JAIIO 16

16ª JORNADAS ARGENTINAS DE INFORMATICA E INVESTIGACION OPERATIVA

TRABAJOS PRESENTADOS

BUENOS AIRES
8 AL 12
DE SEPTIEMBRE
DE 1986

SOCIEDAD ARGENTINA DE INFORMATICA
E INVESTIGACION OPERATIVA

JAIO

16

16as JORNADAS ARGENTINAS DE INFORMATICA E INVESTIGACION OPERATIVA

TRABAJOS PRESENTADOS

BUENOS AIRES
8 AL 12
DE SEPTIEMBRE
DE 1986

SOCIEDAD ARGENTINA DE INFORMATICA
E INVESTIGACION OPERATIVA

Anales de las 16as Jornadas Argentinas
de Informática e Investigación Operativa
SADIO, Buenos Aires, 1986

Um sistema especialista para o Método de Jackson.

Theonilla Estelitta C. Pessoa+, Raul César Baptista Martins* e Daniel Shwabe+.

+Dept. de Informática PUC/RJ
Rua Mq. de São Vicente 225
22453 Rio de Janeiro-RJ
*CTS - IBM BRASIL
Rua Tutóia n 1157/8 andar
04007 São Paulo-SP
BRASIL

RESUMO.

Este trabalho apresenta os resultados de pesquisa na construção de sistemas especialistas para especificação de software. A pesquisa embora tenha sido realizada tanto para programação em ponto grande quanto para programação em ponto pequeno, concentrou-se muito mais na última devido ao grande número de metodologias disponíveis (1, 2, 3) que buscam sua automatização

Em particular a Metodologia de Jackson para Programas tem sido utilizada para resolver problemas relacionados com arquivos sequenciais com bastante sucesso. Todavia sua apresentação com duas exceções (4, 5) tem sido através de exemplos, cada grupo com sua heurística particular.

Apresenta-se o projeto de um sistema especialista que auxilia o programador inexperiente a desenvolver a especificação de programas através do método de Jackson. O sistema usa como entrada os diagramas de Jackson que descrevem as entradas e saídas do problema, a descrição funcional dos elementos que constituem o arquivo de saída e a descrição física do arquivo de entrada. O sistema fornece como saída o diagrama do programa.

Um protótipo está disponível e problemas típicos como " linha balanceada, lotes encaixantes, contagem de lotes" foram resolvidos. O sistema está implementado em programação lógica em particular em micro-Prolog.

1. INTRODUÇÃO.

Uma forma de diminuir o estrangulamento no desenvolvimento de software é o uso de métodos e ferramentas que possibilitem a solução mais precisa e rápida dos novos problemas de aplicação. Dentro desse aspecto a automatização parcial do ato de programar é um fator desejável. Técnicas como documentação, estruturação e padronização de programação, onde padronização aqui significa que usuários diferentes chegam a mesma solução de um dado problema, podem ser consideradas como primeiro passo para esta automatização.

Acoplar técnicas de inteligência artificial (I.A.) (6, 7, 8) e, especialmente, técnicas de sistemas especialistas à construção de programas parece ser promissor, desde que se tenha uma metodologia de programação que permita a sistematização do processo de construir programas.

Um sistema de C.A.S.D.(Computer Assisted Software Design) tem todas as características que o tornam elegível para a construção de um sistema especialista (9) que auxilie o programador.

Das metodologias correntes, Constantine e Yourdon (2), Warnier (3) e Jackson (1), a de Jackson é a que permite a melhor sistematização do processo de construção de programas, pois (10):

1. é uma metodologia simples e que possui uma formalização bem conhecida.
2. é uma metodologia gradual no sentido que o seu aprendizado pode ser feito a partir de exemplos simples até chegar aos mais complexos.
3. é uma metodologia de desenvolvimento de programas bem caracterizável tanto quanto a seu domínio, quanto a apresentação de soluções
4. é uma metodologia particionável e mensurável no sentido em que pode ser descrita em passos bem caracterizados (1).

Descreve-se a seguir um sistema especialista para o auxílio a especificação de programas pelo Método Básico de Jackson. Apresenta-se a estrutura interna da ferramenta, sua base de conhecimento, regras constituintes e maneira de operar. Apresenta-se também um exemplo onde um problema é desenvolvido.

2.A FERRAMENTA

2.1.NOÇÕES GERAIS

A solução de um problema (geração de um diagrama de programa) é feita em quatro fases:

1. Definição das entradas e saídas
2. Testes dos diagramas fornecidos.
3. Geração do diagrama de programas.
4. Testes do diagrama gerado.

