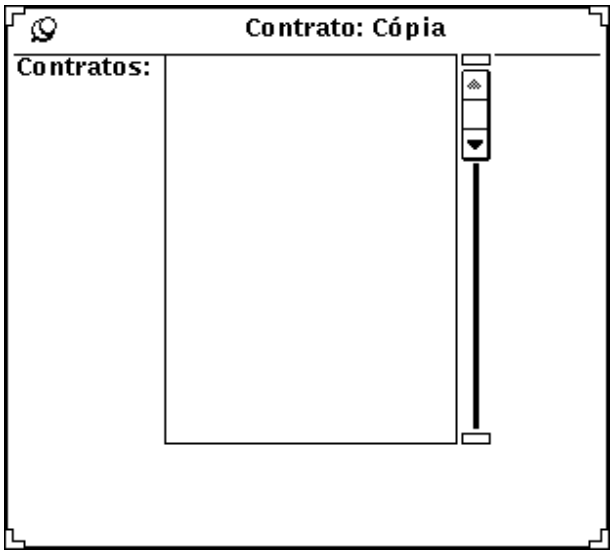

Figura 5.8: Janela da opção Cópia

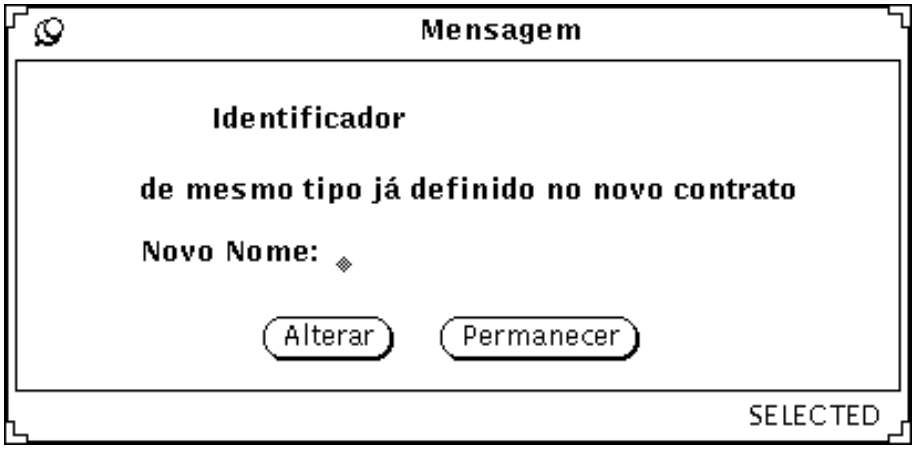

Figura 5.9: Mensagem para variáveis de contexto de mesmo nome e tipo

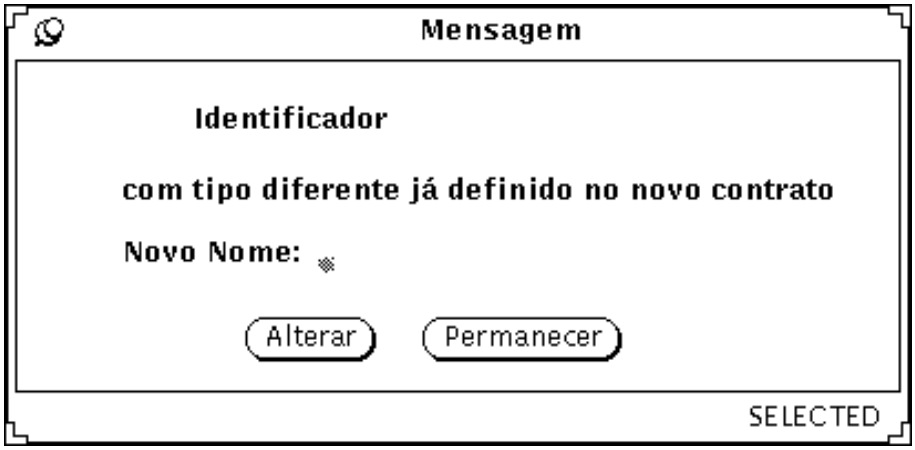

Figura 5.10: Mensagem para variáveis de contexto com o mesmo nome e tipos diferentes

Carregar

A opção “Carregar” indica ao sistema qual contrato encontra-se em definição no momento e causa a apresentação do grafo correspondente no editor gráfico. Um contrato também pode ser carregado trazendo-se um ícone do arquivo com extensão “scr” para a área do editor ou clicando-se duas vezes sobre o mesmo.

Salvar e Salvar Como

As opções “Salvar” e “Salvar Como” servem para que alterações feitas em um determinado contrato sejam atualizadas efetivamente. Quando executa-se um salvamento, as informações contidas nas listas são gravadas em um arquivo. A identificação deste arquivo é composta pelo nome fornecido pelo usuário e a extensão “scr”.

Deletar

A opção Deletar permite descartar um determinado contrato.

5.2.2 Variáveis de Contexto

O gerenciamento das variáveis de contexto, quanto ao aspecto definição, é feito através da opção Contexto. Para a definição de uma nova variável, é apresentada uma janela solicitando informações sobre a variável que devem ser respondidas pelo usuário, consequentemente, é criada uma nova entrada na lista de variáveis de contexto com a respectiva definição. A figura 5.11 mostra a definição de variáveis.

Também é permitida a edição de qualquer definição anteriormente realizada, basta selecionar a variável da lista de variáveis definidas, alterar sua definição e acionar o botão “OK”. A listagem de todas as variáveis de contexto já definidas aparece nesta lista. Os atributos de uma definição são o nome e tipo da variável.

5.2.3 Step

Com a opção “*Step*” é feito o gerenciamento dos *steps* de um contrato. Na Figura 5.12 é mostrado o menu de opções referente à manipulação de *Steps*.

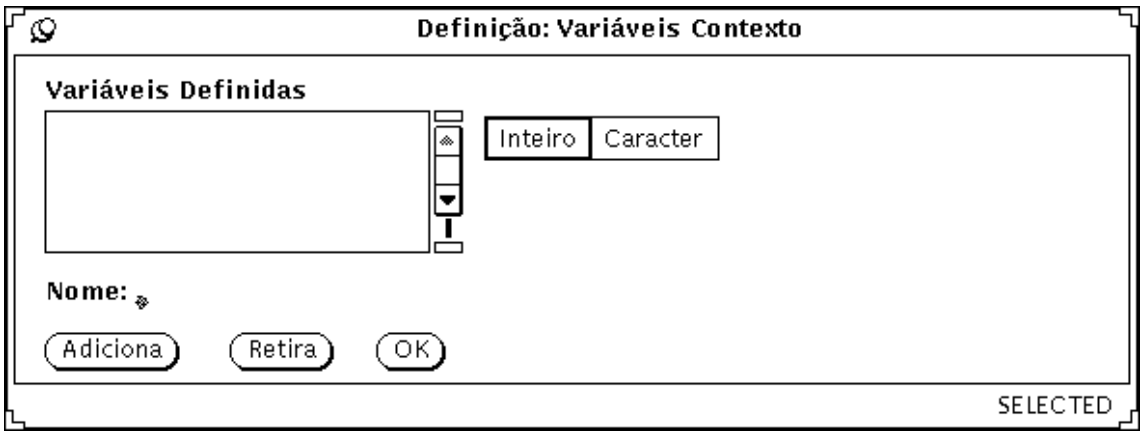

Figura 5.11: Janela Definição: Variáveis de Contexto

Figura 5.12: Menu Step

Step–Código

Esta opção destina-se aos programadores de *step*, os quais são responsáveis por escrever o código fonte dos *steps* e gerar seu código executável. Estas tarefas são providas pelas sub–opções Edita e Compila.

Para a edição do código de um *step* o programador é auxiliado por um editor de textos, através do qual ele pode realizar qualquer operação padrão, como edição, salvamento, etc. A compilação (e pré-compilação, caso seja necessária) será realizada através da chamada de programas compiladores (e pré-compiladores) oferecidos pelo sistema de computação em uso.

