A *NAVIGATION PLAN DEFINITION LANGUAGE*
E
A *NAVIGATIONPLANTOOL*

RELATÓRIO TÉCNICO

Kelly Rosa Braghetto

Departamento de Ciência da Computação
Instituto de Matemática e Estatística - IME
Universidade de São Paulo - USP
Rua do Matão, 1010 - Cidade Universitária
05508-090 São Paulo - SP – Brasil
e-mail: kellyrb@ime.usp.br

JUNHO, 2006

1. Introdução

A NPDL (*Navigation Plan Definition Language*) é uma linguagem para auxiliar a criação e manutenção da representação de processos de negócio em um banco de dados relacional, implementada como uma extensão da linguagem SQL.

O interpretador da NPDL é parte da ferramenta *NavigationPlanTool*. A *NavigationPlanTool* foi desenvolvida para atender às características especificadas pela WfMC (*Workflow Management Coalition*) para um sistema de *workflow*. Essas características estão ilustradas na figura 4.1.

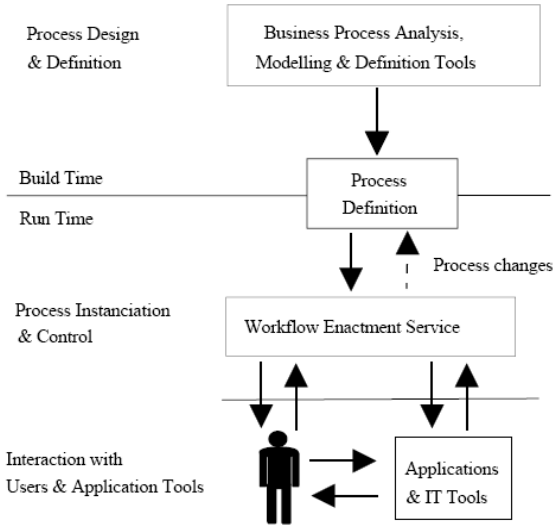

Figura 1.1: Características de um sistema de *workflow*

Em um sistema de *workflow* é possível identificar três interfaces: (1) projeto e definição dos processos; (2) instanciação e controle dos processos; e (3) interação com usuários e aplicativos. A NPDL atende à interface 1 de um sistema de *workflow*. Para as demais interfaces, a *NavigationPlanTool* também provê mecanismos para o controle da instanciação de processos, disponibilizando serviços para a manutenção de instâncias de processos e serviços para o monitoramento da execução do plano de navegação. Em outubro de 2005, a *NavigationPlanTool* foi apresentada na Sessão de Demos do XX Simpósio Brasileiro de Banco de Dados.

2. A Navigation Plan Definition Language (NPDL)

A *Navigation Plan Definition Language* (NPDL) foi definida com o objetivo de viabilizar o controle de processos de negócio para um modelo relacional de dados. Essa linguagem é uma extensão da linguagem SQL e se baseia no conceito de plano de navegação da arquitetura *RiverFish* e em operadores de álgebra de processos.

Assim como na álgebra de processos, processos em NPDL são definidos por expressões algébricas envolvendo ações e operadores. A NPDL contém os operadores mais comuns da álgebra de processos e também define operadores adicionais que modelam comportamentos freqüentes em processos de *workflow*.

A NPDL contém os seguintes operadores:

a) **Composição Seqüencial** (operador •): o termo de processo *A* . *B* significa que inicialmente somente o termo *A* está habilitado para execução. O termo *B* será habilitado para execução somente após o fim da execução do termo *A*;

b) **Composição Alternativa** (operador +): o termo de processo *A* + *B* significa que inicialmente ambos termos *A* e *B* estarão habilitados para execução;

c) **Composição Paralela** (operador ||): o termo de processo *A* || *B* significa que os termos *A* e *B* podem ser executados paralelamente, dividindo a linha de controle em duas;

d) **Composição Paralela Entrelaçada** (operador |*): o termo de processo $A|*B$ significa que os termos *A* e *B* podem ser executados em qualquer ordem (i.e., *A.B* + *B.A*), mas não paralelamente;

e) **Composição Multi-Convergente** (operador &): o termo de processo *A* & *B* significa que o termo *B* será habilitado para execução após o término da execução de cada linha de controle do termo *A*. Por exemplo, o processo $P = (X \,||\, Y) \,\&\, Z$ indica que *Z* será habilitado duas vezes: uma após o término da execução de *X* e outra após o término da execução de *Z*, mas no processo $P = (X + Y) \,\&\, Z$, *Z* será habilitado apenas uma vez, após a execução da atividade escolhida no + (*X* ou *Y*) ;

f) **Composição Discriminatória** (operador ^): o termo de processo *A* ^ *B* significa que o termo *B* será habilitado para execução após o primeiro término de uma

linha de controle do termo *A*. Por exemplo, o processo $P = (X \,||\, Y)\hat{}\, Z$ indica que *Z* será habilitado para execução somente uma vez; se a atividade *Y* terminar primeiro, então *Z* será habilitado depois da execução de *Y*. Neste caso, quando *X* terminar, *Z* não será habilitado novamente – esse comportamento diferencia do operador ^ do operador &;

g) **Repetição Ilimitada** (operador ?*): o termo de processo *A?** significa que o termo *A* pode ser executado um número ilimitado de vezes;

h) **Repetição Numericamente Limitada** (operador ?*n*, sendo *n* um número inteiro positivo e não nulo): o termo de processo *A?5* significa que o termo *A* deve ser executado 5 vezes;

i) **Repetição Limitada por Função** (operador ?*f*, sendo *f* uma função que retorna um número inteiro positivo e não nulo): o termo de processo $A?f_1$ significa que o termo *A* deve ser executado o número de vezes retornado pela função f_1 em tempo de execução;

j) **Execução Condicional** (operador %*r*, sendo *r* é uma regra, ou seja, uma função *booleana*): o termo de processo $\%r_1\ A$ significa que o termo *A* será habilitado para execução se o valor de retorno da regra r_1 (avaliada em tempo de execução) for **verdadeiro**;

k) **Execução Condicional Negativa** (operador %!*r*, sendo *r* é uma regra, ou seja, uma função *booleana*): o termo de processo $\%!r_1\ A$ significa que o termo *A* será habilitado para execução se o valor de retorno da regra r_1 (avaliada em tempo de execução) for **falso**.

