ANEXO III - EXEMPLO DE FONTE X-COBOL

```
 ID DIVISION.
      :
 DATA DIVISION.
      :
 WORKING-STORAGE SECTION.
      :
@VAR-XCOBOL
@    @TELA SOS-12 LINHA 1Ø COLUNA 5 TAMANHO 3Ø
@                 "TELA DE ERROS............"
@                       :
@                 LINHA 2Ø COLUNA 5 TAMANHO 3Ø
@                 "FIM TELA DE ERROS........"
@
      :
 PROCEDURE DIVISION.
      :
@    @TELA DADOS-12 LINHA.................
@                 :
@    @MOSTRA TELA DADOS-12
@    @OBTEM CAMPO-1 TAMANHO 6 MASCARA "Z.ZZ9,99"
@.                :
@    @OBTEM CAMPO-15 TAMANHO 3Ø NORMAL

 LABEL-X.

     IF SOS THEN

@        @SALVA
@        @LIMPA TELA
@        @MOSTRA TELA SOS-12
@        @ALARME ERRO
@        @RESTAURA
@        @RETOMA

         GO TO LABEL-X.

      :
      :
      :
```


UMA LINGUAGEM DE PROGRAMAÇÃO DE ALTO-NÍVEL BASEADA NA SDL

Marcus Vinícius Midena Ramos
Ericsson do Brasil Comércio e Indústria S.A.
Divisão de Desenvolvimento de Comutação
Departamento de Desenvolvimento de Comutação Particular
Seção de Desenvolvimento I
Setor de Software
Rua da Coroa, 500 - 1º andar
São Paulo – SP
CEP 02047
Tel.: 298-2322 - ramal 1105

Resumo:

O presente artigo apresenta uma nova linguagem de programação, voltada para a implementação de especificações funcionais em SDL (Specification and Description Language). Inicialmente é feita uma introdução à SDL, seguida de uma breve análise de algumas linguagens já existentes. Por último, a caracterização da linguagem, que atualmente se encontra em fase final de especificação.

Palavras-Chave: Máquina de Estado – SDL – Linguagem de Programação.

1. Introdução

O desenvolvimento de sistemas, como por exemplo o de uma central telefônica CPA, passa, invariavelmente, pelas seguintes fases:
1. Especificação, na qual são definidos os requisitos do sistema. Estes requisitos são definidos por alguns parâmetros genéricos (capacidade de comutação, limites de temperatura, etc.) e por uma especificação funcional do comportamento desejado.
2. Implementação do comportamento especificado, num sistema real.
3. Depuração do sistema, com eventual repetição dos passos anteriores.
4. Descrição, consistindo de parâmetros genéricos do sistema real, bem como da descrição funcional do seu comportamento.

No que diz respeito às fases 1 e 4, a linguagem SDL (Specification and Description Language) – Linguagem para Especificação e Descrição de Sistemas, vide 1 e 2 constitui-se numa ferramenta bastante apropriada para a realização das mesmas, e desta forma tem sido utilizada com sucesso no desenvolvimento de Sistemas Telefônicos em geral, sejam eles de pequeno, médio ou grande porte, públicos ou privados.

Apesar desta categoria de sistemas ser a que mais se utiliza da SDL, a sua abrangência é bastante ampla e atende, em última instância, às necessidades de especificação e descrição de máquinas de estado, ou seja, adequa-se a sistemas de controle em geral.

Com relação à implementação, devemos antes traçar um perfil das principais características encontradas nos projetos dos Equipamentos Telefônicos mais recentes, que são aqueles pertencentes à segunda geração de centrais CPA (Controle por Programa Armazenado):
1. Multiprocessamento: devido ao ganho em desempenho e segurança que as arquiteturas deste tipo oferecem. Além disso, a contínua redução nos custos de CPDs (microprocessadores comercialmente disponíveis), memórias e outros componentes também contribui bastante no sentido desta opção.
2. Multiprogramação: é necessária devido à partição lógica requerida para o software da aplicação, de modo a permitir um tratamento mais eficiente de problemas bastante complexos, além de facilitar a depuração, documentação e manutenção do software produzido.

Desta forma, a questão da implementação de uma especificação funcional se traduz pela elaboração do software que realize o conteúdo desta especificação.

Isto implica, de fato, na programação de uma máquina distribuída, operando em tempo real.

O principal objetivo deste artigo é a apresentação de uma linguagem de programação adequada para a implementação de especificações funcionais em SDL.

O próximo item faz uma breve introdução à SDL. Os seguintes desenvolvem os aspectos relativos à caracterização da linguagem, considerada "sob medida" para os futuros desenvolvimentos na área de comutação particular da Ericsson do Brasil S.A.

Este trabalho servirá também de base para o Trabalho de Mestrado do autor do Departamento de Engenharia de Eletricidade da E.P.U.S.P.

2. A Linguagem SDL

A SDL é uma linguagem em forma gráfica utilizada para representar Diagramas de Transição de Estado.

2.1. – SDL Padrão

É definida nas recomendações Z.101 a Z.105 do CCITT e constituída dos seguintes elementos:

. PROCESSOS: são as unidades funcionais mais simples que compõem um sistema. Um processo corresponde à implementação de uma máquina de estados associada (normalmente) a uma tarefa específica de controle. Processos podem ser totalmente implementados em hardware, mas este não é o caso mais comum. Na prática, um sistema é composto por um conjunto de processos (implementados em software) e integrados por meio de troca de sinais entre os mesmos. Cada processo possui um conjunto de estados possíveis. A figura 6 ilustra a representação convencional de uma máquina de estados, e a figura 7 a equivalente em SDL.