Para a definição das entradas e saídas além de serem fornecidos os diagramas de entradas e saídas típicos do método de Jackson,devem ser criadas especificações físicas que correspondem a estrutura interna das folhas das árvores de entrada e, especificações funcionais que definem quais elementos dos arquivos de entrada dão origem a um determinado elemento do arquivo de saída. Às especificações funcionais foi acrescentado o conceito de simultaneidade de geração. Simultaneidade de geração acontece quando mais de um elemento da saída provem de um mesmo elemento da entrada.

Para a fase de testes ficam reservadas tarefas como verificar se todos os nomes de elementos são únicos, se todos os elementos filhos do mesmo pai são do mesmo tipo, etc.

A fase de geração de diagramas de programas foi dividida em duas:

1. Geração da estrutura do diagrama de programa
2. Geração das condições de iteração e seleção

2.2. DETALHAMENTO DA OPERAÇÃO

2.2.1. DEFINIÇÃO DAS ENTRADAS E SAÍDAS.

A forma de representação proposta por Jackson e a formalização de Hughes (4) são satisfatórias para a organização do conhecimento.

Na primeira fase do projeto decidiu-se implementar apenas os passos necessários a construção do diagrama de programa sem gerar pseudo-código ou acrescentar ao diagrama de programa as funções necessárias a solução do problema, i.e., gera-se apenas a parte estrutural do diagrama de programa incluindo as condições referentes as iterações e seleções.

A representação escolhida foi a declarativa e pode ser mais facilmente explicada com os exemplos a seguir:

Os diagramas de entrada e saída são os mesmos utilizados por Jackson.
Para descrever estes diagramas tem-se duas declarações:

1. TIPO
2. FILHO,

onde TIPO (nome do nó, tipo do nó, diagrama ao qual pertence), fornece o nome de um nó, seu tipo, e o nome do diagrama ao qual o nó pertence. Por exemplo:

e FILHO (nome do nó, nome do pai do nó, ordem de filiação, diagrama ao qual pertence o nó), fornece para um dado nó seu nome, o nome de seu pai, a sua ordem como filho e o nome do diagrama ao qual pertence. Por exemplo para o diagrama acima têm-se:

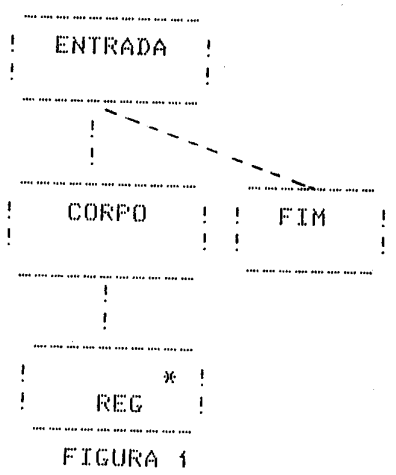


FIGURA 1

```
TIPO ( ENTRADA SEQ ENTRADA )
TIPO ( CORPO   SEQ ENTRADA )
TIPO ( FIM     SEQ ENTRADA )
TIPO ( REG     REP ENTRADA )
```

onde os tipos são:

SEQ para seqüência
SEL para seleção
REP para repetição

```
FILHO ( CORPO ENTRADA 1 ENTRADA )
FILHO ( FIM   ENTRADA 2 ENTRADA )
FILHO ( REG   CORPO   1 ENTRADA )
```

Observe que o nome do nó raiz de um diagrama serve também para denotar o nome do diagrama.
Além dessa declarações tem-se:

DIAGR-ENT (nome de diagrama)
DIAGR-SAI (nome de diagrama),

que caracterizam os diagramas de entrada e saida, respectivamente.

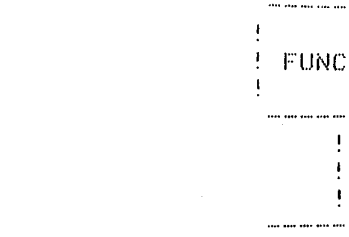
A especificação física dos registros de entrada faz-se necessária para que o sistema possa determinar quais são os campos de cada registro e quais são os valores válidos para os mesmos. Com esse conhecimento passa a ser possível testar e validar as restrições associadas as iterações e seleções nos diagramas uma vez que estas restrições fazem menção ao conteúdo dos registros.
A declaração

.FISICO (nome do nó, nome do campo, ordem do campo),

determina para cada folha (nó) o nome de um campo e sua posição de ocorrência no registro que está sendo caracterizado.
A declaração