l)

Um processo em NDPL é definido por um termo fechado. Este termo fechado é construído a partir do conjunto ***A*** de ações atômicas, de operadores NPDL e do conjunto ***P***, sendo ***P*** o conjunto de todos os processos definidos. Por exemplo, os comandos NPDL seguintes especificam um processo chamado “ProcessoCalculo” (identificado por ***P2***) que realiza a soma ou a multiplicação convencional de dois números um número indeterminado de vezes:

```
CREATE ACTION A1 ‘LerPrimeiroValor’;
CREATE ACTION A2 ‘LerSegundoValor’;
```

```
CREATE ACTION A3 ‘CalcularAdicao’;
CREATE ACTION A4 ‘CalcularMultiplicacao’;
CREATE ACTION A5 ‘MostrarResultado’;
CREATE PROCESS P1 ‘AuxProcessoCalculo’;
CREATE PROCESS P2 ‘ProcessoCalculo’;
SET P1 = (A1 || A2).( A3 + A4 ).A5;
SET P2 = P1. P2 + P1;
```

Usar-se-á como exemplo de especificação de processo mais complexa um **sistema de tratamento de reclamações**. A figura 2.1 mostra o modelo do sistema em Rede de Petri.

O primeiro passo no sistema é o registro da reclamação, e como conseqüência um questionário é enviado ao reclamante. A reclamação é avaliada ao mesmo tempo em que o questionário é enviado. Se o reclamante não retornar o questionário preenchido dentro de 2 semanas o prazo de retorno é considerado expirado e o resultado do questionário será descartado. Após a avaliação, uma reclamação pode ser processada ou não. A reclamação somente é processada após o processamento do questionário ou após o prazo de retorno ter expirado. O processamento da reclamação é repetido até que o resultado seja satisfatório. Ao término destas atividades, a reclamação é arquivada.

Esse caso ilustra, além de um ciclo, o uso de vários padrões de controle de fluxo: **divisão paralela**, após a atividade ***A*** (registro), que habilita a execução paralela de ***B*** (envio do questionário) e ***F*** (avaliação); **escolha postergada**, entre as atividades ***C*** (processamento do questionário) e ***D*** (expiração do prazo); **escolha exclusiva** entre as atividades ***G*** (processamento requerido) e ***H*** (omissão) e entre ***L*** (OK – processamento correto) e ***K*** (NOK – processamento incorreto); **marco** para a atividade ***I*** (processamento da reclamação); e, finalmente, **sincronização** para a execução da atividade ***E*** (arquivamento).

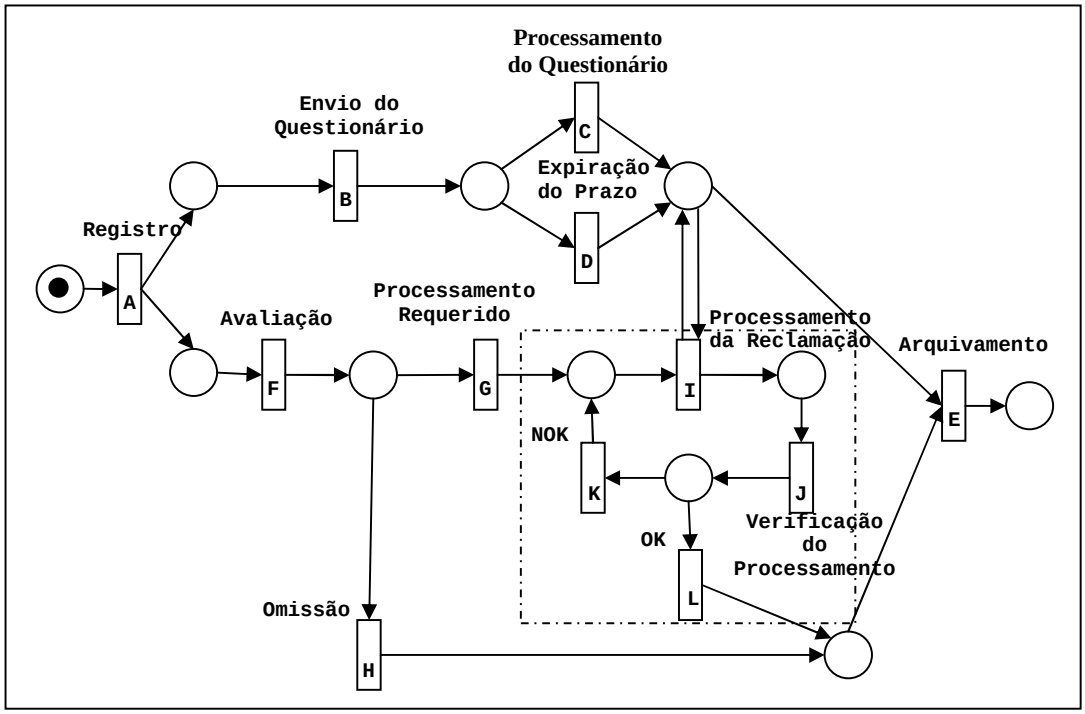

Figura 2.1: Sistema de processamento de reclamações (Fonte: [6])

O processo NPDL que representa o sistema descrito é definido por ***P*** tal que:

```
// Considerando que as ações A, B, C, D, E,
// F, G, H, I, J, K e L
// já foram criadas em NPDL
CREATE PROCESS P1 ;
CREATE PROCESS P;
SET P1 = I.J.(%!r1 K.P1 + %r1 L) ;
SET P = A.((B.(C + D)|| F . %r2 G).P1 +
            (B.(C + D)|| F .%!r2 H)).E ;
```

sendo $\mathbf{r_1}$ uma regra que verifica alguns atributos manipulados pela atividade ***J*** (verificação do processamento) e $\mathbf{r_2}$ uma regra que verifica alguns atributos manipulados pela atividade ***F*** (avaliação).