. SINAIS: designam um fluxo de informação na máquina de estado (ou entre 2 máquinas de estados). A chegada de um sinal a um processo que se encontra em um estado onde este sinal é previsto, provoca a mudança de estado desse processo (máquina de estados). Tais sinais podem transportar dados que são denominados "dados associados ao sinal". A figura 2 mostra os símbolos que representam sinais.

. ESTADOS: caracterizam uma situação de suspensão das atividades de processo aguardando um sinal de entrada (estados de espera). Em cada estado (figura 1) podem ser reconhecidos um ou mais sinais (sinais de entrada, figura 2b), conforme a figura 7.

. TRANSIÇÕES: uma transição (figura 7) consiste de uma seqüência de ações que devem ser executadas quando da transição entre dois estados específicos de um processo. São representadas na forma de fluxogramas convencionais, utilizando-se os símbolos que designam tarefas (figura 4), decisões (figura 3) e também envio de sinais (figura 2c). Desta forma, cada par (estado de espera, sinal de entrada) estará associado a uma particular transição.

. SAVE: indica o retardamento do reconhecimento de um sinal de entrada num certo estado. O processo não muda de estado com este sinal, e o seu reconhecimento é retardado até a próxima mudança para um estado distinto (figuras 5 e 7).
. BLOCOS FUNCIONAIS: são definidos a título de obtenção de um Modelo Funcional do Sistema que se pretende especificar. A consecutiva partição de BFs em novos BFs mais simples e a sua posterior realização física, leva ao conceito de Bloco de implementação (BI). Cada BI comporta um certo número de processos. É conveniente associar cada BI a um único tipo de recurso, modelando desta forma o comportamento do mesmo (por exemplo, troncos, terminais, canais de comunicação, etc.).

2.2 – Extensões

O conceito de indivíduo (ou distância) é uma importante extensão do SDL padrão, pois muito freqüentemente nos deparamos com sistemas (e isto é uma característica dos Sistemas Telefônicos) nos quais existe um certo número de entidades (troncos, por exemplo) idênticas do ponto de vista funcional e que desta forma estão associadas a um mesmo Diagrama de Transição de Estados (isto é, processo).
Em outras palavras, indivíduo é um conceito que permite utilizar uma mesma descrição (descrição de processo) para uma família de entidades similares. Desta forma, cada processo (ou mais genericamente, bloco de implementação) poderá possuir mais de um indivíduo de um mesmo tipo, sendo que num certo instante, cada um deles poderá se encontrar num estado diferente.
Considerando que especificações funcionais serão implementadas como programas, os quais eventualmente necessitarão de área para armazenamento de informações diversas, introduziremos os termos "área individual" e "área comum":
. ÁREA INDIVIDUAL: conjunto de dados associados a um indivíduo de um BI. Existirão tantas áreas individuais quantos forem os indivíduos deste BI. Os dados individuais e o estado de um indivíduo caracterizam de forma completa o estágio atual de processamento do mesmo, num certo instante.
. ÁREA COMUM: é definida como sendo um conjunto de dados comuns a todos os indivíduos de um mesmo BI.
Uma outra importante extensão é o conceito de Processo "Sombra". Trata-se de um processo que não possui indivíduos, e que tem a propriedade de poder manipular as áreas de dados individuais de todos os indivíduos do BI ao qual ele pertence. Sua definição se justifica pelas necessidades de iniciação ou varredura destas áreas, sem que se incorra em tráfego excessivo de sinais (figura 8).

3. Requisitos da Linguagem de Programação

Até o presente momento, todo o desenvolvimento de software efetuado em nosso laboratório de comutação particular tem sido feito em assembly, utilizando um programa assembler dedicado com o tratamento de algumas pseudo-instruções, que tornam viável a sua utilização para a implementação de especificações funcionais em SDL.
Entretanto, com a perspectiva de desenvolvimento de projetos de maior porte, torna-se imperiosa a necessidade de opção por um Sistema de Programação adequado. Como parte constituinte deste sistema deverá existir uma linguagem de programação de alto nível que atenda aos seguintes aspectos genéricos:
a) Proporcionar redução nos custos de desenvolvimento e manutenção de software
b) Aumentar o nível de confiabilidade dos programas produzidos.
c) Tornar programas mais legíveis.
d) Ser independente de hardware.
Além disso, os seguintes requisitos mais específicos também foram formulados:
a) Permitir uma transposição clara dos conceitos da SDL (principal requisito).
b) Não possuir elementos nem construções irrelevantes e de complexidade desnecessária que de outro modo contribuiriam apenas para uma implementação mais problemática, bem como tornariam sua utilização menos imediata e eficiente (vide exemplos no item seguinte).
c) possuir ferramentas que a torne apropriada ao ambiente operacional a que ela se destina (manipulação de estruturas de baixo nível), como por exemplo bits, controle de dispositivos, etc.).