Step–Define

É através desta opção que o programador do *script* informa ao sistema a definição de um *step*. A Figura 5.13 mostra as informações que devem ser fornecidas, as quais são listadas abaixo:

- **Nome Step**: deve-se fornecer um nome que identifique o *step* no *script*;
- **Nome Programa**: nome do arquivo, executável, que contém o código referente ao *step*;
- **Endereço na Rede**: endereço do nodo da rede onde o *step* deve ser executado;
- **Par Entrada ou Par Saída**: a opção “Par Entrada” deve ser selecionada para a definição, logo a seguir, dos parâmetros de entrada do *step* e o mesmo para a opção “Par Saída”;

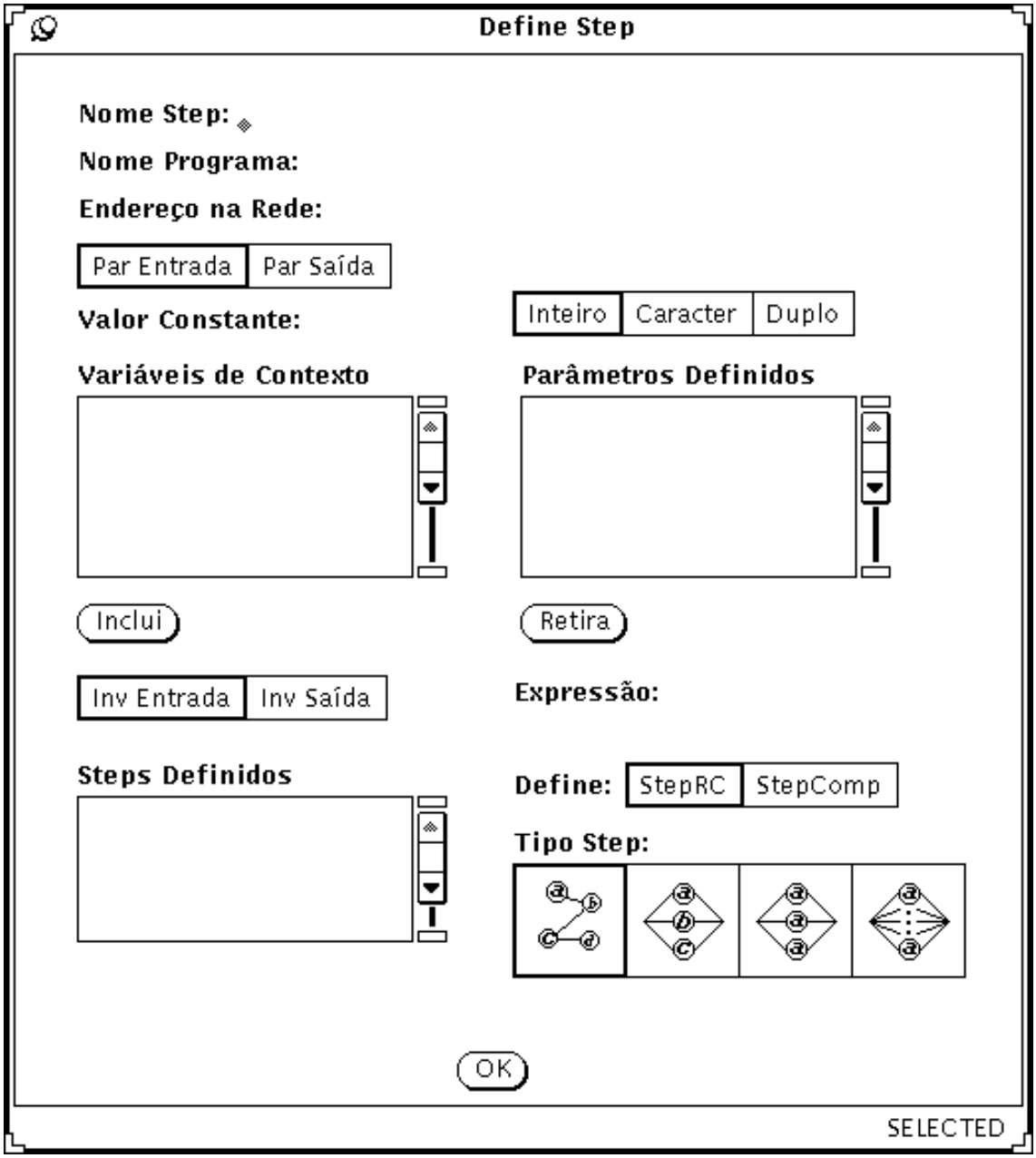

Figura 5.13: Janela Define Step

- **Valor Constante**: parâmetro de entrada ou saída;
- **Variáveis de Contexto**: lista de todas as variáveis de contexto que podem servir como parâmetros de entrada ou saída;
- **Parâmetros Definidos**: lista de parâmetros já definidos para o *step*;
- **Inclui**: inclui uma variável de contexto ou valor constante na lista de parâmetros do *step*;
- **Retira**: exclui determinado parâmetro já definido;
- **Inv Entrada ou Inv Saída**: indicam se o campo "Expressão" se refere à invariante de entrada ou de saída do *step*;
- **Define: StepRC ou StepComp**: indica a definição do *step* de resolução de conflito ou do *step* compensatório, os quais podem ser selecionados, diretamente, do campo "Steps Definidos" ou podem ser criados;
- **OK**: efetiva a definição, alterando as estruturas apropriadas.

Step–Lista

Através da opção Lista pode-se verificar todos os *steps* já definidos para o contrato. A Figura 5.14 mostra a janela ativada no momento em que esta opção é selecionada. Se algum dos *steps* da lista for selecionado, é ativada a janela "Definição: Step". Se as informações forem alteradas e o botão "OK" for acionado, as alterações serão efetivamente realizadas.

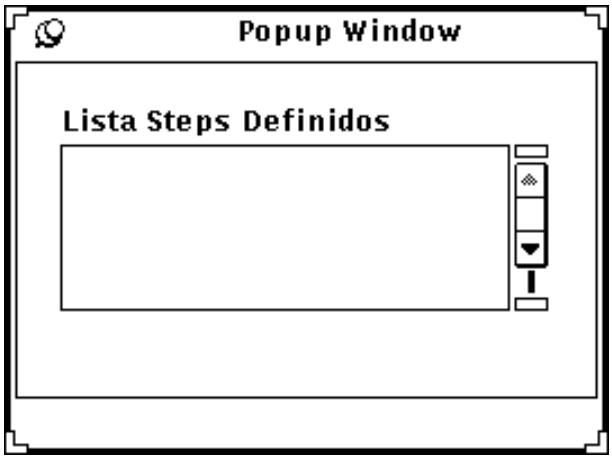

Figura 5.14: Janela com Lista de Steps Definidos

Step–Cria

Ao acionar-se esta opção é criada a identificação de um *step* na área de trabalho do Editor Gráfico e apresentada uma janela solicitando a definição do comando. Como mostra a Figura 5.15, existem quatro tipos básicos de *steps* que podem ser utilizados na construção de um *script*, a saber:

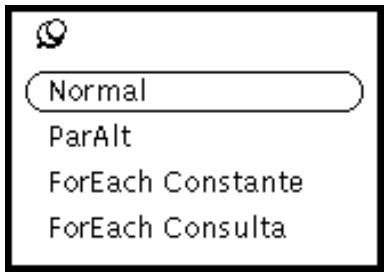

Figura 5.15: Menu Step–Criar

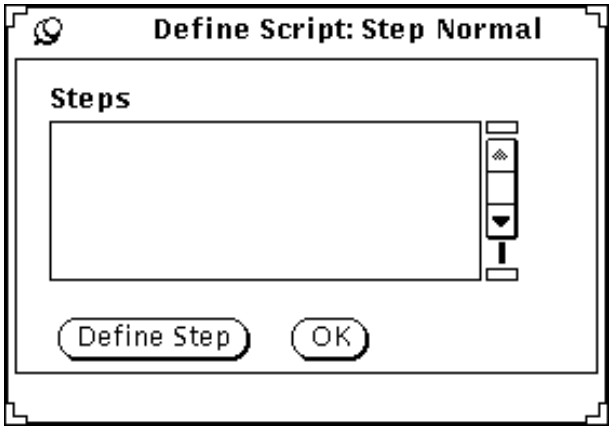

Figura 5.16: Janela para criação de um Step NORMAL

- NORMAL:

  Execução de apenas um *step*, conforme foi definido. A Figura 5.16 mostra a janela aberta para a criação de um *step* deste tipo.