O Anexo 1 deste documento apresenta a sintaxe completa da NPDL.

3. A NavigationPlanTool

A *NavigationPlanTool* foi implementada na forma de uma biblioteca de funções. Esta biblioteca provê mecanismos para a manutenção de ações e processos em bancos de dados relacionais e para o controle de instanciação e execução desses processos. A biblioteca disponibiliza operações como criação/remoção de instâncias e serviços para o monitoramento da execução do plano de navegação. Esses serviços são também responsáveis pela manutenção de *logs* de execução de planos de navegação no banco de dados e pela recuperação de execuções que tenham sido interrompidas antes de serem finalizadas.

A *NavigationPlanTool* foi desenvolvida em Java (*Java 2 Platform Standard Edition* - J2SE 5.0) [28], por esta ser uma linguagem portável e contar com a JDBC API - *Java DataBase Conectivity Application Programming Interface*. A JDBC habilita programas Java a executarem comandos em SQL e interagirem com qualquer banco de dados compatível com a definição padrão da linguagem SQL, tornando assim a NPDL uma extensão independente de sistema gerenciador de banco de dados.

Por ser desenvolvida na forma de biblioteca de funções, a *NavigationPlanTool* pode ser integrada de forma fácil a outras aplicações Java.

A *NavigationPlanTool* é composta por três serviços: o **Interpretador NPDL**, o **Serviço de Instanciação de Processos** e o **Serviço Monitor de Execução de Instâncias de Processos**. Nas seções seguintes, cada um desses serviços será detalhado.

3.1 O Interpretador da Linguagem NPDL

A função do **Interpretador NPDL** é receber um comando de entrada e efetuar as análises léxica, sintática e semântica. Caso o comando seja identificado como um comando NPDL válido, ele é executado sobre as relações criadas pelo próprio interpretador em um banco de dados relacional, para o armazenamento dos dados dos processos, ações e instâncias.

O interpretador é uma implementação da interface *java.sql.Connection* de Java, que recebeu o nome **NPDLConnection**. A interface *java.sql.Connection* representa uma conexão com um banco de dados específico. A partir de um objeto de uma classe de conexão é possível executar consultas SQL sobre o banco de dados.

Criar uma conexão NPDL com um banco de dados significa preparar este banco para ser usado no controle de processos de negócio via *NavigationPlanTool*. Quando uma aplicação instancia um objeto da classe **NPDLConnection**, a rotina de iniciação do objeto verifica a existência das relações utilizadas pela *NavigationPlanTool* no banco de dados alvo da conexão. Caso essas relações ainda não existam no banco, elas são criadas pela rotina de iniciação.

Uma conexão do tipo **NPDLConnection** permite que tanto comandos NPDL quanto comandos SQL sejam submetidos ao banco de dados. Ao receber um comando, uma conexão NPDL o verifica e, caso o identifique como um comando NPDL válido, executa as rotinas que traduzem o comando NPDL para operações em SQL (padrão da linguagem). Esses comandos SQL são então executados sobre as relações associadas à *NavigationPlanTool*. Quando o comando submetido para a conexão NPDL não é reconhecido como um comando NPDL válido, ele é repassado diretamente ao banco de dados.

A estrutura de dados relacional mantida pela NavigationPlanTool em banco de dados é mostrada no Anexo 2 deste documento.

3.2 O Serviço de Instanciação de Processos

O **Serviço de Instanciação de Processos** disponibiliza funções para a criação de instâncias de processos. Uma instância representa uma requisição de um determinado processo. Toda instância possui uma referência para o processo ao qual está associada, e pode também armazenar informações sobre o usuário que a requisitou e a sua data de criação.

Quando a criação de uma instância é solicitada à *NavigationPlanTool* por uma aplicação, o serviço de instanciação retorna à aplicação um identificador da instância. Por

meio desse identificador, a aplicação pode obter os demais dados da instância e interagir com o serviço que controla a execução do plano de navegação da instância.

3.3 O Serviço Monitor de Execução de Instâncias de Processos

O serviço **Monitor de Execução de Instâncias de Processos** é responsável por ligar uma instância de processo aos seus dados de execução (plano de navegação). Neste serviço, estão agrupadas as funções que iniciam e monitoram a execução do plano de navegação de uma instância de processo.

Para controlar a execução do plano de navegação, são utilizados elementos do *log* de execução de instâncias e estruturas de dados em memória.

Quando a execução de uma instância é iniciada, o plano de navegação associado a ela é recuperado do banco de dados**.** A partir da expressão algébrica prefixa que representa o plano de navegação, é construída uma **árvore de expressão** da instância. As árvores de expressão são utilizadas pelo serviço monitor para determinar a ordem de execução dos passos de um plano de navegação. Neste trabalho, denominou-se de **árvore de navegação** a árvore de expressão de uma instância de processo.

Após a criação da árvore de navegação, o monitor realiza a recuperação do estado atual da execução da instância, por meio de consultas aos dados mantidos no *log* de execução de instâncias**.** Esta operação só é realizada para as instâncias que tiveram o seu processo de execução iniciado, mas ainda não finalizado. Cada passo iniciado, finalizado ou cancelado do plano de navegação de uma instância resulta na inserção ou atualização de um registro no *log*. Instâncias que não iniciaram ao menos um passo de seu plano de navegação não possuem informações gravadas no *log*.

A execução de instâncias distintas de um mesmo processo pode resultar em *logs* diferentes. Isso ocorre porque o plano de navegação mapeia todas as possibilidades de execução de um processo, mas são os dados da instância que determinarão o comportamento final de cada nova execução desse processo.