4. Breve Análise das Linguagens Existentes

Tendo em mente os objetivos relacionados no item anterior, foram avaliadas as linguagens de programação CHILL (CCITT High Level Language, referencias 3 e 4, PLEX (Programming Language for Exchanges, referências 5 e 6, e ERIPASCAL (desenvolvida pela Ericsson para a programação de aplicações no sistema APN167).
Todas elas são linguagens de alto nível, adequadas à programação de sistemas distribuídos e que operam em tempo real.
Entretanto, elas não possuem uma representação clara de todas as estruturas da SDL. O conceito de Estado, por exemplo, não é tratado explicitamente, como convém aos sistemas ora desenvolvidos. Nestas linguagens este conceito fica implícito em operações de recebimento de sinais (RECEIVE e WAIT). Fica, portanto, caracterizado um estado de espera, que no entanto não pode ser referenciado simbolicamente, o que acaba por prejudicar a legibilidade dos programas produzidos.
A idéia de que transições devam ser agrupadas por estado para então se chegar ao processo, não transparece na forma de construções sintáticas destas linguagens, o que poderia ser um fator atuando no sentido da não homogeneidade da documentação das unidades funcionais que compõem um programa.
Outro ponto importante é que estas linguagens representam um superdimensionamento com relação às nossas expectativas.
Exemplos disso são a possibilidade de concorrência efetiva entre processos (em SDL apenas uma transição de um processo está ativa a cada instante), os múltiplos e elaborados mecanismos para comunicação e sincronização de processos e o tratamento de números reais.
Além disso, linguagens como PLEX e CHILL perdem em legibilidade quando comparadas, por exemplo, com ERIPASCAL.
Finalmente, pelo fato de se tratarem de linguagens bastante extensas e complexas, a sua implementação num Sistema de Desenvolvimento de pequeno porte talvez se defrontasse com problemas específicos (reduzida capacidade de armazenamento primário e secundário, baixa velocidade de processamento) com possíveis implicações na eficiência do processo de compilação.

5. Caracterização da Nova Linguagem

A nova linguagem se propõe a, antes de mais nada, permitir a implementação de especificações funcionais em SDL de uma forma tão direta quanto possível.
De acordo com a linguagem, uma determinada aplicação é constituída por um ou mais Blocos de Implementação (BIs). Cada BI estará associado a um único tipo de recurso.
O tipo de indivíduo tratado por um BI é definido pelo algoritmo por ele implementado e também pela estrutura de dados pertinente.
Os BIs são entidades fechadas. Os únicos nomes que são exportados (automaticamente) por todos os BIs de uma aplicação são os seus nomes e também os dos processos neles contidos. Isto ocorre devido à necessidade de especificação de destino nas operações de envio de sinal (que também envolve a informação de indivíduo, se for o caso).
A figura 9 apresenta o layout de uma aplicação e também o formato genérico de um bloco de implementação.
Como se percebe, um BI é composto de pelo menos um processo, uma área de dados individuais, uma de dados comuns e, eventualmente, procedimentos e/ou funções globais (aos processos do BI).
Um BI possui um certo número (constante) de indivíduos. Isto implica em que todos os seus processos possuem este mesmo número de indivíduos. A exceção fica por conta dos processos "sombra", que conforme foi mencionado, não possui indivíduos.
Processos de um mesmo BI deveriam se comunicar exclusivamente por meio de sinais (assim como ocorre com processos de BIs distintos). Entretanto, em certos casos particulares, esta comunicação poderá ser feita através das áreas comum e individual do BI comum aos processos.

5.1 – Estruturação de Programa

A figura 10 apresenta a estrutura de um programa-exemplo, onde algumas das principais unidades funcionais têm sua presença registrada apenas pelo emprego da(s) palavra(s) reservada(s) que as introduz (estas plavras estão sublinhadas para maior facilidade de identificação).
A linguagem foi desenvolvida a partir da PASCAL (referência 8), da qual foram aproveitados os conceitos de estruturação de dados e comandos, sintaxe de várias construções sintáticas (procedimentos, funções, comandos, declarações de constantes, tipos e variáveis) e regras que definem o escopo dos nomes.
Analisando a figura 10, temos que o cabeçalho do BI define o nome do mesmo e também o número de indivíduos do(s) processo(s) que o compõe(m).
A seguir, a declaração (opcional) de alguns nomes globais: constantes (iniciada pela palavra reservada CONSTANT), tipos (TYPE), variáveis comuns (COMMON) e individuais (INDIVIDUAL). Assim como a PASCAL, estas unidades funcionais são opcionais e, se presentes (no máximo uma vez), devem obedecer à ordem especificada.
Opcionalmente podem ser declaradas procedures e/ou funções (globais) também na forma habitual da PASCAL. Variáveis locais destas unidades são declaradas a partir da palavra LOCAL.
Constantes e tipos também podem ser locais.
Por último, o BI deverá conter a definição de ao menos um processo.
Em sendo o BI uma entidade fechada, ele é a unidade de compilação.

5.2. Tipos, Operações e Funções

Assim como em PASCAL, são definidos os tipos simples: inteiro, booleano, caracter, além de intervalos e funções. Números reais não são tratados. Por outro lado, o tipo inteiro é bastante versátil: podem ser definidos inteiros de 8, 16 ou 32 bits, com ou sem sinal.
Outra extensão é o tipo BINARY (<número de bits>), representando uma cadeia de bits sobre os quais se definem operações lógicas.
Os tipos estruturados englobam os já conhecidos records, arrays e sets. Uma extensão do conceito de tipo BINARY é o da estrutura BITSTRING. Trata-se de uma estrutura semelhante ao records tradicional, apenas que todos os seus campos devem ser posicionados de acordo com as informações de número de ordem do bit inicial e do bit final do mesmo. A versatilidade reside no fato de que estes campos podem ser sobrepostos, conforme o exemplo abaixo:

```
. STATUS: BITSTRING
            REG [ 0. . .2 ]: BINARY;
            CONTROL [ 5. . . 6 ] : (AUTO, MANUAL, OFF);
            DADO [ 8. . . 11 ] : 0. . . 15;
            DADO-HIGH [ 8. . .9 ] : 0. . .3;
            DADO-LOW [ 10. . .11 ] : 0. . .3
        END;
```