CONDFIS (nome de nó, nome de condição),

determina para cada folha (nó), o nome da condição que define os valores válidos para a folha.
Por exemplo, para a um diagram semlhante ao anterior, onde o campo **FIM** é substituído pelo campo **FLAG** onde **REG** e **FLAG** são registros com os campos NUM e VALOR e o FLAG é tal que o valor do campo NUM é o maior inteiro armazenável (MAXIMO) têm-se então:




```
FUNCIONAL ( TOT-PAIS FUNC1)
FUNCIONAL ( TOTAL    FUNC2)

DIAGR-FUNC( FUNC1 )
DIAGR-FUNC( FUNC2 )

TIPO      ( FUNC1 SEQ FUNC1 )
TIPO      ( REG   REP FUNC1 )
TIPO      ( FUNC2 SEQ FUNC2 )
TIPO      ( REG   REP FUNC2 )

FILHO     ( REG FUNC1 1  FUNC1 )
FILHO     ( REG FUNC2 2  FUNC2 )
```

FIGURA 2

```
FISICO (REG NUM 1)
FISICO (REG VALOR 2)
FISICO (FLAG NUM 1)
FISICO (FLAG VALOR 2)
CONDFIS (REG C1)
CONDFIS (FLAG C2)
```

e as condições C1 e C2 são expressas por:

```
C1 (==) REG.NUM = MÁXIMO
C2 (==) FAG.NUM = MÁXIMO
```

Somente os nós terminais, folhas, tem especificação física e, o nome do nó folha também designa o registro que está sendo descrito.

A especificação funcional fornece a semântica do processamento, determinando que elementos, i. e., que estruturas ou partes ,dos argumentos de entrada devem ser processados a fim de que a saída especificada possa ser construída.

Por exemplo ,para diagramas de entrada e saída semelhantes ao diagrama anterior onde para o diagrama de saída o campo ENTRADA passa a ser chamada de SAÍDA, CORPO de CORPO-S, FLAG de TOTAL e REG de TOT-PAIS e a seguinte especificação funcional:

```
tot-pais (-- ( reg )*
total    (-- ( reg )*
```

deduz-se que tanto tot-pais quanto total são obtidos pelo processamento de um certo conjunto de registros "reg".

Para declarar a especificação funcional tem-se:

FUNCIONAL (nome de nó, nome de diagrama)

a qual associa a cada nó um diagrama tipo Jackson que exprime a especificação funcional deste nó

Na figura 2 têm-se a especificação funcional de "tot-pais" e "total" onde além de exemplificar-se a declaração FUNCIONAL, acrescenta-se as declarações complementares do diagrama da figura 1.

Se por acaso "tot-pais" e "total" forem obtidos da mesma massa de dados a declaração SIMULT (nome de nó, nome de nó), completa a especificação.No caso ter-se-ia: SIMULT (TOT-PAIS TOTAL) ou SIMULT (TOTAL TOT-PAIS)

Como apêndice ao banco de conhecimento permite-se ao usuário, quando da especificação funcional o uso de estruturas padrões que podem ser redefinidas para cada ambiente de trabalho. Como exemplo para o processamento de arquivos sequenciais sugere-se:

1. Arquivo

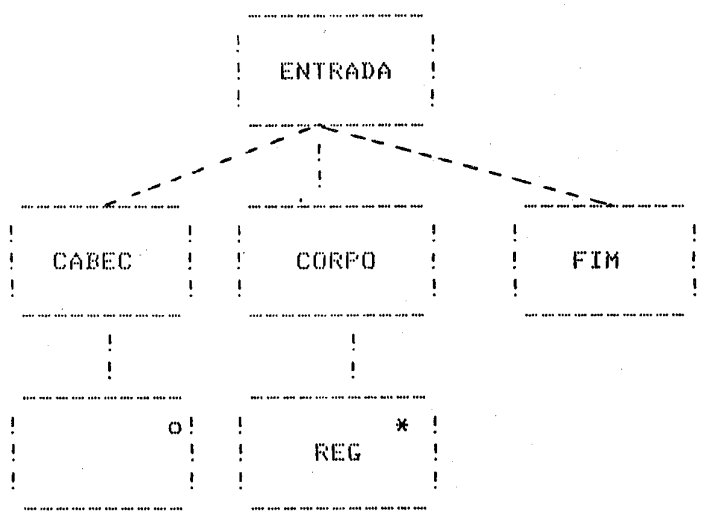


FIGURA 3

2. Contador (x) ,onde x satisfaz C1 que caracteriza o elemento sendo contado

a. 1 opção. O elemento a ser contado pertence a um grupo onde também se encontram elementos diferentes do desejado:

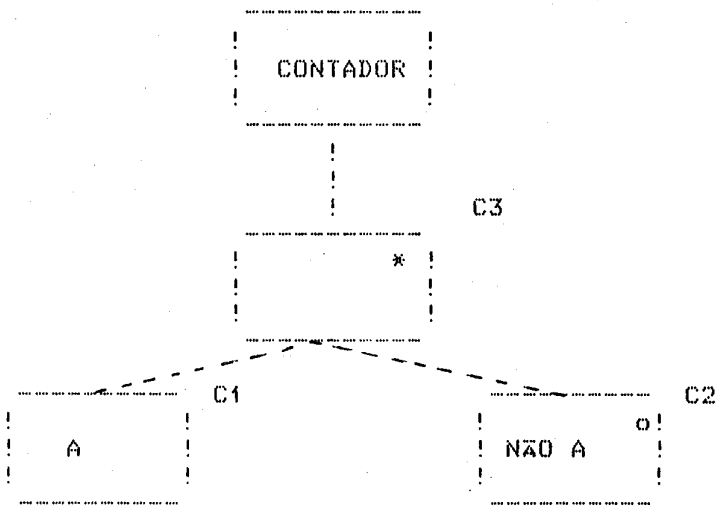


FIGURA 4

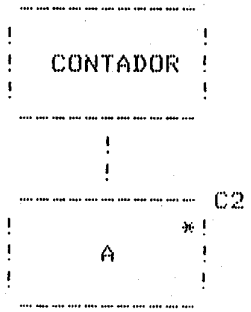
A condição C1, conforme já descrito, caracteriza o elemento a ser contado; C2 deve ser tal que:

C2 ==) ^ C1 e C3, por sua vez, deve ser tal que

C3 ==) ^ C1 e ^ c2

b. 2 opção. O elemento a ser contado pertence a um grupo onde todos são iguais:

FIGURA 5



onde C2 deve ser tal que

C2 ==) ^C1

2.2.2.TESTE DOS DIAGRAMAS FORNECIDOS.

Foram criadas regras tanto para a fase de testes quanto para a fase de geração do diagrama de programa. Esta parte do banco de conhecimento está preparada para adições incrementais que aumentem seu domínio de aplicabilidade.

Segue-se algumas das regras para os testes de validação da especificação:

1. verificar para cada diagrama se os filhos de um mesmo pai são do mesmo tipo.
2. verificar se em cada diagrama de entrada e em cada diagrama de saída, os nomes de nós são únicos a menos de irmãos do tipo seleção. Não é permitido tambem haver repetição de nome de nó entre diagramas de entrada nem entre diagrama de entrada e diagrama de saída.
3. verificar se todas as folhas dos diagramas de entrada têm especificação funcional.

2.2.3.GERAÇÃO DOS DIAGRAMAS DE PROGRAMAS.

Seguem-se algumas regras para a geração dos diagramas de programa:

1. Os diagramas de especificação funcional devem ser complementados, se for o caso, mediante as informações contidas no diagrama de entrada.
2. Para cada caixa do tipo repetição ("*") ou do tipo seleção ("o") dos diagramas funcionais obtidos, gera-se uma restrição. No caso de repetição a restrição deve informar a condição de parada e no caso de seleção, o que caracteriza a existência do elemento em questão.

   Essas restrições podem neste momento ter um carater genérico, sendo mais tarde definidas pelo usuário.
3. Para cada elemento da saída que possua especificação funcional, têm-se tres opções:
   a. Se o elemento não é simultâneo a nenhum outro o sistema copia, em seqüência, na nova entrada, a parte da subárvore de saída ao qual o elemento pertence. Esta estrutura começa no filho do nó saída e termina no elemento em questão, inclusive. O sistema também acrescenta à estrutura, a árvore de especificação funcional do elemento em questão. Se forem acrescentados elementos com "*" ou "o" o sistema gera para cada um destes elementos uma restrição de carater genérico.
   b. Se o elemento é simultâneo a algum outro, cuja especificação ainda não foi alocada na nova entrada o procedimento é idêntico ao anterior.
   c. Se o elemento é simultâneo a algum outro, cuja especificação já foi alocada a entrada sendo gerada, o sistema tenta acoplar a estrutura do elemento em questão com a estrutura do elemento dito simultâneo que já está na entrada sendo gerada. O acoplamento obedece a várias regras dentre as quais têm-se:
      1) àcoplar-se primeiro as caixas de nível mais alto.
      2) se houver mais de uma opção no mesmo nível, começar pelo elemento mais a esquerda até encontrar um acoplável.
      3) no caso de num dado nível da nova entrada, não existir um nó acoplável com o elemento em questão, passa-se a trabalhar um nível abaixo na nova entrada desde que haja uma regra que compatibilize as estruturas.
4. se não for possível casar as estruturas então ou falta informação ou existe informação incorreta. O sistema pede novas informações ao usuário ou que corrija as antigas.

Uma vez obtida a estrutura do novo diagrama de entrada, o sistema começa a definir as restrições que precisem ser associadas ao diagrama.