4. Como utilizar a *NavigationPlanTool*

A *NavigationPlanTool* é uma biblioteca de funções Java. Para utilizá-la, basta incluir o arquivo "jar" da biblioteca no projeto. A ferramenta possui um conjunto extenso de classes, mas as classes de interesse para os usuários são:

- Pacote ***br.usp.ime.nptool.database***
  - *NPDLConnection*
  - *NPDataAccessor*
- Pacote ***br.usp.ime.nptool.entities***
  - *Action*
  - *Rule*
  - *Function*
  - *Process*
  - *ProcessInstance*
- Pacote ***br.usp.ime.nptool.services***
  - *NPExecutionMonitor*
  - *NPExecutionStep*

A classe *NPDLConnection* é uma implementação da interface *java.sql.Connection*; por meio de um objeto desta classe, é possível submeter comandos NPDL e SQL para o banco de dados. O construtor da *NPDLConnection* recebe como parâmetro um objeto da classe *Connection*, ou seja, a conexão com o banco que mantém os dados de processos. Exemplo de uso da *NPDLConnection*:

```
...
Connection commonConn;
try{
      // Abre conexão com o banco que mantém os dados de processos
      commonConn = DriverManager.getConnection(
        "jdbc:postgresql://" + host + "/db_process", user, password);
} catch (SQLException e) {
      e.printStackTrace();
}
NPDLConnection npdlConn = new NPDLConnection(commonConn);
```

```
// Cria uma ação e a associa a ela a uma instrução de
// execução. A instrução de execução pode ser um .exe,
// um nome de método, etc – o importante é que a aplicação
// saiba como executar a ação.
npdlConn.createStatement().execute("CREATE ACTION actionTest; " +
               "ADD actionTest EXECUTION_CALL 'actionTest.exe'");
// Cria um processo recursivo que executa a ação actionTest de
// forma seqüencial, um número irrestrito de vezes.
npdlConn.createStatement().execute("CREATE PROCESS processTest; ");
npdlConn.createStatement().execute(
                      "SET processTest = actionTest.processTest;");

// Exemplo de execução de um comando NPDL que retorna um conjunto
// de registros. Consulta todas as ações cadastradas no banco de
// dados e as armazena em um vetor

ResultSet res = npdlConn.createStatement().executeQuery("SELECT
ACTIONS");
ArrayList dbActions = new ArrayList();

while (res.next()) {
        dbActions.add(res.getString("step_description"));
}
```

A classe *NPDataAccessor* provê métodos para a manipulação das estruturas de dados mantidas pela *NavigationPlanTool*. As instâncias de processos definidos em NPDL devem ser criadas por meio dela.

O construtor da *NPDataAccessor* recebe como parâmetro um objeto de *NPDLConnection*. Exemplo de uso da *NPDataAccessor*:

```
...
NPDataAccessor npAcessor = new NPDataAccessor(npdlConn);
// Cria uma instância do processo “processTest”, associando
// a ela o login do usuário que a requisitou e data de requisição
ProcessInstance instance = npAccessor.newProcessInstance(“processTest”,
                                                "user1", new Date());
// Também é possível por meio de um objeto NPDataAccessor obter a
// listagem de ações, processos, regras ou funções cadastradas no BD
```

```
Action[] actions = npAccessor.getActions();
...
```

A classe *NPExecutionMonitor* implementa o serviço de controle de execução de instâncias de processos. O construtor da classe recebe como parâmetro um objeto *NPDataAccessor* e uma instância de processo. O monitor recupera do banco de dados o log de execução da instância (caso ela tenha tido sua execução iniciada) e mantém uma estrutura de dados em memória, capaz de determinar os próximos passos válidos para a execução na instância. Os passos válidos são sempre retornados como um objeto de *NPExecutionStep*. A partir de *NPExecutionStep*, é possível obter a listagem de ações, regras ou funções habilitadas para a execução. Também é a partir de um *NPExecutionStep* que a instância indica que a execução de uma ação foi iniciada ou encerrada.

```
...
// Cria monitor de execução para a instância de processo criada
// anteriormente
NPExecutionMonitor npExecMonitor = new NPExecutionMonitor(npAccessor,
                                                          instance);
// Obtém os próximos passos válidos para a execução na instância
NPExecutionStep executionStep = npExecMonitor.getNextActions();
Action[][] actualActions = executionStep.getActions();
do {    actualActions = executionStep.getActions();
        String actionExecution = actualActions[0][0].getDescription();
        String actionExecution = actualActions[0][0].getExecutionCall();
        // Marca no log o início da execução da ação
        Long actionId = executionStep.setStartedAction(actionName);
        // Indica a execução da ação – aqui, hipoteticamente representado
        // por um método “execute”
        execute(actionExecution);
        // Marca no log o término da execução da ação
        executionStep.setFinishedAction(actionId);

       // Obtém os próximos passos válidos para execução
        executionStep = npExecMonitor.getNextActions();
} while (!npExecMonitor.isExecutionFinished());
...
```

Toda instância possui um número que a identifica univocamente no banco de dados. Como é possível que uma instância de processo tenha sua execução iniciada por uma aplicação e encerrada antes do seu término, é preciso que a aplicação mantenha os identificadores das instâncias que criou, para que seja capaz de recuperá-las do BD via *NavigationPlanTool* e continuar suas execuções a partir do ponto em que foram interrompidas.

```
...
// Obtém o número identificador da instância
Long idInstance = instance.getInstanceId();
...
...
...
// Recupera uma instância do BD a partir de seu número identificador
ProcessInstance recoveredInstance =
                                    npAccessor.getProcessInstance(idInstance);
...
```

É importante ressaltar que as classes citadas possuem muitos outros métodos além dos apresentados nos exemplos de uso; não é o escopo deste documento mostrar e descrever todos eles. Um manual da ferramenta está sendo desenvolvido e atenderá este requisito.