  Deve ser selecionado apenas um *step* da lista de *steps* definidos. Neste caso, é criada uma identificação para o *step* na área de trabalho do Editor Gráfico. Outra alternativa é a definição de um novo *step*, para isto abre-se uma janela "Definição: Step", como a apresentada na opção Define, e, após o usuário concluir a definição, retorna-se à janela mostrada na Figura 5.16.

- FOR_EACH_CONST:

  Execução de um ou mais *steps* em paralelo, de acordo com o número de parâmetros constantes enviados ao comando. Tais parâmetros podem ser tanto valores constantes como variáveis de contexto e devem coincidir com os parâmetros de entrada dos *steps* envolvidos. A Figura 5.17 mostra a janela para a definição de um *step* deste tipo.

  A descrição da janela "Define Script: Step ForEach Constante" é a seguinte:

  – as opções "Par" ou "Alt" definem se o próximo comando, na seqüência de execução do *script*, deve ter sua execução iniciada quando todos os *steps* sendo executados em paralelo tiverem sua execução terminada (opção "Par") ou quando apenas um deles tiver acabado (opção "alt");

  – o campo "Valor Constante" serve para a definição de um parâmetro para o comando que seja uma constante, o tipo da constante deve ser identificado no campo ao lado;

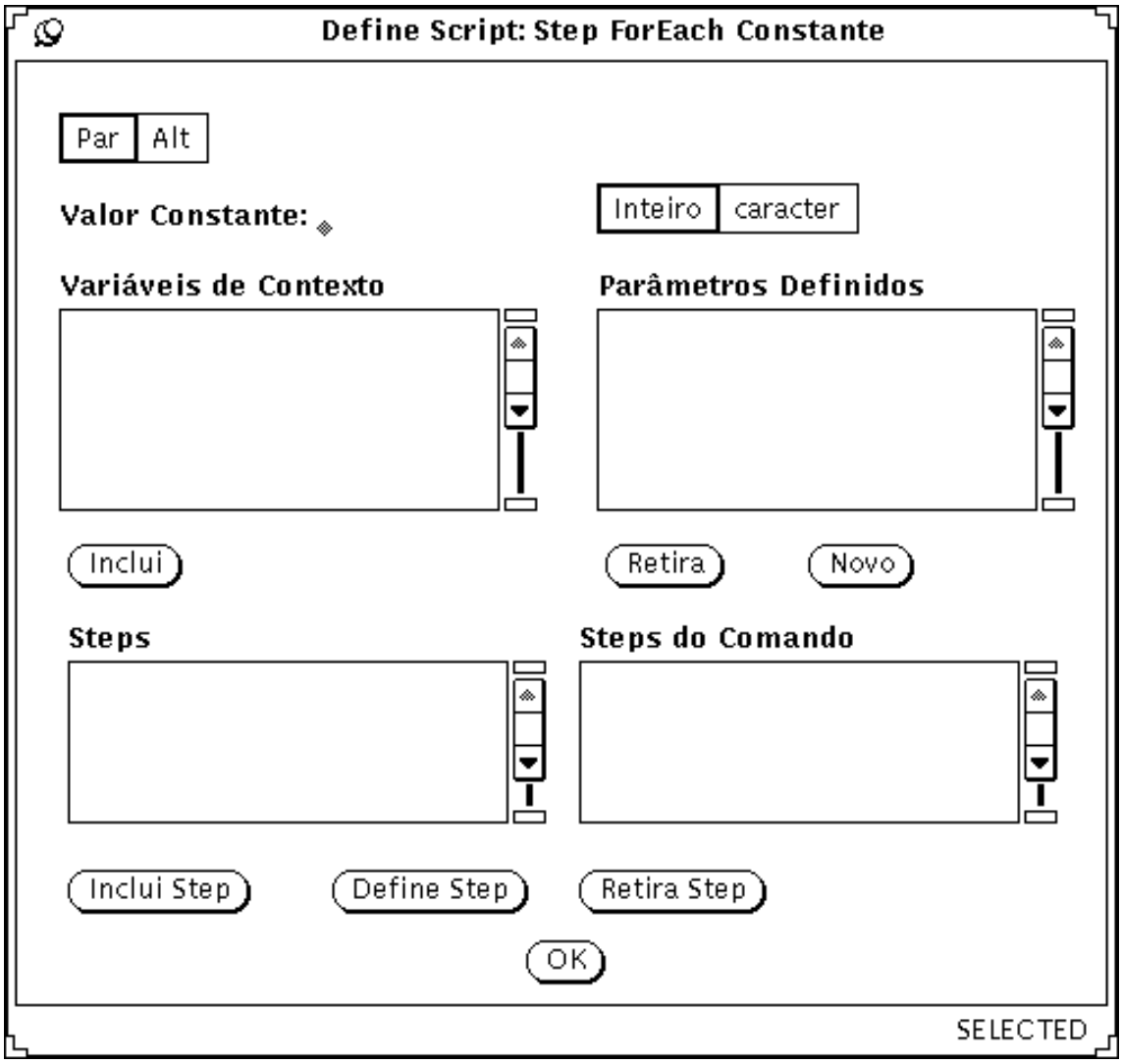

Figura 5.17: Janela para criação de um Step ForEach Constante

  - o campo “Variáveis de Contexto” apresenta a lista de todas as variáveis de contexto definidas para aquele contrato. O usuário pode selecioná-las para que participem como parâmetros do comando;
  - o botão “Inclui” inclui o valor do campo “Valor Constante” ou a variável que estiver selecionada nos parâmetros definidos para o comando;
  - o botão “Retira” exclui um parâmetro selecionado no campo “Parâmetros Definidos”;
  - o botão “Novo” deve ser acionado sempre que o usuário desejar definir um novo conjunto de parâmetros para o comando.
  - o campo “Steps” apresenta a lista de steps já definidos para o contrato, o usuário pode, então, selecionar alguns destes *steps* para fazerem parte do comando;
  - o campo “Steps do Comando” mostra os *steps* escolhidos pelo usuário para fazerem parte do comando;
  - o botão “Inclui Step” inclui determinado *step*, selecionado na lista apresentada no campo “Steps”, para participar do comando;
  - o botão “Define Step” permite a definição de um novo *step* para participar do comando;
  - o botão “Retira Step” exclui determinado *step* do comando;
  - o botão “OK” efetiva a definição realizada, alterando as estruturas apropriadas.

- FOR_EACH_CONS:

  Execução de um ou mais *steps*, de acordo com o número de valores retornados de uma consulta realizada a um banco de dados local. A Figura 5.18 mostra a definição de um comando deste tipo.