Para definir estas restrições faz-se necessário interagir com o usuário e então para cada restrição que não seja definida, isto é, que não seja uma equivalência de conjunções, o sistema pede ao usuário mais informações, sempre informando o estágio atual do conhecimento. Por exemplo, para restrições associadas à seleção ("o"), pergunta-se:

Qual a condição ---------- (número da restrição) ---------- que caracteriza a seleção do elemento ---------- (nome do nó) ---------- ?

O sistema sabe que:
segue lista de restrições associadas a este elemento

2.2.4.TESTES DOS DIAGRAMAS GERADOS.

São feitas verificações nas respostas dos usuários tais como:

1. a existência de campos em registros mencionados pelo usuário, levando-se em consideração que um nó pode herdar os campos de seus filhos, se estes são do tipo seleção.
2. não contradição com restrições já existentes.

Não existem respostas obrigatórias, mas esta fase tem como objetivo caracterizar da melhor forma possível o conjunto de restrições em análise.

Como próximo passo, o sistema passa a verificar, se o diagrama de entrada sintetizado é compatível com o(s) diagrama(s) de entrada fornecidos pelo usuário. Verifica-se, na verdade, se a estrutura fornecida pelo usuário, ainda, está presente no diagrama que acaba de ser sintetizado.

O teste consiste, basicamente, em verificar se o diagrama sintetizado está contido no(s) diagrama(s) fornecido(s) pelo usuário; sendo que no caso de existir mais de um arquivo só deve ser levado em consideração o conjunto de nós do diagrama sintetizado referente àquele arquivo.

Se neste teste descobrir-se elementos obrigatórios, da árvore original, faltando na árvore sintetizada, é perguntado ao usuário onde estes elementos devem ser colocados.

Uma vez completo o diagrama sintetizado, passa-se para o último teste, que consiste em verificar os sucessores lógicos obrigatórios: para cada elemento do tipo repetição verifica-se se seu sucessor obrigatório satisfaz a restrição que marca o fim da repetição.

Sucessor obrigatório de um nó é o que se segue , da esquerda para a direita, do tipo sequencia. Diz-se que o sucessor obrigatório satisfaz a restrição quando substituindo PROX, na restrição, pelo sucessor, ela é satisfeita. Quando a restrição for apenas parcialmente satisfeita, continua-se a buscar tantos sucessores obrigatórios quantos forem necessários, para a completa satisfação da mesma. Caso isto não aconteça, houve um erro na montagem da estrutura sintetizada.

Este teste precisa ser feito apenas para as repetições de nível mais alto de cada ramo, pois para cada estrutura como a da figura 6, gera-se uma restrição do tipo **C1 ===> C2** .Logo se o sucessor encontrado satisfaz **C1** também satisfaz **C2** .

A construção do diagrama de programa é incremental e as regras são na maioria independentes. Resumindo:

1. Regras para gerar a estrutura apropriada dos diagramas de entrada e saída
2. Regras para ajudar ao usuário na construção dos diagramas de especificação funcional.
3. Regras para testar a especificação do usuário.
4. Regras para gerar o diagrama de programa.
5. Regras para gerar as restrições das condições de iteração e seleção

3.ESTRATÉGIA DE INFERÊNCIA E INTERFACE COM O USUÁRIO

Para poder-se analisar rapidamente resultados e, validar a base de conhecimento, resolveu-se utilizar como estratégia de inferência a do PROLOG e implementou-se o sistema em

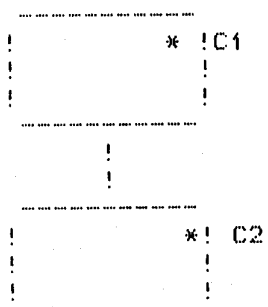


FIGURA 6

micro-PROLOG PROFESSIONAL para numa versão posterior do sistema fazer-se uso de mecanismos mais apropriados.

No estágio atual do protótipo o usuário é obrigado a escrever os fatos que caracterizam sua especificação e recebe como resposta fatos que caracterizam o diagrama de programa.

Numa etapa posterior pretende-se permitir ao usuário comunicar-se com o sistema atraves de diagramas.

4.EXEMPLO.

O departamento de estoque de uma fábrica recebe e envia peças. Cada envio ou recebimento é registrado num cartão. Cada cartão contém o número da peça, o tipo do movimento ("R" para recebimento e "E" para envio) e a quantidade. Os cartões estão em ordem crescente por número de peça; o arquivo de dados chama-se MOVIMENTO e termina quando o número da peça é Z9999. O programa a ser desenvolvido deve emitir um relatório com o resumo do movimento de cada peça. O formato do relatório deve ser o seguinte:

```
                RESUMO DO MOVIMENTO
A1736           movimento líquido      -450
A1932           movimento líquido       +35
  .                     .                 .
  .                     .                 .
Y4640           movimento líquido      +1845
FIM DO RESUMO
```

Nesse relatório o movimento líquido da peça A1736 foi de menos 450 unidades, i.é, no período coberto pelo relatório teve-se mais 450 peças enviadas do que recebidas. Cada uma das linhas do relatório tem um significado análogo.