Anexo 1

A Sintaxe da *Navigation Plan Definition Language*

Considere que a gramática da linguagem NPDL é definida por *(N, T, P, S)*, sendo *N* o **conjunto de símbolos de variáveis** (ou símbolos não-terminais); *T* o **conjunto dos símbolos terminais**; *P* o **conjunto das regras de produção da gramática** e *S* o **símbolo inicial** de *N*.

A NPDL obedece às seguintes propriedades:

1) $S \in N$

2) *N* = { action-description, add-action-execution-call, add-function-execution-call, add-process-service-description, add-rule-execution-call, choice-operator-sign, command, condition, deadlock-symbol, delete-action, delete-function, delete-process, delete-rule, discriminator-operator-sign, execution-call, factor, function-description, function-limited-repetition, interleaved-parallel-operator-sign, left-parentheses, limited-repetition, list-actions, list-functions, list-navigation-plan, list-processes, list-rules, multi-merge-sign, new-action, new-function, new-process, new-rule, parallel-operator-sign, process, process-description, process-expression, right-parentheses, S, service-description, sequential-operator-sign, term, rule-description, unlimited-repetition}

3) *T* = {ACTION, ACTIONS, ADD, CALL, CREATE, DESCRIPTION, DROP, EXECUTION, FROM, FUNCTION, FUNCTIONS, NAVIGATION, PLAN, PROCESS, PROCESSES, RULE, RULES, SELECT, SERVICE, SET, \n**,** (**,**) **,** = **,** . , +**,** ^, || , |*, #, *, ?, &, %, %!}

4) O conjunto *P* é gerado a partir das regras de produção da tabela A.1, descritas no formato BNF (*Backus-Naur Form*).

Tabela A.1: Regras de produção da NPDL

```
<S> ::= (<command>? \n )+
<command> ::= <new-action>
    | <new-function>
    | <new-process>
    | <new-rule>
    | <add-action-execution-call>
    | <add-function-execution-call>
    | <add-process-service-description>
    | <add-rule-execution-call>
    | <delete-action>
    | <delete-function>
    | <delete-process>
    | <delete-rule>
    | <process-expression>
    | <list-actions>
    | <list-functions>
    | <list-rules>
    | <list-processes>
    | <list-navigation-plan>
<new-process> ::= CREATE PROCESS <process-description>
                            [<service-description>]
<new-action>  ::= CREATE ACTION <action-description>
                            [<execution-call>]
<new-function> ::= CREATE FUNCTION <function-description>
                            [<execution-call>]
<new-rule> ::= CREATE RULE <rule-description>
                            [<execution-call>]
<add-process-service-description> ::= ADD <process-description>
                        SERVICE DESCRIPTION <service-description>
<add-action-execution-call> ::= ADD <action-description>
                        EXECUTION CALL <execution-call>
<add-function-execution-call> ::= ADD <function-description>
                         EXECUTION CALL <execution-call>
<add-rule-execution-call> ::= ADD <rule-description>
                         EXECUTION CALL <execution-call>
<process-expression> ::= SET <process-description> = <process>
<delete-process> ::= DROP PROCESS <process-description>
<delete-action> ::= DROP ACTION <action-description>
<delete-function> ::= DROP FUNCTION <function-description>
<delete-rule> ::= DROP RULE <rule-description>
<list-processes> ::= SELECT PROCESSES
<list-actions> ::= SELECT ACTIONS
```

```
<list-functions> ::= SELECT FUNCTIONS
<list-rules> ::= SELECT RULES
<list-navigation-plan> ::= SELECT NAVIGATION PLAN FROM PROCESS
  <process-description>
<left-parentheses> ::= (
<right-parentheses> ::= )
<choice-operator-sign> ::= +
<sequential-operator-sign> ::= .
<parallel-operator-sign> ::= ||
<interleaved-parallel-operator-sign> ::= |*
<multi-merge-operator-sign> ::= &
<discriminator-operator-sign> ::= ^
<deadlock-symbol> ::= #
<unlimited-repetition> := ?*
<limited-repetition> := ? <unsigned_integer>[1]
<function-limited-repetition> := ? <function>
<condition> := % <rule>
<not-condition> := %! <rule>
<process> ::= <term>
          | <process> <choice-operator-sign> <term>
<term> ::= <subterm>
          | <term> <interleaved-parallel-operator-sign> <subterm>
<subterm> ::= <prefactor>
          | <subterm> <parallel-operator-sign> <prefactor>
<prefactor> ::= <factor>
          | <prefactor> <sequential-operator-sign> <factor>
<factor> ::= <subfactor>
          | <factor> <multi-merge-operator-sign> <subfactor>
          | <factor> <discriminator-operator-sign> <subfactor>
<subfactor> ::= <action>
          | <process-description>
          | <process>
          | <deadlock-symbol>
          | <left-parentheses> <process> <right-parentheses>
          | <subfactor><unlimited-repetition>
          | <subfactor><limited-repetition>
          | <subfactor><function-limited-repetition>
          | <condition><subfactor>
          | <not-condition><subfactor>
<action> ::= <action-description>
```

[1] <unsigned_integer> está definido na especificação BNF do ANSI SQL92.

```
<rule> ::= <rule-description>
<function> ::= <function-description>
<action-description> ::= <identifier>[2]
<rule-description> ::= <identifier>
<process-description> ::= <identifier>
<service-description> ::= <character_string_literal>[3]
<execution-call> ::= <character_string_literal>
```

[2] <identifier> está definido na especificação BNF do ANSI SQL92.

[3] <character_string_literal> está definido na especificação BNF do ANSI SQL92.

Anexo 2

Estruturas Relacionais de Dados

As estruturas de dados criadas em banco de dados relacional pela *NavigationPlanTool* baseiam-se nas estruturas sugeridas pela arquitetura *RiverFish*. Para representar processos de negócio e controlar sua instanciação e execução por meio da abordagem proposta por esta dissertação, são necessárias as estruturas descritas no diagrama da figura B.1. A notação utilizada no diagrama é a especificada em [38] para Modelos Entidade-Relacionamento Estendidos.