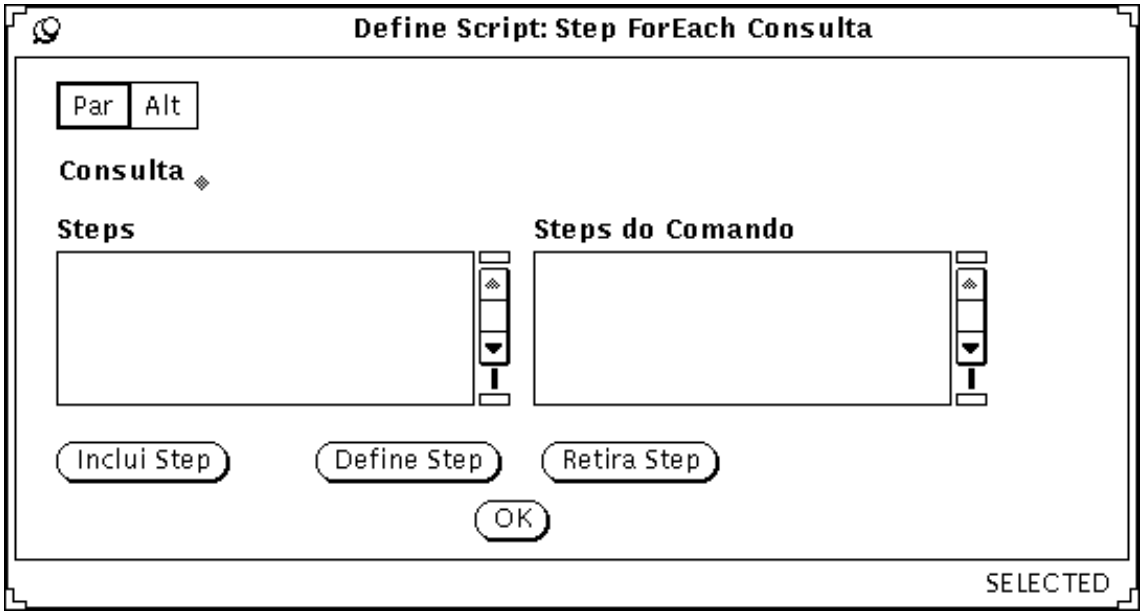

Figura 5.18: Janela para a criação de um Step do tipo ForEach Consulta

Neste tipo de comando os identificadores dos atributos da relação que estão sendo selecionados devem ser os mesmos utilizados como parâmetros de entrada no *step*.

O funcionamento da janela “Definição Script: Step ForEach Consulta” é semelhante ao da janela apresentada no comando anterior. O único campo diferente é o “Consulta” onde o usuário deve definir a consulta que realizará ao banco de dados;

- PAR_ALT:

  Com este comando vários *steps* diferentes podem ser executados em paralelo. Na janela “Definição Script: Step PArAlt” (Figura 5.19), apenas devem ser selecionados os *steps* já definidos ou pode-se optar pela definição de um novo *step* para participar do comando.

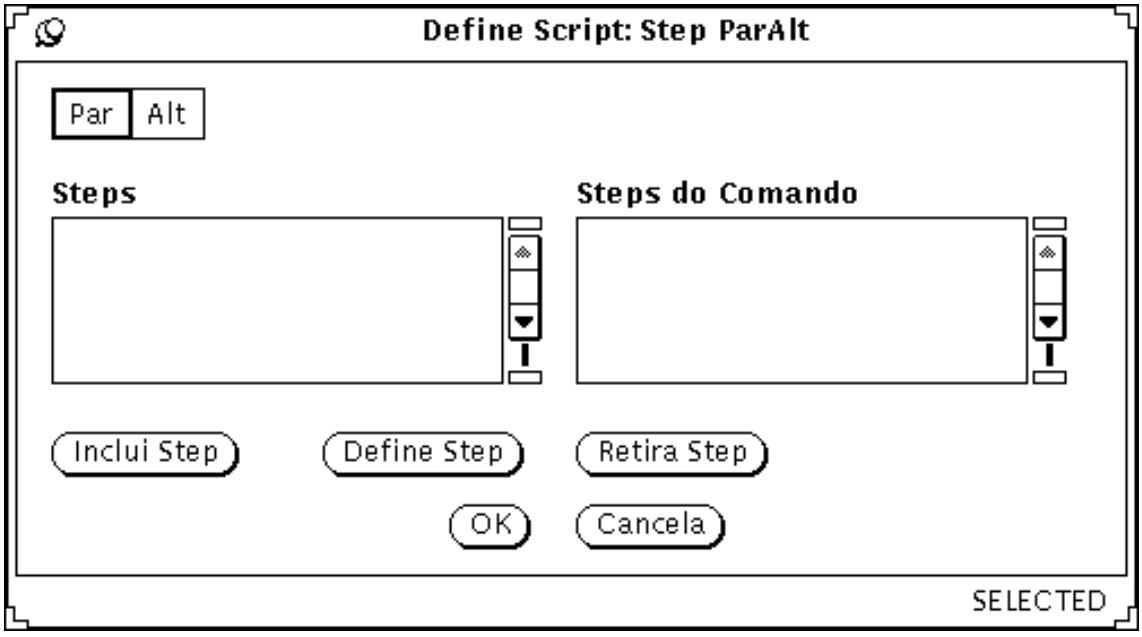

Figura 5.19: Janela Definição Script: Step ParAlt

Step–Deleta

Quando a opção Deleta é utilizada, a entrada referente ao *step* é retirada da lista de definição e a representação gráfica é alterada, sendo retirado o símbolo do *step* do grafo assim como todos os arcos que chegam ou partem dele.

5.2.4 Arco

A manipulação das conexões entre *steps* é feita através da opção Arco, onde devem ser indicados os *steps* a serem conectados ou desconectados.

Arco–Cria

Através da janela “Define Arco”, figura 5.20, o usuário indica os *steps* fonte e destino das conexões que está realizando. Existem duas opções para a criação de arcos, como mostra a Figura 5.21. Em ambas deve-se indicar tanto o *step* fonte como o destino.

A opção “Falha” indica que existe um comando IF implícito e que depende do resultado da execução do *step* fonte. Se o resultado for negativo, ou seja, o *step* não conseguiu realizar a tarefa para a qual foi definido, então, o *step* destino deverá ser executado. Para que a definição do *script* esteja correta, deve existir também um arco do tipo “Normal” partindo do mesmo *step* fonte para um outro *step* qualquer.

Figura 5.20: Menu Arco

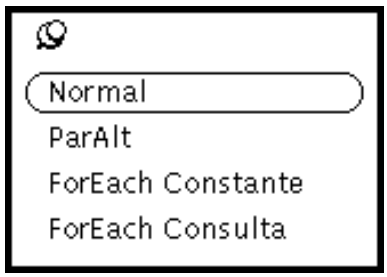

Figura 5.21: Menu Cria–Arco

Arco–Deleta

A remoção de um arco ocorre através desta opção.

5.2.5 Transação

Através desta opção são definidas as transações, englobando *steps*. Para isto, é apresentada uma janela mostrando a lista de *steps* já incluídos no *sript*, o usuário deve, então, selecionar aqueles que devem participar da transação. Para o exemplo de script apresentado no capítulo 2, seriam apresentados todos os steps e o usuário escolheria os *steps* S_4 e S_5 e indicaria que eles formam uma transação. O mesmo seria feito em relação aos *steps* S_6 e S_7.

5.2.6 Compila

Através da opção Compila realiza-se a compilação de um contrato e conseqüente geração de código, código este que será, posteriormente, interpretado pelo módulo executor.

5.2.7 Ajuda e Fim

Como já mencionado, através da opção Ajuda inicia-se o "Sistema de Ajuda" organizado em forma de hipertexto. E a opção Fim finaliza o módulo DEFINIÇÃO, verificando se havia algum contrato em definição não salvo. Caso isto seja verdade o usuário é avisado, através da janela mostrada na Figura 5.22, e fica ao seu encargo tomar a decisão apropriada.

5.2.8 Editor Gráfico

Como descrito neste Capítulo, o Editor Gráfico permite que a definição do *script* de um contrato seja feita através de um grafo. O Editor é composto por duas partes: a primeira é a área onde ficam os objetos da linguagem visual, mostrados na Figura 5.23; a outra é a área de trabalho, onde o usuário edita e visualiza o grafo em definição.