SOLUÇÃO

Supõe-se os diagramas das figuras 7 e 8 sejam fornecidos pelo usuário com as especificações anexas:

Diagrama de saída

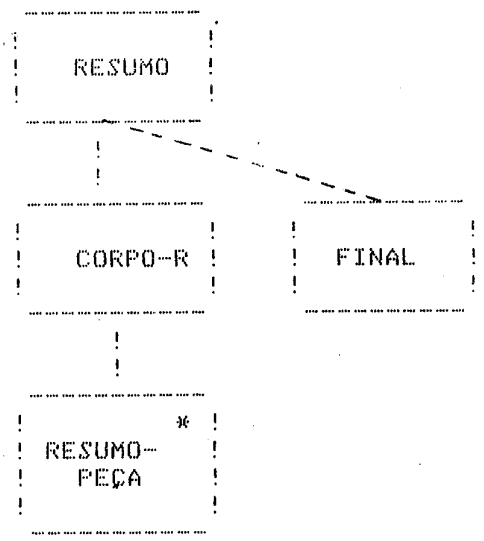


FIGURA 7

Diagrama de entrada

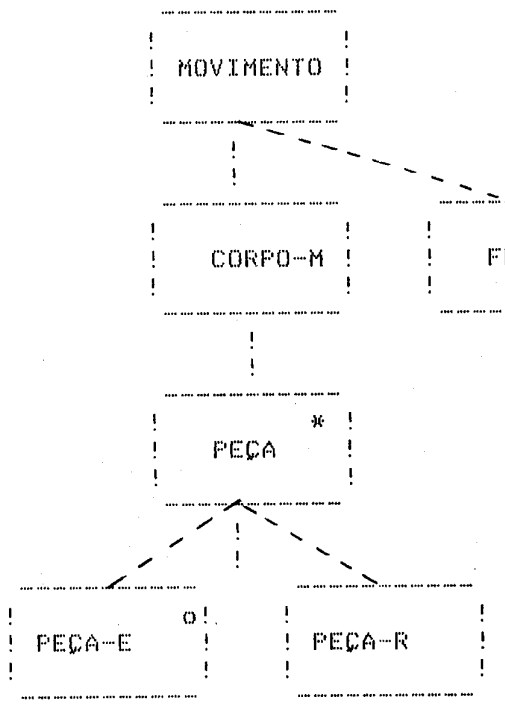


FIGURA 8

Especificação física.

PEÇA-E: (código, tipo, quantidade) onde PEÇA-E.código = Z9999
e PEÇA-E.tipo = "E"
PEÇA-R: (código, tipo, quantidade) onde PEÇA-R.código = Z9999
e PEÇA-E.tipo = "R"
FLAG: (código, tipo, quantidade) onde FLAG.código = Z9999

Especificação Funcional.

RESUMO-PEÇA <--- (PEÇA)*
FINAL <--- FLAG
A diagrama da figura 9 onde FUNC1 e FUNC2 são respectivamente, as árvores de especificação funcional de RESUMO-PEÇa e FINAL traduzem graficamente a especificação:

Os dados fornecidos pelo usuário passam pelos testes de validação, descritos anteriormente.

Estando os dados validados, começa a fase de geração do diagrama de programa, conforma explicitado anteriormente.

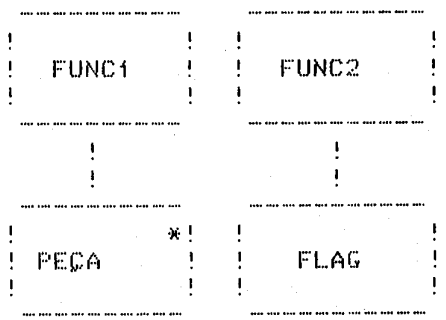


FIGURA 9

1. Nesse exemplo o diagrama FUNC1 é complementado, usando a informação contida em MOVIMENTO e acrescenta-se as condições C1, C2 e C3 obtendo-se o diagrama da figura 10.

