ELETRICAL: UMA LINGUAGEM PARA PESQUISA EM PROGRAMAÇÃO CONCORRENTE

Felipe Afonso de Almeida
Instituto Tecnológico de Aeronáutica
Divisão de Processamento de Dados
São José dos Campos - SP

Sueli Mendes
Programa de Engª de Sistemas de Computação
Universidade Federal do Rio de Janeiro
COPPE/UFRJ - Rio de Janeiro - RJ

SUMÁRIO

Este trabalho apresenta a linguagem Electrical que se propõe ser um instrumento para investigação de problemas em programação concorrente, distribuída ou não. Para tanto, a ênfase é dada aos aspectos de definição, criação, destruição e escalonamento de processos, assim como a seus mecanismos de comunicação e sincronização.

ABSTRACT

Electrical the language presented here aims to be a tool for investigating problems related to concurrent programs, distributed or not. Thus, it emphasizes aspects of definition, creation, destruction and schedulling of process as well as its communication and synchronization mechanisms.

1. INTRODUÇÃO

A linguagem Eletrical (Almeida 1986) foi desenvolvida com a finalidade de se ter disponível uma ferramenta de programação que possibilite a pesquisa na área de linguagens concorrentes, o desenvolvimento de programas concorrentes e aplicações de tempo real, por parte de alunos de graduação e pós-graduação vinculados a cursos de computação.

O desenvolvimento da linguagem em si concentrou-se basicamente na pesquisa de mecanismos para modularização de programas, sincronização e comunicação entre processos, criação e gerenciamento de processos e acesso ao 'hardware'. Uma grande ênfase

foi dada no sentido de que os programas escritos em Eletrical possam refletir, de modo claro, as estruturas encontradas nas soluções de aplicações.

O projeto tem como ponto de partida o conceito de RECURSO e algumas das propriedades existentes nas sentenças de comunicação entre processos da linguagem SR (Andrews 1982). Deve-se considerar, também a influência desempenhada pelas pesquisas acerca do conceito de módulo (Wirth 1977) e dos mecanismos de gerenciamento de processos implementados nas linguagens Modula (Wirth 1977), Modula-2 (Wirth 1983) e Ada (Millitary 1983).

A fim de que o objetivo principal do projeto fosse atingido manteve-se a estrutura geral da linguagem Pascal com modificações nos aspectos referentes a vetores e verificação de tipos. Desta forma, é possível codificar programas em Eletrical inteiramente seqüenciais, com um mínimo de conhecimento a respeito dos modernos conceitos de programação (concorrente ou não) atualmente em pesquisa.

2. A LINGUAGEM ELETRICAL

No nível mais alto da linguagem encontra-se o RECURSO que representa módulos completamente funcionais, em que pode ser dividida a implementação de uma determinada aplicação. Ele pode ter um comportamento estático ou dinâmico.

O RECURSO tem declarações de importação e exportação de identificadores.

Ao exportar um identificador indica-se para quais RECURSOS ele será visível e de que forma ele pode ser usado por quem o importar. Do mesmo modo, um RECURSO ao importar um identificador deve indicar o nome do RECURSO em que tal identificador foi definido, podendo para algumas classes de identificadores indicar o que na realidade será importado. Tal flexibilidade aplica-se a procedimentos que podem ter exportado o seu código e um RECURSO tem, então, a opção de importar somente o identificador, seus parâmetros formais e tipos de dados.

No segundo nível encontram-se processos. Isto permite que se estruture um RECURSO pelo uso de vários processos e seu modelo de interação. Na realidade, processos devam ser usados quando os módulos funcionais que constituem um programa devem respon-

der a eventos assíncronos e implementem ações que ocorrem de modo simultâneo. Eventos podem então ser mapeados em processos (um para cada) de tal forma que cada evento seja síncrono com respeito ao processo que o manipula. Cada processo somente responde a eventos sincronizados com sua execução e o comportamento assíncrono do módulo completamente funcional é implementado pela execução assíncrona dos processos que o compõe. Similarmente, outras propriedades de processos, como prioridade de execução, privilégios diferentes e alocação de recursos, podem ser exploradas para modelar a estruturação encontrada durante a fase de projeto de uma aplicação para os módulos completamente funcionais (ex.: gerenciador de memória num Sistema Operacional).

Processos, em Eletrical, podem ser criados e destruídos dinamicamente. A linguagem permite, também, o escalonamento explícito de processos.

O terceiro nível implementa a nível de linguagem o conceito de tipo abstrato de dados (denominado em ELETRICAL de CLASS), oferecendo uma estrutura a parte e independente.

A última estrutura oferecida é o procedimento. Ela é bem adequada para a descrição de eventos abstratos (operações). Assim, no ponto onde o programador deseja invocar uma subrotina ele pode tratá-la como uma "caixa preta" que executa uma função abstrata específica, por meio de um algoritmo não conhecido. Deste modo, ao nível onde o procedimento é invocado, os detalhes relevantes de "o que" estão separados dos detalhes irrelevantes do "como". Ao nível onde é implementado, é desnecessário complicar o "como" com considerações sobre o "porque", isto é, as razões para invocar um procedimento freqüentemente não precisam ser levadas em consideração por seu implementador.