A linguagem visual é composta pelos objetos mostrados na Figura 5.23. Cada um dos objetos pode realizar uma ou duas atividades. A seguir são apresentadas as funcionali-

Figura 5.22: Janela Informando que Contrato não foi Gravado

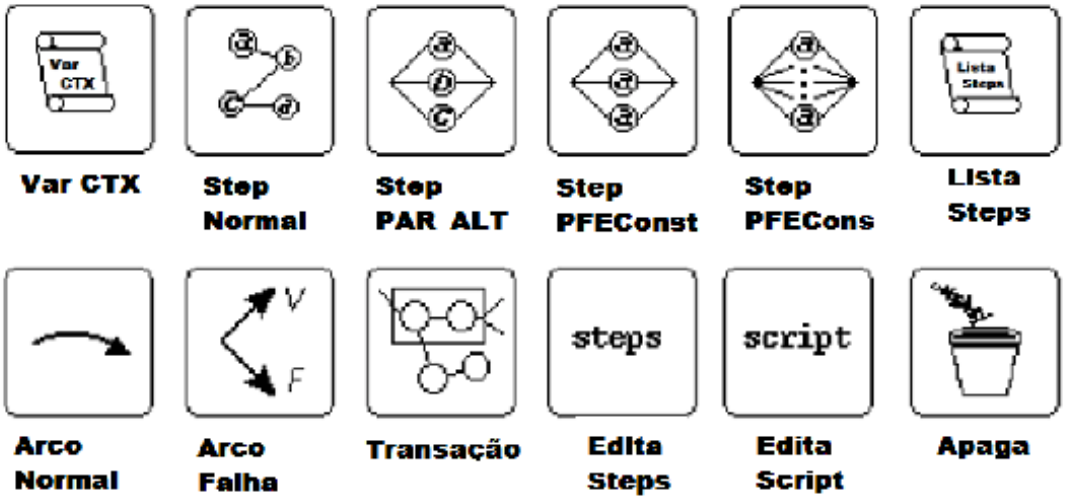

Figura 5.23: Objetos da Linguagem Visual

dades de cada um deles.

Objeto Edita Steps

Indica á interface que estão sendo realizadas definições de *steps*, portanto, permanecem ativos apenas os objetos Var CTX, Step Normal, Lista Steps e Edita Script. A seguir é descrita a função de cada um destes objetos:

- Objeto Var CTX: Definição de variáveis de contexto. Apresenta a mesma funcionalidade do botão “Contexto” localizado na área de botões;
- Objeto Step Normal: Definição de *Steps*. Apresenta a mesma funcionalidade da opção “Step–Define”;
- Objeto Lista Steps: Lista de *Steps*. Mesma funcionalidade apresentada pela opção “Step–Lista”;
- Objeto Edita Script: indica à Interface que começará a definição do grafo que representa o *script*.

Objeto Edita Script

Quando o objeto edita Script é acionado todos os outros tornam-se ativos. A seguir é descrita a função de cada objeto que ainda não foi definido ou que teve sua definição alterada:

- Objeto Step Normal: criação no grafo de um *step* do tipo NORMAL, mesma funcionalidade apresentada pela opção “Step–Cria–Normal”. O objeto deve ser “arrastado” até área de trabalho do Editor;
- Objeto Step PAR_ALT: criação no grafo de um *step* do tipo PAR_ALT, mesma funcionalidade apresentada pela opção “Step–Cria–ParAlt”. O objeto deve ser “arrastado” até área de trabalho do Editor;
- Objeto Step PFEConst: criação no grafo de um *step* do tipo PAR_FOR_EACH_CONST, mesma funcionalidade apresentada pela opção “Step–Cria–ParForEach Constante”. O objeto deve ser “arrastado” até área de trabalho do Editor;
- Objeto PFECons: criação no grafo de um *step* do tipo PAR_FOR_EACH_CONS, mesma funcionalidade apresentada pela opção “Step–Cria–ParForEach Consulta”. O objeto deve ser “arrastado” até área de trabalho do Editor.
- Objeto Arco Normal: define arco Normal, mesma funcionalidade da opção “Arco–Cria–Normal”. Após acionar o objeto, o usuário deve “clicar” na área de trabalho sobre os dois *steps* a serem conectados;
- Objeto Arco Falha: semelhante ao anterior para a opção “Arco–Cria–falha”;
- Objeto Transação: indica a criação de uma transação. Após este objeto ser acionado os *steps* que participam da transação devem ser indicados na área de trabalho;
- Objeto Apaga: permite a retirada de *steps* e arcos do grafo.

5.2.8.1 Escolha da técnica de Edição

A primeira idéia a respeito de como implementar a edição de grafos considerou a utilização de alguma ferramenta que proporcionasse *layout* automático de grafos. Algoritmos de *layout* automático (ROBINS, 1987; ROWE et al., 1987; SUGIYAMA; TAGAWA; TODA, 1981) posicionam nodos e arcos de um grafo de acordo com um ou mais padrões de estética e tornam-se essenciais quando grandes grafos estão envolvidos. Entretanto, os algoritmos existentes não permitem que sejam considerados aspectos específicos da aplicação. Além disto, o *layout* automático não é estável, uma pequena mudança na estrutura do grafo pode modificá-lo drasticamente.

A ferramenta Edge (PAULISCH; TICHY, 1990) oferece vários algoritmos de *layout* diferentes e provê um gerente de restrições para suportar requisitos de *layout* especiais assim como estabilidade do *layout*. Outra característica importante observada em alguns grafos é a sua complexidade, ou seja, o grande número de nodos e arcos envolvidos. Isto faz com que eles se tornem inoperáveis sem o auxílio de alguma ferramenta apropriada. Edge permite que nodos sejam agrupados em subgrafos e visualizados em diferentes níveis de abstração (NEWBERY, 1989).

Prevendo que aplicações de editores de grafo precisem customizar tanto a aparência visual dos grafos (ex.: forma dos nodos e arcos, cor, etc.) assim como a interface da própria aplicação, Edge utiliza uma descrição em GRL [1] (MANKE, 1990; PAULISCH, 1991) para especificar o formato e *layout* dos nodos e arcos dos grafos. Através de uma GRL também é possível armazenar informações parciais sobre o grafo e a sua seção de edição, permitindo ao usuário recomeçar uma edição com a mesma configuração que finalizou.

A filosofia de programação do UNIX (MORGAN; MCGILTON, 1987), de que sistemas complexos deveriam ser construídos como uma combinação de vários programas mais simples, foi aplicada no desenvolvimento desta ferramenta. Existem vários modos através dos quais sistemas podem ser construídos incluindo composição funcional (pipes) e geradores de programas (lex, yacc) (JOHNSON, 1975a,b; MANSON; BROWN, 1990). Edge foi projetado dentro desta filosofia de integração e flexibilidade. Por exemplo, é possível associar a saída de um analisador de código ao Edge para mostrar o grafo de chamadas de um programa ou, ao contrário, projetar graficamente a relação de dependência entre módulos de um sistema e colocar o resultado em um *Makefile*. O próprio Edge contém um número de componentes potencialmente reusáveis (algoritmos de
layout, algoritmo de concentração, etc.) que podem ser utilizados fora do Edge com mínima reprogramação.

Várias funções providas pela ferramenta Edge poderiam ser integradas de maneira eficiente ao trabalho aqui apresentado. Entretanto, como algum nível de reprogramação é necessário e cada módulo implementa uma função relativamente complexa, torna-se oneroso neste momento um estudo mais aprofundado da ferramenta, mesmo porque este não é o propósito principal do trabalho. Como um trabalho futuro propõe-se a análise desta e de outras ferramentas relacionadas ao mesmo tema, a saber: Xgrab, dag/dot (KOUTSOFIOS; NORTH, 1993), GraphEd e DaVincci.

Outro fator relevante e que não contribui para a adoção desta primeira idéia é a compilação de um grafo. Conforme visto, Edge armazena os grafos em uma GRL, a qual possui uma gramática relativamente complexa, pois permite a codificação de um variado número

[1] Graph Representation Language

de informações. Entretanto, como o objetivo principal deste trabalho é a geração de um código intermediário que represente o fluxo de execução da aplicação, muitas informações contidas na GRL são desperdiçadas. Também a análise de uma GRL para a geração de código envolve um esforço maior do que a de uma estrutura voltada especificamente para a resolução do problema.