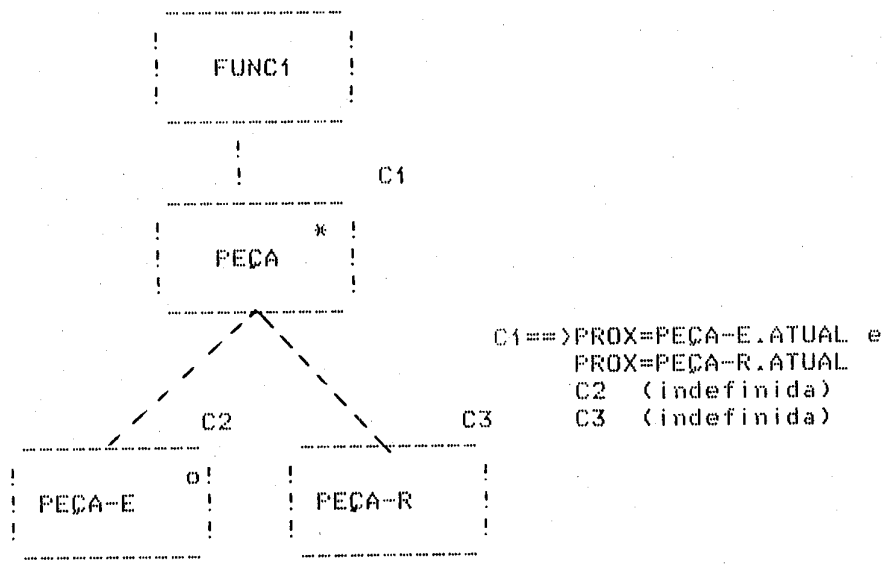


FIGURA 10

2. A partir desse ponto passa-se propriamente a gerar a estrutura de programa, que chamar-se-á NOVA-ENTRADA.

Seguindo os passos descritos anteriormente e do fato que RESUMO-PEÇA não é simultâneo tem-se a alocação de RESUMO-PEÇA em sequencia na NOVA-ENTRADA. Gera-se, então, a estrutura da figura 11 copiando-se o pedaço da subárvore ao qual RESUMO-PEÇA pertence.

Ao elemento, RESUMO-PEÇA, acrescenta-se sua especificação funcional, obtendo a estrutura da figura 12.

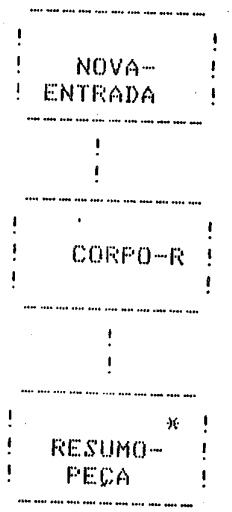


FIGURA 11

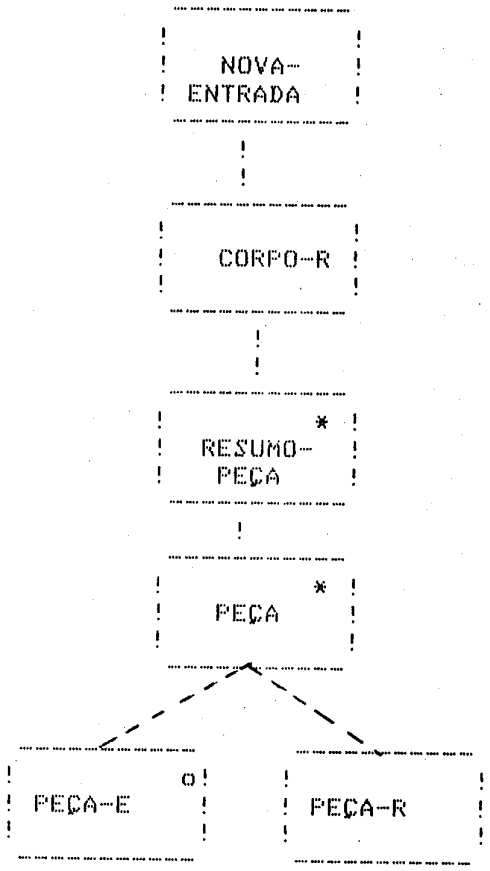


FIGURA 12

A seguir aloca-se o elemento FINAL, que também não é simultâneo e então é alocado em sequencia na NOVA-ENTRADA. Tem-se o diagrama da figura 13 copiando-se o pedaço da subárvore de saída ao qual FINAL pertence complementado a seguir com a especificação funcional.

A partir desse ponto, o sistema questiona o usuário a respeito das condições Supondo as seguintes respostas:

C1 <==> PROX.CODIGO = ATUAL.CODIGO

C2 <==> ATUAL.TIPO = "E"

C3 <==> ATUAL.TIPO = "R"