Sobre os tipos inteiros são definidos os operadores aritméticos tradicionais (+, –, *, DIV, MOD) e sobre os booleanos, além dos bastante utilizados (NOT, OR, AND e XOR), os mais recentes mas bastante úteis operadores condicionais: AND THEN e OR ELSE. Estes operadores implementam a avaliação condicional (em tempo de execução) de expressões booleanas.
As funções SHIFTRIGHT e SHIFTLEFT efetuam o deslocamento de valores do tipo BINARY. Ainda, a função BOOL transforma em booleano o valor de um determinado bit de uma variável BINARY:

```
COMMON STATUS : BINARY(5);
        B : BOOLEAN;
BEGIN
        B : BOOL (STATUS,3)
```

Coerções (conversões implícitas de tipo) só são executadas nos casos onde seguramente não possa ocorrer perda de informação (como é o caso da transformação INTEGER → LONGINT). Conversões explícitas entre quaisquer tipos simples poderão ser feitas referindo-se ao tipo para o qual se deseja a conversão:

```
INDIVIDUAL A, B : INTEGER;
C : LONGINT ;
BEGIN A : = B + INTEGER (C)
```

5.3 – Processos

Os processos também podem declarar nomes locais, cujo escopo é definido por ele mesmo. As procedures/funções assim definidas são locais ao processo e são de uso exclusivo do mesmo. As variáveis locais do processo (LOCAL) não retêm seus valores entre ativações consecutivas do mesmo (isto é, se perdem ao fim de cada transição). Um processo fica caracterizado quando se define a sua matriz de estados x sinais e as transições correspondentes.

5.3.1 – Matriz Estados x Sinais

A sua função é relacionar todos os sinais (através de seus nomes) que são reconhecidos em cada estado do respectivo processo.
A declaração da matriz estados x sinais serve também como meio para declaração de todos os estados que compõem o processo (figura 10). Já os sinais são declarados externamente, numa Biblioteca de Sinais (vide item 5.5).
Cada par (estado, sinal) deverá ser provido, na unidade funcional seguinte (transições) com um trecho de código correspondente.
A utilização da cláusula OTHERWISE (opcional) na matriz é equivalente a repetição do(s) nome(s) do(s) sinal(is) nela relacionados em todos os estados do processo (significando que o sinal passa a ser reconhecido em todos os estados nos quais ele não fora declarado explicitamente). A diferença é que haverá uma transição única (independente de estado) para todos os sinais declarados através da OTHERWISE.

5.3.2 – Transições

Definem o trecho algorítmico do processo, ou seja, a seqüência de ações que deve ser executada nos instantes imediatamente anteriores à mudança de estado de um indivíduo.
Assim como a matriz estados x sinais, as transições estão agrupadas por estado e sinal, devendo obedecer a mesma ordem de declaração observada na matriz (figura 10).
Uma transição é iniciada pelo nome do sinal de entrada que o dispara, uma relações de variáveis associadas e o trecho de código correspondnte.

Estas "variáveis associadas" nada mais são do que variáveis locais à transição e que são declaradas com a finalidade de possibilitar o recebimento e a manipulação dos dados associados do respectivo sinal de entrada. Elas deverão ser em quantidade e tipos compatíveis com a declaração do sinal constante da Biblioteca de Sinais (item 5.5).

Eventualmente, a declaração de uma ou mais destas variáveis locais à transição poderão ser substituídas pelo caracter especial "*", significando que o correspondente dado associado do sinal não é relevante ao processamento executado pela respectiva transição.
O compartilhamento de código por mais de uma transição só será possível se isto for feito via procedimentos/funções. Isto ocorre por motivos de escopo dos nomes das variáveis locais à transição, bem como por motivo de uma melhor estruturação e manutenção do software produzido.

Opcionalmente, todo o trecho de comandos de uma transição poderá ser substituído pela palavra SAVE, significando que o sinal recebido deverá ser retardado até a próxima mudança de estado do respectivo indivíduo.

As transições que se seguem à palavra OTHERWISE correspondem aos sinais declarados na matriz estados x sinais na cláusula OTHERWISE.

5.3.3 – Processo Sombra

Cada BI poderá ter ou não processos sombra, que são declarados de forma: SHADOW PROCESS < nome do processo >.
Estes processos não possuem indivíduos e são capazes de acessar a área de dados individuais de todos os indivíduos do BI.
Para que um comportamento destes seja alcançado (figura 8), todas as variáveis individuais referenciadas por um processo sombra deverão possuir uma dimensão a mais do que o número original de dimensões correspondentes (constantes em INDIVIDUAL). Esta é a dimensão dos indivíduos do BI, e é do tipo (utilizando a notação de intervalo da PASCAL):
1. . . < número de indivíduos do BI >
Por exemplo:

```
INDIVIDUAL: A : SHORTINT;
            B : ARRAY [ 1..5, 3...15 ] OF INTEGER;
                .
                .
                .
SHADOW PROCESS P;
                ,
                ,
                ,
BEGIN
                .
                .
                .
    IF A [2] = Ø. . .
    B [ 3, 4, 1 ] : ^ 1200;
                .
                .
                .
```

5.4 – Primitivas

São agrupadas em primitivas de manipulação de sinais e de manipulação de estados. Novas primitivas podem ser definidas em uma Biblioteca de Primitivas (vide item 5.5).

As primitivas de manipulação de sinais englobam:
I) envio de sinal : SEND < nome do sinal > ["C" < dados associados > ")"] [TO < destino do sinal >].
O termo < dados associados > denota uma série de expressões compatíveis com a definição do sinal, constante da Biblioteca de Sinais.

O campo que especifica o destino do sinal poderá ser total ou parcialmente omitido, casos em que serão utilizados valores default para a especificação completa do mesmo (composto por nome de BI, de processo e número do indivíduo).