Portanto, após esta análise, verifica-se que o desenvolvimento de um editor gráfico mais simples, responsável pelas tarefas de desenho do grafo, seu armazenamento em uma estrutura especificamente projetada para o objetivo maior do trabalho, a geração do código intermediário, representa a solução mais apropriada.

A seguir é proposta uma estrutura considerada apropriada para a representação dos grafos envolvidos na definição de contratos e que favorece o processo de compilação e geração de código. A Figura 5.24 mostra o nodo principal desta estrutura.

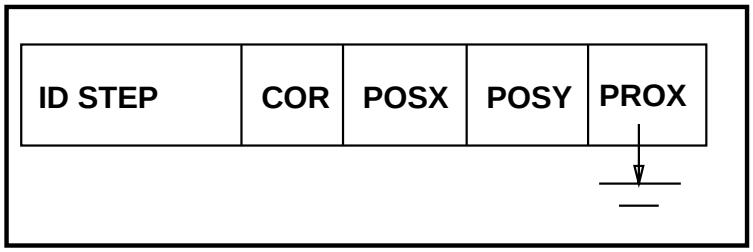

Figura 5.24: Nodo principal da estrutura de armazenamento de um grafo

A estrutura é uma lista, cujo nodo principal é composto por campos descrevendo informações sobre o *layout* de cada nodo. O primeiro campo componente do nodo principal é destinado a uma identificação única para o *step*. Os três campos seguintes descrevem a cor, posição em *x* e posição em *y* do nodo. Por fim, o campo Prox contém o endereço do próximo nodo criado seqüencialmente na lista.

Conforme o grafo é composto, a cada *step* identificado no editor, um novo nodo da estrutura é criado. É possível criar todos os *steps* sem estabelecer qualquer ligação entre eles, pois o campo Prox tem a função de interligar todos os nodos pela ordem de criação, não possuindo, necessariamente, nenhuma relação com a aplicação.

Quando uma transação é estabelecida as informações sobre sua posição também são armazenadas em uma lista. Cada nodo desta lista tem o formato mostrado na Figura 5.25.

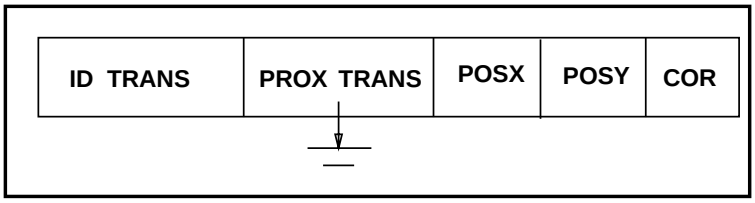

Figura 5.25: Nodo principal da estrutura de armazenamento de transações

O campo “IdTrans” é uma identificação única criada para a transação, “ProxTrans” aponta para a próxima transação criada e “posX”, “posY” e “COR” identificam a aparência do traçado que representa a transação.

5.3 Interface Execução

A interface do módulo de EXECUÇÃO, mostrada na Figura 5.26 é responsável pelo monitoramento do processo de execução de um contrato. Sua função é de comunicação com o Módulo Gerente de Contratos e com o usuário que submete o contrato para execução. A Figura 5.27 apresenta a legenda que permite verificar o estado de execução de cada *step* do *script*.

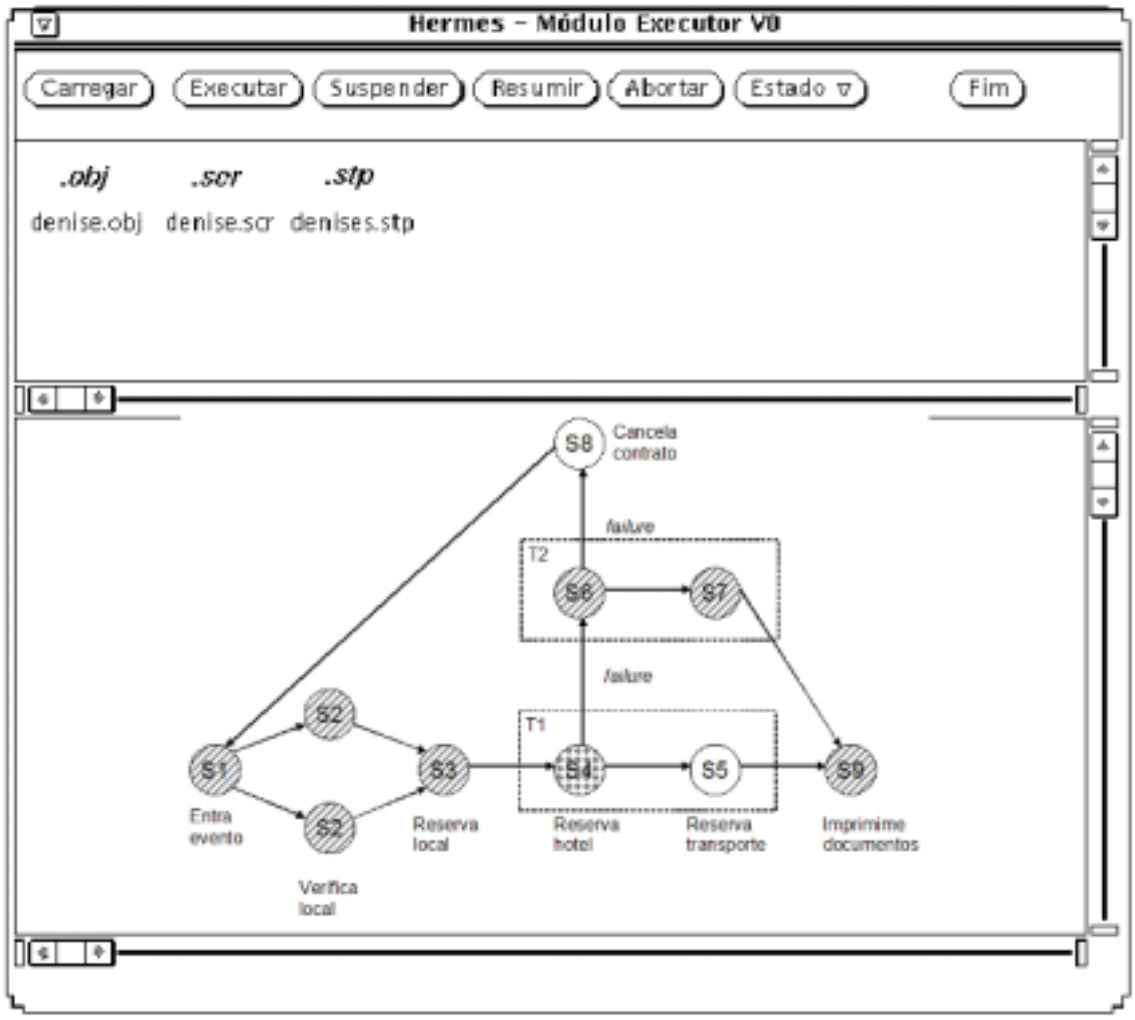

Figura 5.26: Histórico de Execução de um Contrato

Carregar

Quando a opção Carregar é acionada, aparece, na área de monitoramento de contratos, o grafo correspondente ao *script* do contrato carregado. Isto também pode ser obtido executando-se uma operação de “drag and drop” sobre um arquivo de extensão “scr” a partir da área de gerenciamento de arquivos até a área de monitoramento.

Executar

Após um contrato ser carregado, ele pode ser executado. Para isto utiliza-se a opção Executar. A partir do instante que um contrato entra em execução inicia-se um processo de comunicação constante entre os Sub–Módulos Interface Execução e executor. Os re-

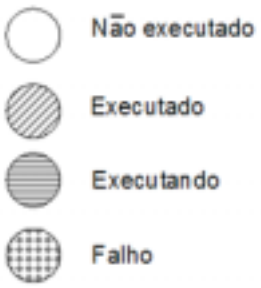

Figura 5.27: Legenda sobre o andamento da execução de um contrato

sultados desta comunicação podem ser verificados pela animação do grafo, na área de monitoramento, onde é possível saber qual o *step* está em execução a cada momento. A Figura 5.28 mostra o andamento da execução de um contrato na área de animação. A cada modificação no estado de execução do contrato a Interface Execução é notificada e atualiza a animação.