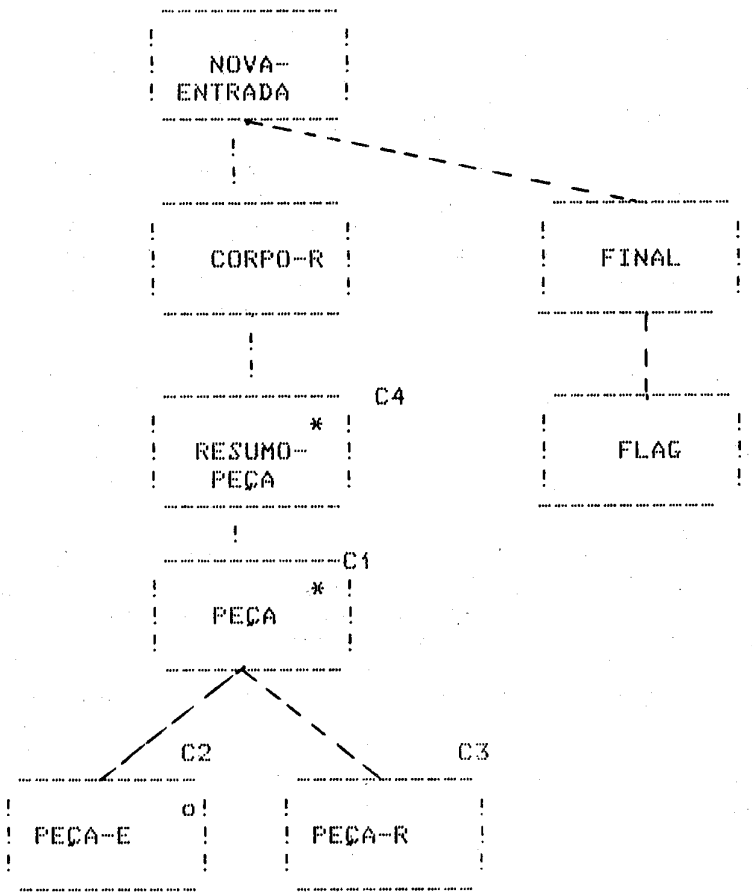


FIGURA 13

C4 ⟨==⟩ PROX.CODIGO = Z9999

O sistema então verifica se as respostas são válidas e se as restrições são satisfeitas, i.é:

PROX.CODIGO = ATUAL.CODIGO ==⟩ PROX=PEÇA-E.ATUAL e
PROX=PEÇA-R.ATUAL

PROX.CODIGO = Z9999 ==⟩ PROX.CODIGO=ATUAL.CODIGO

O primeiro item é verdadeiro pois o registro atual no ponto C1 só pode ser PEÇA-E ou PEÇA-R e se o código do próximo registro é diferente do atual, então ele é diferente do código de PEÇA-E e de PEÇA-R. Portanto o próximo registro, PROX, é diferente de PEÇA-E e PEÇA-R atuais.

O segundo item também é satisfeito uma vez que apenas o FLAG tem o campo código com o valor Z9999 e ele é único no arquivo.

Assim sendo passa-se as verificações finais:

Verificar se a entrada sintetizada está contida no original. No caso sim.

Verificar sucessores lógicos.

O sucessor obrigatório da repetição com condição C4, FLAG, satisfaz a mesma. De acordo com a especificação física o campo código de FLAG tem o valor Z9999, e assim

PROX.CODIGO = Z9999 é trivialmente satisfeita.

Portanto o diagrama anterior está correto.

5.CONCLUSÕES

Problemas como os de múltiplas quebras (lotes encaixantes), intercalação(linha balanceada) já foram resolvidos pelo protótipo. O protótipo carece de uma utilização maior que permita não só validar como aumentar sua base de conhecimento e necessita tambem de uma interface mais amigavel. Como resultado marcante tem-se a prova da viabilidade de construção de sistemas especialistas para construção de software a partir de uma metodologia bem conhecida.

6. BIBLIOGRAFIA.

1. Jackson,M.,Principles of Program Design, Academic-Press, London,1975
2. Yourdon,E., Constantine,L.L., Structured Design, Prentice-Hall,Inc, New Jersey, 1979
3. Warnier,J. D., Les Procedures de Traitement et Leurs Donées (LCP),Les Editions D'Organisation, Paris, 1979
4. Hughes,J.W.,A Formalization and Explanation of the Michael Jackson Method of Program Design, in Software Pratice and Experience vol 9, 1979
5. Coleman,D.,Hughes,J.,Powell,M.A.,A Method for the Sintax Directed Design of Múltiprograms, in IEEE Trans on S.E. Vol SE 7 No 2, 1981
6. Nilsson,N.J., Principles of Artificial Intelligence, Addison Wesley, 1979
7. Winston,P.H.,Artificial Intelligence, Addison-Wesley, 1979
8. Kowalski,R., Software Engineering and Artificial Intelligence and New Generation Computing, The SPL- Insight 1983/1984 Award Lecture, 1984
9. Hayes -Roth,F., Knowledge Based Expert System, in IEEE Computer vol. 17 n 10 oct 1984, 1984
10. Freiling,M.,et all, Starting a Knowledge Engineering Project: A Step-by-Step Approach, in the AI Magazine, Fall, 1985