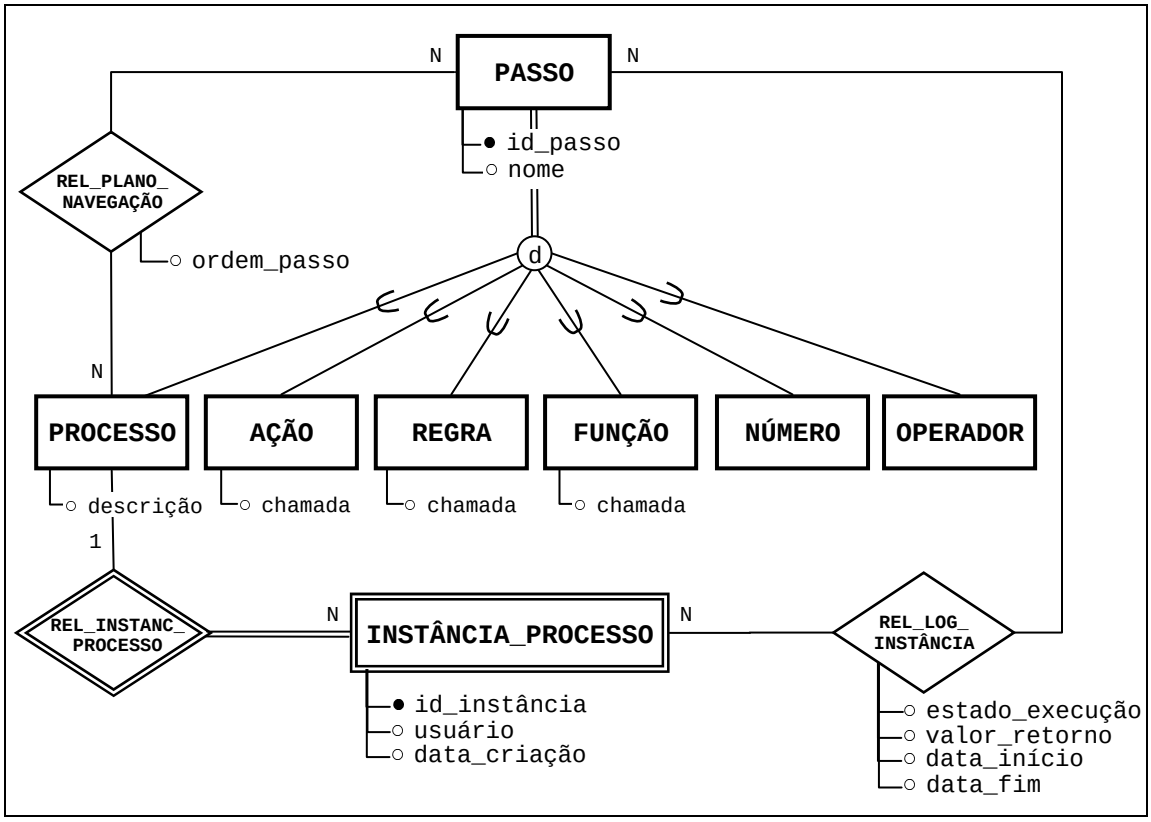

Figura B.1: Diagrama Entidade-Relacionamento

Em NPDL, o plano de navegação de um processo é definido por uma expressão algébrica que pode envolver ações, operadores, regras, funções, números e outros processos. O relacionamento **REL_PLANO_NAVEGAÇÃO** entre as entidades **PROCESSO** e **PASSO** representa esta definição. A especialização da entidade

PASSO é total e disjunta, ou seja, todo elemento dessa entidade obrigatoriamente possui um dos seguintes tipos: processo, ação, regra, função, número ou operador.

Antes de ser armazenada em banco de dados por meio do relacionamento **REL_PLANO_NAVEGAÇÃO**, a expressão algébrica de processo é convertida em uma expressão equivalente na notação prefixa. A expressão prefixa é mais apropriada para o armazenamento em banco de dados e para manipulação nos algoritmos que avaliam a validade da expressão. O atributo **ordem_passo** do relacionamento **REL_PLANO_NAVEGAÇÃO** é utilizado para armazenar a posição do passo dentro da expressão algébrica. Esta informação é necessária para a posterior recuperação da expressão do processo.

A entidade **INSTÂNCIA_PROCESSO** e o relacionamento **REL_LOG_INSTANCIA** estão associados à instanciação e ao controle de execução de processos e por isso são utilizadas pelos outros serviços da *NavigationPlanTool*.

Um elemento da entidade **INSTÂNCIA_PROCESSO** está sempre associado a um elemento da entidade **PROCESSO**, por meio do relacionamento **REL_INSTANC_PROCESSO**. Uma instância representa uma requisição a um processo, por isso possui como atributos o usuário que efetuou a requisição e a sua data. O relacionamento **REL_LOG_INSTÂNCIA** representa os dados relativos ao estado da execução dos passos que compõem o plano de navegação de uma instância de processo. Cada elemento de **REL_LOG_INSTÂNCIA** representa a execução de um passo em uma determinada instância de processo. O atributo **estado_execução** do relacionamento **REL_LOG_INSTÂNCIA** pode assumir um dos valores descritos na tabela B.1.

Os tipos de passos que são realmente "executáveis" são **AÇÃO**, **REGRA** E **FUNÇÃO**. Por este motivo, somente estas três entidades da especialização de **PASSO** possuem o atributo **chamada**, que representa uma instrução para a execução do passo.

Tabela B.1: Possíveis estados para passos

Valor	**Representando um passo...**
I	Iniciado, mas ainda não finalizado
F	Finalizado
C	Cancelado

Quando uma conexão para a execução de comandos NPDL é estabelecida pela primeira vez com um determinado banco de dados, o serviço interpretador da NPDL

cria no banco as relações utilizas pela *NavigationPlanTool* para a manutenção de processos, instâncias e seus dados de execução. A partir do diagrama da figura B.1, são obtidas cinco relações (tabelas); a estrutura destas relações e exemplos dos dados armazenados são mostrados nas tabelas de B.3 a B.9.