Exemplo: "SEND" FOFOGA "TO PROCESS" = PCTERMN, "IND" = 2. Neste caso, assume-se (e verifica-se) que o BI destino (Block = . . .) é (seja) o mesmo ao qual esta primitiva pertence.
II) temporização de sinal: TEMP < nome do sinal > ["(" < dados associados > ")"] [TO < destino do sinal >] WITH TIME = < informação de tempo >.
Neste caso, o sinal só será escalado após vencida a condição de tempo especificado (tempo absoluto ou relativo).
III) cancelamento de temporização de sinal : CAN < nome do sinal > [TO < destino do sinal >].
As primitivas de manipulação de estado são:
I) mudança de estado : NEXT < nome do estado >. O indivíduo para o qual a transição é executada muda para o estado definido por < nome do estado > Só é permitida a mudança para estados de um mesmo processo.
II) mudança para o mesmo estado : SAME. É um caso particular de NEXT, e sua definição se justifica pela existência de transições OTHERWISE (no qual não se conhece o estado atual).

As primitivas NEXT e SAME, além de efetuar mudanças de estado, sinalizam o final dinâmico de uma transição (após a sua execução, o controle é automaticamente transferido para o Sistema Operacional).

5.5. Bibliotecas

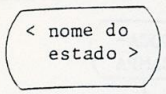
Existem duas bibliotecas, referenciadas em tempo de compilação, que apesar de não serem partes consistentes da linguagem propriamente dita, serão a seguir brevemente descritas, com o intuito de introduzir alguns aspectos de Metodologia de Programação idealizada:
I) A Biblioteca de Sinais comporta os descritores de todos os sinais utilizados por uma Aplicação. Um descritor é composto por nome de sinal e estruturação dos dados associados (quantidade e tipos). O propósito desta Biblioteca é o de centralizar as definições dos sinais, o que é vantajoso sobre diversos aspectos: maior confiabilidade e eficiência na troca de informações entre usuários dos mesmos sinais, realizações de testes de consistência sobre a utilização de sinais (operações de envio e recebimento) pelo compilador, elaboração e manutenção de uma documentação dos sinais.
Esta Biblioteca sofrerá edições, à medida em que sinais tiverem suas definições alteradas, mais sinais forem criados ou mesmo se antigos forem suprimidos.
II) A Biblioteca de Primitivas também contém apenas descritores. Estes descritores são descritores de procedimentos (nome e parâmetros formais) definidos na versão mais recentemente implantada do Sistema Operacional, e portanto, disponíveis aos programas de aplicação na forma de simples chamadas de procedimentos.
O propósito desta Biblioteca é o de tornar a linguagem independente de um conjunto restrito e básico de primitivas (que possuem inclusive sintaxe própria como por exemplo a SEND, NEXT, etc.). Como conseqüência, a linguagem ganha tanto em adaptabilidade a novos sistemas (caracterizados pelos respectivos Sistemas Operacionais) como na capacidade de acompanhar a evolução de uma particular aplicação no que tange à interface com novas versões de um mesmo Sistema Operacional.

6. Conclusão

No presente momento, a linguagem proposta encontra-se em fase final de especificação. Os próximos passos englobam a caracterização de um Sistema Operacional que dê suporte à linguagem, bem como à construção de um compilador para a mesma. Outras atividades envolvidas são a definição e a implementação dos Gerenciadores das Bibliotecas.
Apesar da linguagem ser independente de máquina, o primeiro Gerador de Código será para o microprocessador M6809 da Motorola. Trata-se de uma CPU que possui um conjunto bastante bom de instruções associado de modo homogêneo a um grupo versátil de modos de endereçamento, o que a torna adequada para a implementação de linguagens do nível da PASCAL.

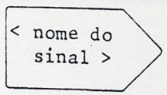
Bibliografia

1. CCITT, "FUNCTIONAL SPECIFICATION AND DESCRIPTION LANGUAGE", RECOMMENDATION Z.101 – Z.105, Vol. VI, Fascículo VI.7 – Yellow Book, Genebera, Nov. 1980.
2. ———, "FUNCTIONAL SPECIFICATION AND DESCRIPTION LANGUAGE (SDL)", DRAFT RECOMMENDATION Z.101 – Z.105, DOCUMENT AP VII n. 20-E, 99 páginas, june 1980.
3. ———, "CCITT HIGH LEVEL LANGUAGE (CHILL)", RECOMMENDATION Z.200, Vol. VI, Fascículo VI.8 – Yellow Book.
4. ———, "INTRODUCTION TO CHILL, THE CCITT HIGH LEVEL LANGUAGE", DRAFT RECOMMENDATION Z.200, 98 páginas, MAY 1980.
5. TELEFONAKTIEBOLAGET LM Ericsson, "FLEX PROGRAMMING LANGUAGE", DOCUMENTO INTERNO, 80 páginas, Feb. 1979.
6. ———, "PROGRAMMING OF CP", DOCUMENTO INTERNO, 151 páginas, Aug. 1979.
7. ———, "THE PROGRAMMING LANGUAGE ERIPASCAL", DOCUMENTO INTERNO, 147 páginas, Feb. 1982.
8. WIRTH, N., "THE PROGRAMMING LANGUAGE PASCAL", ACTA INFORMÁTICA, 1, 29 páginas, 1971.