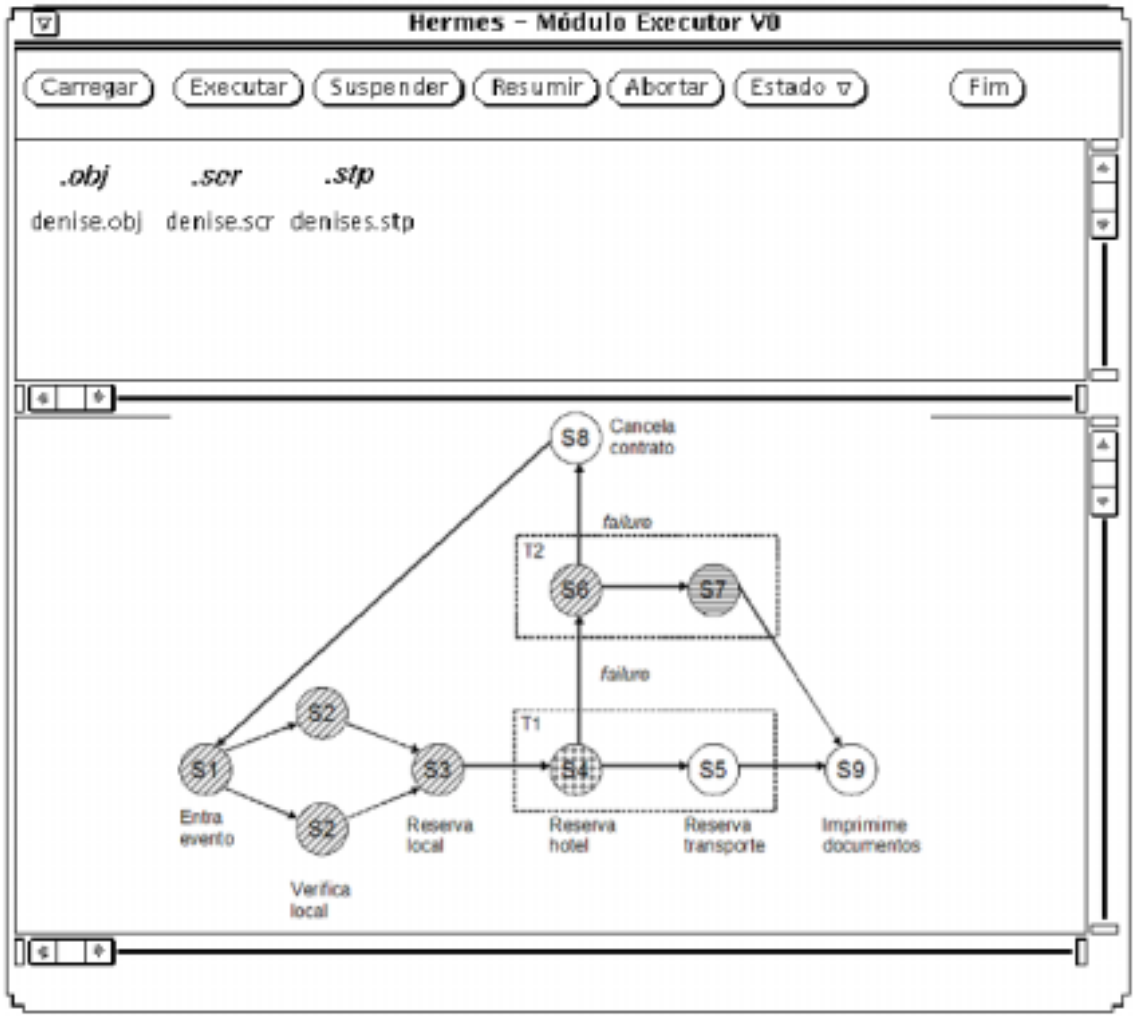

Figura 5.28: Andamento da Execução de um Contrato

Estado

Também é possível obter-se informações de estado sobre o contrato, existem dois tipos de informações oferecidas pela interface: sobre o contrato como um todo e sobre cada um dos *steps*, como mostra a Figura 5.29.

Figura 5.29: Menu Estado

Esta série de informações sobre os *steps* também pode ser obtida clicando-se duas vezes sobre a representação gráfica do referido *step* na área de monitoramento.

Os outros procedimentos referentes ao gerenciamento do processo de execução de um contrato e que podem ser aplicados sobre os mesmos são: Suspender um contrato, Resumir e Abortar. As ações tomadas pelo sistema quando cada uma destas opções é escolhida estão descritas no Capítulo 3.

Do mesmo modo que é provido na interface PRINCIPAL e DEFINIÇÃO, na interface EXECUÇÃO também é possível ter acesso ao “Sistema de Ajuda”. E a interface EXECUÇÃO é encerrada com a opção “Fim”, neste caso, apenas a interface é encerrada. A execução do contrato, se estiver em andamento, permanece inalterada.

6 CONCLUSÕES E EXTENSÕES FUTURAS

Este trabalho apresentou um protótipo para um modelo de transações Longas de Banco de Dados, O Modelo de Contratos, e o projeto de uma Interface para este protótipo. O Capitulo 1 introduziu os temas desenvolvidos durante o trabalho, os quais possibilitaram a definição do protótipo e da Interface. O segundo capítulo explorou o Modelo de Contratos, apresentando todos os seus conceitos através de um exemplo elucidativo. No Capítulo 3 foi descrito um protótipo para o Modelo de Contratos e, também, foram estabelecidos alguns requisitos para a elaboração de uma interface para este protótipo. Um estudo sobre interfaces foi apresentado no Capítulo 4, com objetivo de prover o projeto de uma interface adequada ao protótipo definido. No Capítulo 5 foi, então, definida uma Interface Gráfica para o protótipo de Modelo de Contratos.

Os requisitos para a elaboração da interface, identificados no Capítulo 3, foram satisfeitos pelo projeto apresentado no Capítulo 5. A interface é bastante flexível no que diz respeito ao tipo de usuário ao qual irá servir. O usuário mais especializado tem a oportunidade de realizar as tarefas providas pela interface de maneira rápida, principalmente através da Linguagem Visual, e o usuário menos experiente não é prejudicado, pois os menus, praticamente auto–explicativo, oferecem as mesmas operações que a linguagem visual.

No Capítulo 4 foram realizados alguns estudos sobre áreas da ciência, como psicologia, ergonomia, comunicação e artes gráficas. As idéias encontradas nestas áreas influenciaram o projeto da interface de maneira absoluta e o seu emprego demonstrou um bom resultado. A Definição de Contratos e *steps*, através da Interface Gráfica é bem mais rápida do que a codificação através de uma linguagem textual. Isto pode ser verificado, pois, em outro trabalho, foi desenvolvida uma Interface Textual para a edição do *script* e pelo fato do usuário precisar lembrar de todas as construções possíveis para a elaboração de um contrato.

Como identificado no Capítulo 3, o projeto de interfaces não deve ser uma atividade realizada em apenas uma etapa. O projeto iterativo, através de diversas análises feitas junto aos próprios usuários do sistema, constitui-se uma técnica de grande importância para chegar-se ao projeto ideal, ou, pelo menos, próximo dele. Por isto, um dos próximos passos para a continuação deste trabalho é a sua utilização por diversos tipos de usuários, com objetivo de identificar-se melhorias no projeto de interface proposto.

Também a etapa de execução de contratos não foi tratada neste trabalho e deve ser largamente explorada. Uma arquitetura definitiva para o Módulo Gerente de Contratos deve ser definida. E, assim que a execução efetiva de contratos puder ser realizada, o código intermediário também deverá ser revisto.