Os comandos SQL executados pelo serviço interpretador NPDL sobre o banco de dados para a criação das relações são os listados na tabela B.2.

Tabela B.2: Comandos SQL para a criação das relações

```
/* Criação da tabela de Processos */
CREATE TABLE np_process(
      process_id DECIMAL(15) NOT NULL,
      process_description VARCHAR NOT NULL,
      service_description VARCHAR NULL,
      CONSTRAINT pk_np_process PRIMARY KEY (process_id));
CREATE UNIQUE INDEX np_process_unique_index
      ON np_process(process_description);
GRANT ALL PRIVILEGES ON np_process TO PUBLIC;

/* Criação da tabela de Passos */
CREATE TABLE np_step (
      step_id DECIMAL(15) NOT NULL,
      step_description VARCHAR NOT NULL,
      step_type CHAR NOT NULL,
      execution_call VARCHAR NULL,
      CONSTRAINT pk_np_step PRIMARY KEY (step_id));
CREATE UNIQUE INDEX np_step_unique_index
      ON np_step (step_description);
GRANT ALL PRIVILEGES ON np_step TO PUBLIC;

/* Criação da tabela de Plano de Navegação */
CREATE TABLE np_navigation_plan (
      navigation_plan_id DECIMAL(15) NOT NULL,
      process_id DECIMAL(15) NOT NULL,
      component_type CHAR(1) NOT NULL,
      component_id DECIMAL(15) NULL,
      CONSTRAINT pk_np_navigation_plan PRIMARY KEY
         (navigation_plan_id),
      CONSTRAINT fk_np_navigation_plan FOREIGN KEY
         (process_id) REFERENCES np_process (process_id));
CREATE UNIQUE INDEX np_navigation_plan_unique_index ON
          np_navigation_plan (navigation_plan_id);
GRANT ALL PRIVILEGES ON np_navigation_plan TO PUBLIC;
```

```
/* Criação da tabela de Instâncias */
CREATE TABLE np_requested_process (
      instance_id DECIMAL(15) NOT NULL,
      process_id DECIMAL(15) NOT NULL,
      user_name VARCHAR NOT NULL,
      request_date DATE NULL,
      request_time TIME NULL,
      CONSTRAINT     pk_np_requested_process     PRIMARY     KEY
(instance_id),
      CONSTRAINT fk_np_requested_process FOREIGN KEY
            (process_id) REFERENCES np_process (process_id));
CREATE UNIQUE INDEX np_requested_process_unique_index ON
       np_requested_process (instance_id);
GRANT ALL PRIVILEGES ON np_requested_process TO PUBLIC;

/* Criação da tabela de Log de execução de instâncias */
CREATE TABLE np_instance_log (
      instance_log_id DECIMAL(15) NOT NULL,
      instance_id DECIMAL(15) NOT NULL,
      component_id DECIMAL(15) NOT NULL,
      returned_value VARCHAR NULL,
      execution_status CHAR(1) NOT NULL,
      initial_date DATE NOT NULL,
      initial_time TIME NOT NULL,
      final_date DATE NULL,
      final_time TIME NULL,
      CONSTRAINT       pk_np_instance_log       PRIMARY       KEY
(instance_log_id),
      CONSTRAINT fk_np_instance_log FOREIGN KEY (instance_id)
            REFERENCES np_requested_process (instance_id),
      CONSTRAINT fk2_np_instance_log FOREIGN KEY (component_id)
         REFERENCES np_step (step_id));
CREATE UNIQUE INDEX np_instance_log_unique_index ON
      np_instance_log (instance_log_id);
GRANT ALL PRIVILEGES ON np_instance_log TO PUBLIC;

/* Insere na tabela de Passos os operadores existentes na NPDL
*/
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (1, 'O', '.');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (2, 'O', '+');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (3, 'O', '||');
INSERT INTO np_step (step_id, step_type, step_description)
```

```
      VALUES (4, 'O', '|*');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (5, 'O', '%');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (6, 'O', '%!');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (7, 'O', '&');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (8, 'O', '^');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (9, 'O', '?*');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (10, 'O', '?F');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (11, 'O', '?N');
INSERT INTO np_step (step_id, step_type, step_description)
      VALUES (12, 'O', '#');
```

Os comandos da tabela B.2 utilizam apenas a sintaxe do SQL padrão, para que possam ser aplicados sobre bancos mantidos por qualquer SGBDR.

As relações criadas pela *NavigationPlanTool* possuem em seu nome o prefixo "np_", para diferenciá-las das demais relações que o banco de dados acessado possa conter.

A relação **np_process**, representada na tabela B.2, armazena o nome de um processo (coluna **process_description**) e uma descrição da sua funcionalidade (coluna **service_description**). Dados de ações, como nome (coluna **step_description**) e instrução para execução (coluna **execution_call**), são mantidos na relação **np_step**, representada na tabela B.4.

A relação **np_step** também mantém dados dos operadores da NPDL, regras (utilizadas nos operadores "%" e "%!"), funções (utilizadas no operador "?") e números (também utilizados no operador "?"). Regras e funções, assim como as ações, possuem valor para a coluna **execution_call**; operadores e números possuirão sempre NULL como valor para a coluna. Um registro da relação **np_step** pode assumir em sua coluna **step_type** um dos valores indicados na tabela B.5.