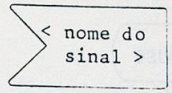
ESTADO

Figura 1



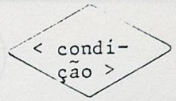
SINAL DE SAÍDA

Figura 2a



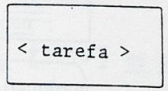
SINAL DE ENTRADA

Figura 2b



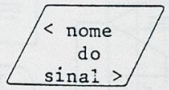
DECISÃO

Figura 3



TAREFA

Figura 4



SAVE

Figura 5

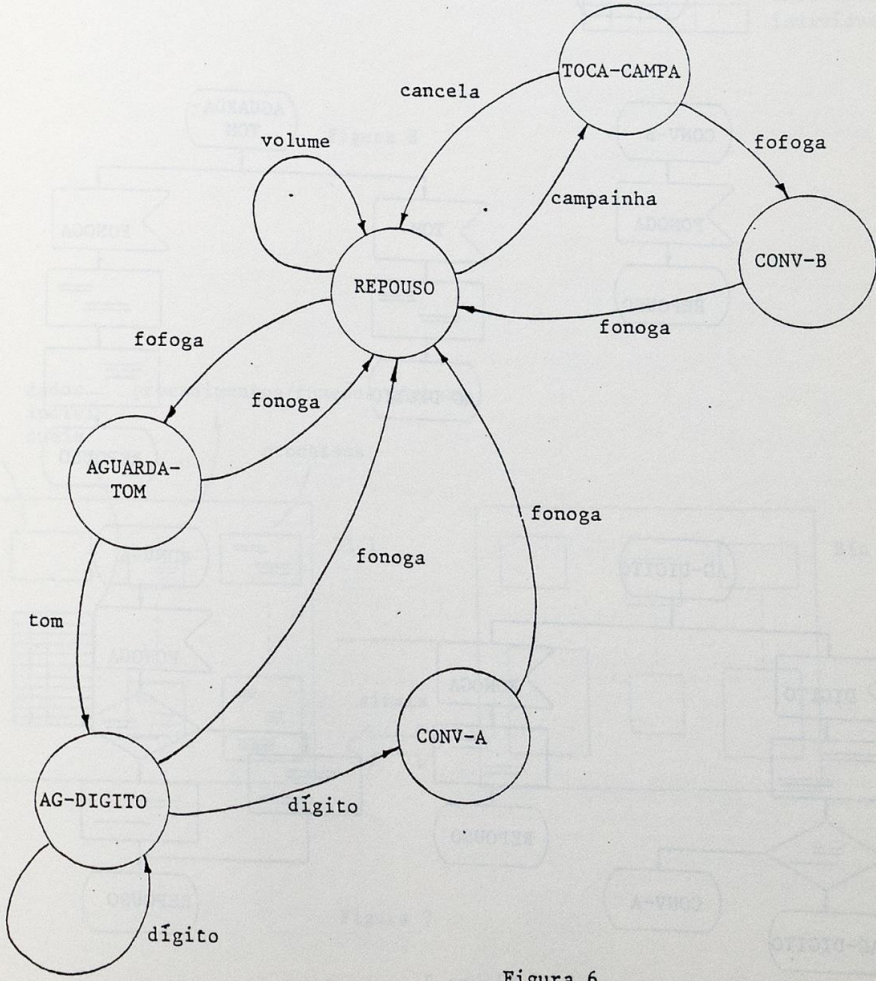


Figura 6

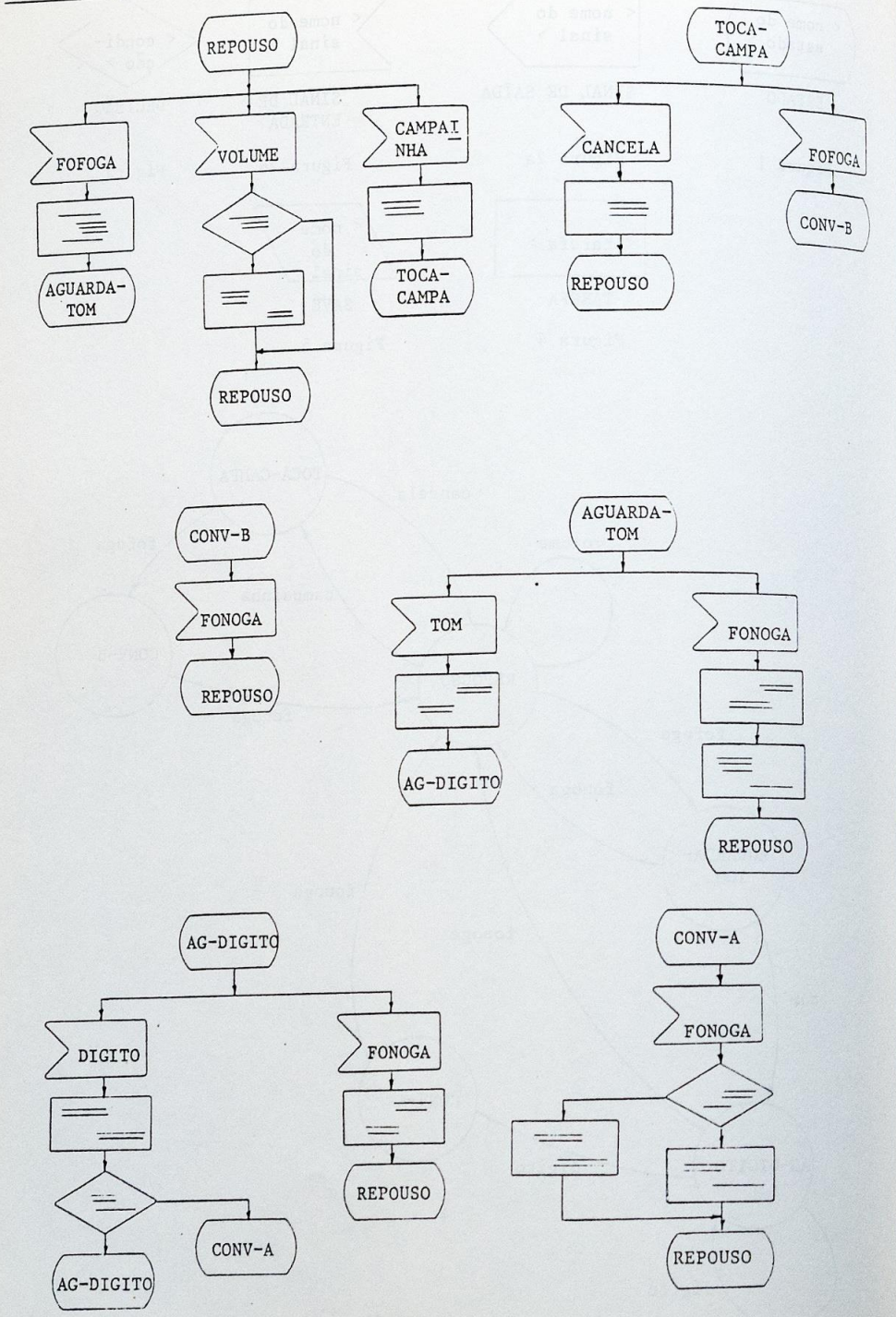


Figura 7

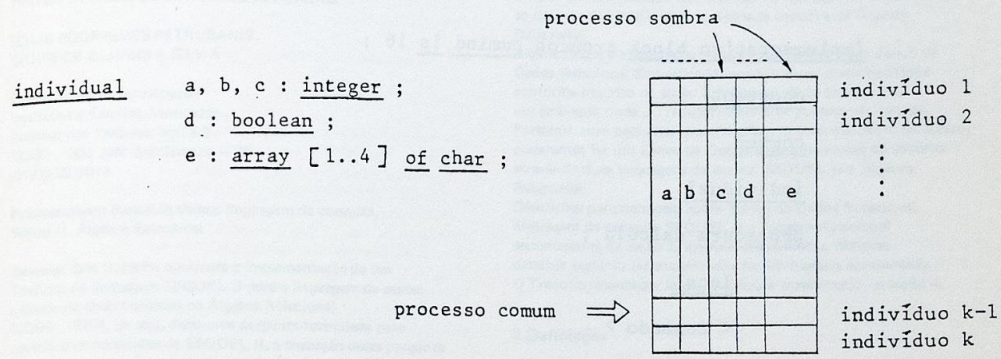


Figura 8

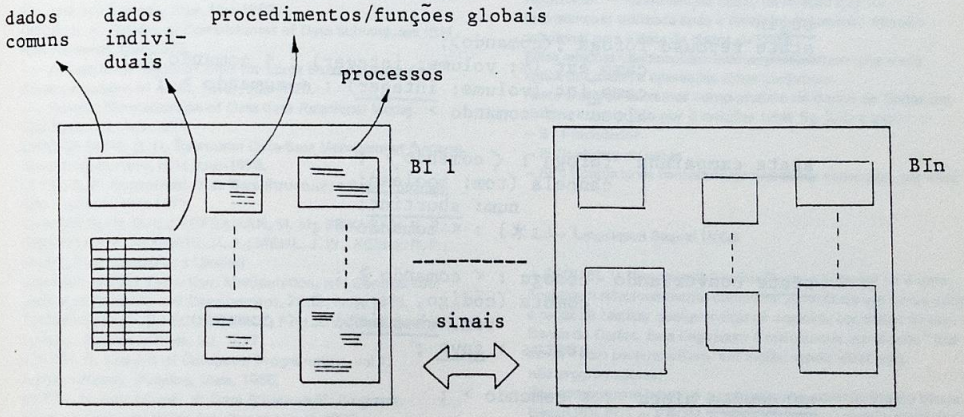


Figura 9

```
implementation block troncos numind is 16 ;

    constant

    common

    individual

    procedure calcula ;

        local

        < comando >

    process controla ;

        constant

        local

        procedure testa ;

            < comando >

    state repouso signal fofoga, camp_ext, camp_int, aloca ;
    state campainha signal fofoga, cancela ;
    state conversando signal fonoga, comuta, status ;
    otherwise signal aloca ;.

    begin
    state repouso fofoga :<comando>;
                  camp_ext (*; volume: integer) : < comando > ;
                  camp_int (volume: integer) : < comando > ;
                  aloca : < comando > ;

    state campainha  fofoga : < comando > ;
                     cancela (tom: boolean;
                              num: shortint;
                              *) : < comando > ;

    state conversando  fonoga : < comando > ;
                       comuta (código,
                               tom: shortint) : < comando > ;
                       status : save ;

    otherwise aloca  : < comando > ;
    end
```

Figura 10

UM TRADUTOR DA LINGUAGEM SEQUEL II PARA A ÁLGEBRA RELACIONAL

BOLIS RODRIGUES PETRUSANIS
ORION DE OLIVEIRA SILVA

Centro Técnico Aeroespacial
Instituto de Estudos Avançados
Rodovia dos Tamoios, Km 5,5
12200 São José dos Campos – SP
(0123) 22.5011

Palavras-chave: Banco de dados, linguagem de consulta, Sequel II, Álgebra Relacional

Resumo: Este trabalho apresenta a implementação de um Tradutor da linguagem SEQUEL II para a linguagem de acesso a Banco de Dados baseada na Álgebra Relacional (CODD, 1970), ou seja, dada uma pergunta formulada pelo usuário com comandos da SEQUEL II, a tradução dessa pergunta resultará em comandos equivalentes da Álgebra Relacional.