REFERÊNCIAS

AHLERT, H. **Um Modelo Não Procedural de Especificação e implementação voltado a sistemas transacionais em Banco de Dados**. 1994. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

BANCILHON, F.; KIM, W.; KORTH, H. F. A Model of CAD Transactions. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 1985. **Proceedings...** [S.l.: s.n.], 1985. p.25–33.

BAPTISTA, H. R. Comunicação Digital: sem fronteiras. **PC Magazine Brasil**, [S.l.], v.4, n.1, Jan. 1994.

BERNSTEIN, P. A.; HADZILACOS, V.; GOODMAN, N. **Concurrency Control and Recovery in Database Systems**. [S.l.]: Addison-Wesley Publishing Company, 1987. (Addison-Wesley Series in Computer Science).

EDWARDS, A. D. N. Visual Programming Languages: the next generation ? **SIGPLAN Notices**, [S.l.], v.23, n.4, p.43–50, Apr. 1988.

FOLEY, J.; KIM, W. C.; KOVACEVIC, S.; MURRAY, K. Defining interfaces at a high level of abstraction. **IEEE Software**, [S.l.], Jan. 1989.

FREITAS, C. D. S. **Um Estudo Sistemático sobre Linguagens Visuais**. Porto Alegre: CPGCC da UFRGS, 1992. 86p. (RP197).

GLINERT, E. P. (Ed.). **Visual Programming Environments**: paradigms and systems. [S.l.]: IEEE Computer Society Press Tutorial, 1990.

HAERDER, T.; REUTER, A. Principles of Transaction-Oriented Database Recovery. **Computing Surveys**, [S.l.], v.15, n.4, p.287–317, Dec. 1983.

HARTSON, H. R.; HIX, D. Human-Computer Interface Development: concepts and systems for its management. **Computing Surveys**, New York, v.21, n.1, p.5–92, Mar. 1989.

HELLER, D. **Xview Programming Manual**. [S.l.]: O'Reilly & Associates, Inc., 1990.

JOHNSON, S. C. **Lex - A Lexical Analyzer generator**. [S.l.]: AT&T Bell Laboratories, 1975.

JOHNSON, S. C. **Yacc - Yet another Compiler Compiler**. [S.l.]: AT&T Bell Laboratories, 1975.

KIM, H.; KORTH, H. F.; SILBERSCHATZ, A. P. A graphical query language. **Software Practice and Experience**, [S.l.], v.13, n.3, p.169–203, Mar. 1988.

KORTH, H. F.; KIM, W.; BANCILHON, F. On Long-Duration CAD Transactions. **Information Sciences**, [S.l.], v.46, p.73–107, 1988.

KOUTSOFIOS, E.; NORTH, S. C. **Drawing Graphs with dot**. Murray Hill, NJ: AT & T Bell Laboratories, 1993.

LEONG, M.; SAM, S.; NARASIMHALU, D. Towards a visual language for an object-oriented multi-media database system. **Visual Database Systems**, [S.l.], p.465–495, 1989.

LIANG, T.-P. User interface deseign for decision support systems: a self-adaptative approach. **Information & Management**, [S.l.], v.12, n.4, p.181–193, Apr. 1987.

MANKE, S. **Generation of Graph Manipulation Programs from Object-Oriented Specifications**. 1990. Dissertação (Mestrado em Ciência da Computação) — University of Karlsruhe, Department of Informatics.

MANSON, T.; BROWN, D. **Lex & Yacc**. [S.l.]: O'Reilly & Associates, Inc., 1990.

MELLO, R. S. **Uma Linguagem Visual de Consulta para o Ambiente STAR**. 1994. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

MORGAN, R.; MCGILTON, H. **Introducing UNIX System V**. [S.l.]: McGraw-Hill, 1987.

NEWBERY, F. J. Edge Concentration: a method for clustering directed graphs. **ACM SIGSOFT Software Engineering Notes**, [S.l.], v.14, n.7, Nov. 1989.

NIELSEN, J. The Usability Engineering Life Cycle. **Computer**, [S.l.], v.25, n.3, p.12–22, Mar. 1992.

NIELSEN, J. Iterative User–Interface Design. **Computer**, [S.l.], v.26, n.11, p.32–41, Nov. 1993.

NODINE, M.; SKARRA, A.; ZDONIK, S. Synchronization and Recovery in Cooperative Transactions. **Systems Design, Implementation and Use**, [S.l.], Mar. 1990.

PAULISCH, F. N. **The Design of an Extendible Editor**. 1991. Tese (Doutorado em Ciência da Computação) — University of Karlsruhe, Department of Informatics.

PAULISCH, F. N.; TICHY, W. F. EDGE: an extendible graph editor. **Sotware Practice and Experience**, [S.l.], v.20, n.S1, p.63–88, June 1990.

PEW, R. W.; AL., E. **Research needs for human factors**. Washington, D. C.: National Academy Press, 1983.

PIMENTA, M. S. **Um Modelo Canônico de Ferramenta para Desenvolvimento de Interface com o Usuàrio**. 1991. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

REUTER, A. Contracts: a means for extending control beyond transaction boundaries. In: INTERNATIONAL WORKSHOP ON HIGH PERFORMANCE TRANSACTION SYSTEMS, 3., 1989. **Proceedings...** [S.l.: s.n.], 1989.

REUTER, A.; SCHWENKREIS, F.; WAECHTER, H. **Zuverlaessige Abwicklung grosser Verteilter Anwendungen mit Contracts - Architektur eines Prototypen**. Germany: University of Kaiserslautern, 1992.

ROBINS, G. The ISI Grapher: a portable tool for displaying graphs pictorially. In: Symboliikka, 1987. **Proceedings...** [S.l.: s.n.], 1987.

ROWE, L. A.; DAVIS, M.; MESSINGER, E.; MEGER, C.; SPIRAKIS, C.; TUAN, A. A Browser for Directed Graphs. **Software Practice and Experience**, [S.l.], v.17, n.1, p.61–76, Jan. 1987.

SHNEIDERMAN, B. Direct manipulation: a step beyond programming languages. **IEEE Computer**, [S.l.], v.8, n.16, 1983.

SILVA, D. B.; IOCHPE, C. Interface para um Sistema Gerenciador de Transações Longas de Banco de Dados. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, SEMISH, 21., 1994, Caxambú. **Anais...** Belo Horizonte: UFMG, 1994. p.491–504.

SOUTO, M. A. M.; IOCHPE, C. Transações de Banco de Dados para Suporte ao Controle Automatizado da Produção: análise de requisitos e comparação de modelos. In: CONGRESSO NACIONAL DE AUTOMAÇÃO INDUSTRIAL, 5., 1999, São Paulo. **Anais...** São Paulo: Sucesu, 1992. v.1, p.313–320.

SOUTO, M. A. M.; IOCHPE, C. Suporte á Execução de Planos de Processo Industriais com Base em Transações de Banco de Dados. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 12., 1992, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1992.

SUGIYAMA, F.; TAGAWA, S.; TODA, M. Methods for Visual Understending of Hierarchical System Structures. **IEEE Trans. on Systems, Man and Cybernetics, SML**, [S.l.], v.11, n.2, p.109–125, Feb. 1981.

THIMBLEBY, H. **User Interface Design**. [S.l.]: ACM Press, 1990.

WACHTER, H. **ConTracts**: a means for improving reliability in distributed computing. In: IEEE COMPCON, 1991. **Proceedings...** [S.l.: s.n.], 1989.

WACHTER, H.; REUTER, A. The Contract Model. **IEEE Data Engineering Bulletin**, [S.l.], v.14, n.1, p.39-43, 1991.

WACHTER, H.; REUTER, A. The ConTract Model. In: ELMAGARMID, A. K. (Ed.). **Database Transaction Models For Advanced Applications**. San Mateo: Morgan Kaufmann, 1993. p.219–263.

WU, C. T.; HSIAO, D. K. Implementation of visual database interface using an object-oriented language. **Visual Database Systems**, [S.l.], p.105–125, 1989.