Tabela B.3: Relação **np_process**

process_id	process_description	service_description
1	PPagamento	Controla o processo de pagamento…
2	PValidacao	Valida os dados do cliente…
3	PEstoque	Controla estoque…

Tabela B.4: Relação **np_step**

step_id	step_description	step_type	execution_call
1	.	O	NULL
2	+	O	NULL
3	||	O	NULL
4	|*	O	NULL
5	%	O	NULL
6	%!	O	NULL
7	&	O	NULL
...	...	...	...
20	A1	A	EnviaForm.exe
21	A2	A	ProcessaForm.exe
22	R1	R	PossuiCredito.exe
23	F1	F	NumDeItens.exe
24	5	N	NULL
25	A3	A	CancelaRequisicao.exe

Tabela B.5: Tipos possíveis de passos

Valor	Representando um(a)...
A	Ação
R	Regra (função que retorna valor lógico)
F	Função que retorna um inteiro
N	Número inteiro
O	Operador

De acordo com a estrutura definida para as relações **np_process** e **np_step**, a tradução dos comandos NPDL

```
CREATE PROCESS ProcExemplo;
ADD ProcExemplo SERVICE_DESCRIPTION 'Processo exemplo...';
CREATE ACTION AcaoExemplo;
ADD AcaoExemplo EXECUTION_CALL 'AcaoExemplo.exe';
CREATE RULE RegraExemplo;
ADD RegraExemplo EXECUTION_CALL 'RegraExemplo.exe';
```

para SQL é:

```
INSERT INTO np_process (process_id, process_description)
       VALUES (4, 'ProcExemplo');
UPDATE np_process
       SET service_description = 'Processo exemplo...'
       WHERE process_id = 4;
INSERT INTO np_step (step_id, step_description, step_type)
       VALUES (26, 'AcaoExemplo', 'A');
UPDATE np_step SET execution_call = 'AcaoExemplo.exe'
                 WHERE step_id = 26;
INSERT INTO np_step (step_id, step_description, step_type)
```

```
        VALUES (27, 'RegraExemplo', 'R');
UPDATE np_step SET execution_call = 'RegraExemplo.exe'
            WHERE step_id = 27;
```

A relação **np_navigation_plan,** da tabela B.6, armazena os dados do plano de navegação dos processos. Em NPDL, o plano de navegação de um processo é definido por uma expressão algébrica que pode envolver ações, operadores, regras, funções, números e outros processos previamente definidos. A relação **np_navigation_plan** mantém os relacionamentos entre processos e passos por meio das colunas **process_id** e **component_id**. Como o plano de navegação de um processo pode ser recursivo ou envolver outros processos, a coluna **component_id** pode referenciar registros tanto da relação **np_step** quanto da relação **np_process**. A coluna **component_type** indica qual relação o valor do campo **component_id** referencia. Um registro de **np_navigation_plan** pode assumir em sua coluna **component_type** um dos valores indicados na tabela B.7.

Tabela B.6: Relação **np_navigation_plan**

navigation_plan_id	process_id	component_id	component_type
1	1	1	S
2	1	2	S
3	1	20	S
4	1	6	S
5	1	22	S
6	1	21	S
7	1	25	S
8	3	1	S
9	3	1	P
10	3	19	S

Tabela B.7: Tipos possíveis de componentes

Valor	Para component_id referenciando relação...
S	np_step (tabela 2)
P	np_process (tabela 1)

Na relação **np_navigation_plan,** as expressões algébricas dos processos ficam armazenadas em notação prefixa.

A tradução do comando NPDL de definição de plano de navegação do processo “ProcExemplo”

```
SET ProcTest = A2 . (%R1 AcaoExemplo + %!R1 PEstoque);
```

resulta nos seguintes comandos SQL (considerando como dados já cadastrados os relacionados nas tabelas B.3, B.4 e B.6):

```
/* A expressão do processo em notação prefixa é:
   ProcTest = . A2 + %R1 AcaoExemplo  %!R1 PEstoque */
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (11, 4, 1, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (12, 4, 21, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (13, 4, 2, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (14, 4, 5, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (15, 4, 22, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (16, 4, 26, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (17, 4, 6, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (16, 4, 22, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                        process_id,component_id, component_type)
       VALUES (17, 4, 3, 'P');
```

A relação **np_requested_process**, ilustrada pela tabela B.8, representa as instâncias de processos. Cada instância está associada ao seu processo (coluna **process_id**), ao usuário que a solicitou (coluna **user_name**) e à data da requisição (colunas **request_date** e **request_time**).

Tabela B.8: Relação **np_required_process**

instance_id	processo_id	request_date	request_time	user_name
1	3	12/05/2005	22:45:23	user_6
2	3	06/09/2005	17:38:22	user_2
3	1	18/12/2005	09:05:58	user_9

Tabela B.9: Relação **np_instance_log**

instance _log_id	instance _id	component _id	execution _status	initial _date	initial _time	final _date	final _time
1	1	20	F	18/12/2005	10:03:51	18/12/2005	10:03:54
2	1	22	F	18/12/2005	10:05:37	18/12/2005	10:05:38
3	3	15	F	20/12/2005	18:29:17	20/12/2005	18:29:18
4	3	32	F	20/12/2005	18:31:05	20/12/2005	18:31:09
5	3	17	I	21/12/2005	09:46:07		

Cada registro da relação **np_instance_log**, como mostra a tabela B.9, representa a execução de um passo em uma determinada instância de processo. A coluna **instance_id** indica a instancia à qual a execução está associada, **component_id** indica o passo executado e **execution_status** mantém o estado da execução (iniciada, finalizada ou cancelada) do passo. Além disso, um registro do *log* também possui informações sobre a data e hora de início e término da execução e o seu valor de retorno (utilizado no caso da execução de regras ou funções).

Um registro da relação **np_instance_log** pode assumir como valor para a coluna **execution_status** um dos valores descritos na tabela B.1.

É importante ressaltar que a estrutura relacional de dados utilizada pela *NavigationPlanTool* não precisa ser mantida em um banco de dados de acesso exclusivo à ferramenta. A prática mais comum é manter esta estrutura no banco que armazena os dados da aplicação que a utiliza ou em um banco de dados acessado por várias aplicações, com o objetivo de compartilhar as definições de processos.