Referências Bibliográficas

GRIES, D. Compiler Construction for Digital Computers. New York, 1971
AHO, A. V.; ULLMAN, J. D. Principles of Compiler Design. 2.ª ed. Addison-Wesley, Reading, Mass, 1978.
McKEEMAN, W. M.; HORNING, J. J.; WORTMAN, D. B. A Compiler Generator. Englewood Cliffs, NJ, Prentice-Hall, 1970.
DATE, C. J. An Introduction to Database Systems. 2.ª ed. Reading, Mass, Addison-Wesley, 1977.
HARTMANN, A. C. A Concurrent Pascal Compiler por Mini-Computers. Berlin, Spring-Verlag, 1977.
PEMBERTON, S. Comments on an Error-Recovery Scheme by Hartmann Software-Pratice and Experience, 10(3), Mar-1980.
OLIVEIRA, O. M. Um tradutor para a linguagem Sequel 2/Query. São José dos Campos, Inpe, Mar-1982.
CODD, E. F. Relational Completeness of Data Sublanguage IBM Research, Mar-1972.
—— A Relational Model of Data for Large Share Data Banks. Communications of ACM, 13(6), Jun-1970.
—— Further Normalization of Data Base Relational Model. IBM Research, Ago-1971.
CHAMBERLIN, D. D. Relational Data-Base Management Systems. Computing Surveys, 8(1), Mar-1976.
CODD, E. F. Normalized Data Base Structure: a Brief Tutorial. IBM Research, Nov-1971.
CHAMBERLIN, D. D.; ASTRAHAN, M. M.; ESWARAN, K. P.; GRIFFITHS, P. P.; LORIE, R. A.; MEHL, J. W.; RESNEIR, P.; WADE, B. W. Sequel 2: a Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Development, 20(6), Nov. 1976.
TSICHRITZIS, D. C.; LOCHOVSKY, F. H. Data Base Management Systems. Academia Press, NJ, 1977.
KNUTH, D. The Art of Computer Programming, vol 1. Addison-Wesley, Reading, Mass, 1968.
WIRTH, N. Algorithms + Data Structure = Programs. 1.ª ed., Englewood Cliffs, NJ, Prentice-Hall, 1976.
PRAT, T. W. Programming Languages: Design and Implementation. Englewood Cliffs, NJ, Prentice-Hall, 1975.

1. Introdução

A SEQUEL II foi elaborada e é extensivamente usada como uma linguagem de acesso a Banco de Dados Relacional, tanto para programadores profissionais como para usuários não especializados em programação.
Excetuando-se o uso de conceitos de variáveis limitantes e quantificadores, a SEQUEL II apresenta um conjunto de operações simples em tabelas, que tem o poder de recuperação equivalente ao Cálculo dos Predicados de primeira ordem e da Álgebra Relacional.
Alguns desses comandos da Álgebra Relacional e um Banco de Dados Relacional Simplificado, encontram-se implementados conforme descrito na seção 3. Visou-se, desta forma, criar um ambiente onde o Tradutor/Montador pudesse ser testado.
Portanto, com pequenas modificações no gerenciamento de acesso, poderemos ter um Banco de Dados acessado, a nível de usuário, através de duas linguagens de acesso: SEQUEL II e Álgebra Relacional.
Descrições parcimoniosas sobre Banco de Dados Relacional, linguagem de consulta SEQUEL II e Álgebra Relacional encontram-se na seção 2, intitulada Definições. Maiores detalhes poderão ser encontrados na Bibliografia apresentada.
O Tradutor/Montador implementado é apresentado na seção 4.

2. Definições

2.1 – Banco de Dados Relacional

Um Banco de Dados é uma coleção de dados operacionais armazenados, usados pelos sistemas de aplicação de alguma empresa particular, sistemas estes, que por intermédio de seus usuários, realizam operações de recuperação, alteração, remoção e inserção de informações.

2.2 – Linguagem de Consulta

A manipulação de um Banco de Dados está intimamente vinculada à linguagem de consulta ao banco. Quanto mais eficiente for uma linguagem de consulta, melhor seu manuseio no aprendizado. A eficiência de uma linguagem de consulta liga-se à forma: "eficiência – linguagem – usuário". Vide Figura 2.1.
As linguagens em questão – SEQUEL II/QS e "Álgebra Relacional" – fazem uso da noção de relação que foi intensamente utilizada após o desenvolvimento do modelo relacional para a base de dados de Codd.
Estas relações (associações) unárias possibilitam uma visão lógica dos dados e operações sobre conjuntos.
Neste trabalho usaremos como modelo de Banco de Dados um sistema constituído por 3 relações (vide fig. 2.2) a saber:
– S : Fornecedor
– P: Conjunto de peças
– SP : Conjunto de fornecedores e as peças fornecidas por eles.

2.3 – Linguagem Sequel II/QS

A SEQUEL II (Structured English Query Language II) é uma linguagem relacionalmente completa. Através de um único comando é capaz de realizar qualquer tipo de consulta aos dados de um Banco de Dados. Essa linguagem é estruturada em blocos "top down" com palavras-chave em inglês, sendo ideal para não programadores.
Seu componente básico é o mapeamento constituído do bloco básico SELECT – FROM – WHERE onde o usuário seleciona as colunas (cláusula SELECT) que deseja, em que tabela as colunas são selecionadas (cláusula FROM) e as condições de cada consulta (cláusula WHERE). No apêndice A é apresentada a sintaxe utilizada neste trabalho para a SEQUEL II/QS (Query Statement).
A forma geral do comando é a seguinte: SELECT (UNIQUE)
(lista de atributos)
FROM nome-da-relação
(WHERE qualificação)
(GROUP BY nome-de-atributo (HAVING qualificação)
( ) indica cláusulas opcionais.