3. COMUNICAÇÃO E SINCRONIZAÇÃO ENTRE PROCESSOS

Em Eletrical os processos se comunicam via troca de mensagens explícitas ou usando o mecanismo de chamada remota de procedimento, depositadas em entidades denominadas PORTAS (PORTS) (Balzer 1971). Estas são declaradas na parte de interface de um RECURSO. A idéia é definir os canais de comunicação entre unidades completamente funcionais, além dos objetos visíveis, que se

rão compartilhados via memória, numa única unidade sintática, completamente definida, que forma a especificação do RECURSO.

PORT nome de porto (descrição das mensagens) PRIORITY limite.

Descrição das mensagens:

VAR
RES identificador: tipo
VAL

A decrição das mensagens especifica os tipos dos diversos componentes de uma mensagem e informa se eles são: valores (VAL), campos para devolução de resultados (RES) ou ambos (VAR).

A comunicação entre processos por compartilhamento de memória ocorreria o uso de variáveis globais do RECURSO que são compartilhadas por todos os processos nele declarados. O acesso concorrente é garantido, por implementação, ser feito de modo indivisível.

Para a comunicação via troca de mensagens, quatro sentenças são usadas. Duas para envio, denominadas CALL e SEND, e duas para recepção, denominadas MONITOR e de ENTRADA.

As sentenças CALL e SEND apresentam a seguinte sintaxe:

CALL nome de porto (mensagem) BY expressão

SEND nome de porto (mensagem) BY expressão

CO sentenças CALL OC

A sentença CALL bloqueia o processo emissor até que seja executada, por um outro processo, uma sentença de recepção de mensagem que retire e manipule da porta onde foi depositada a mensagem enviada. Para isto ambas sentenças devem referenciar a mesma porta. Na realidade esta sentença funciona como uma chamada remota de procedimento.

A sentença SEND funciona de modo semelhante, só que o processo emissor não fica bloqueado, ou seja, se constitue numa sentença de envio de mensagem não bloqueante.

A implementação do procedimento remoto é feito pelas sentenças MONITOR e de ENTRADA, que possuem três cláusulas que

especificam o seu comportamento. A primeira, denominada RESPONSE DO ou THEN DO faz com que ela se comporte como um procedimento remoto, ou seja, espera uma invocação, atende-a e manda de volta os resultados, caso existam. A segunda, denominada ELSE DO só é válida para a sentença de ENTRADA e é executada quando não existe uma mensagem no porto denotado, que satisfaça os critérios de seleção especificados. A terceira denominada PROG DO é executada quando não existe nenhuma mensagem no porto especificado. Desta forma as sentenças de recepção de mensagem podem funcionar estritamente como um procedimento remoto, ou como apenas um "receive" bloqueado ou não.

A sentença de ENTRADA apresenta a seguinte sintaxe:

IN nome de porto seleção cláusulas NI

INRECEIVE nome de porto seleção (lista de variáveis)
cláusula parcial NI

cláusulas -- THEN DO/RESPONSE DO: corpo END
cláusula parcial

cláusula parcial -- ELSE DO: corpo END
PROG DO: corpo END

corpo --- declarações
START
seqüência de sentenças

seleção --- AND (expressão lógica) BY expressão aritmética

A sentença MONITOR tem o formato:

ACCEPT nome de porto cláusulas ENDACCEPT

A sentença de ENTRADA permite que uma mensagem seja selecionada caso algum de seus valores tornem verdade a expressão lógica especificada, e dentre aquelas selecionar a que minimiza o valor da expressão aritmética especificada.

Uma das falhas apresentadas por procedimentos remotos, como implementados na maioria das linguagens que fornecem esta facilidade, é que a própria invocação a ser atendida e a primeira que foi depositada na fila de invocações ou uma outra, dependendo da política de colocação de invocações adotada quando

se executa uma chamada remota de procedimento. Isto nem sempre é o ideal, visto que muitas aplicações exigem um exame detalhado destas mensagens antes de decidir qual será a primeira a ser atendida (SIL 1981). A política de colocação de mensagens nas portas (ou mecanismos semelhantes) deve ser decidida pelo emissor que na maioria das vezes desconhece o estado em que se encontra o receptor, portanto políticas ótimas de colocação de mensagens não podem ser adotadas. A alternativa seria o emissor perguntar ao receptor se ele deseja a mensagem a ser enviada e enviá-la ou não conforme a resposta. Isto claro ocasionaria problemas de espera ocupada, e cada comunicação envolveria no mínimo o envio de duas mensagens, além de outras questões. Desta forma a sentença de ENTRADA fornece um mecanismo de, seletivamente, escolher qual mensagem manipular. Obviamente isto implica o exame de todas mensagens presentes na porta no instante de sua execução, o que às vezes pode tornar-se impraticável. Assim as sentenças de envio de mensagem podem determinar uma ordem em que estas serão colocadas na porta destino. Claro que isto não resolve de todo o problema como visto acima. Para isto usa-se a sentença de ESPERA que permite enfileirar mensagens (caso a avaliação da expressão lógica especificada resulte no valor verdadeiro) já retiradas da porta onde se encontram numa fila a parte (cada nome de condição determina uma destas filas) para posteriormente serem completamente manipuladas. Isto é possível, pois tais filas na realidade contêm registros de espera que além da mensagem, mantêm o ambiente volátil do processo no momento da interrupção do processamento de uma invocação remota.

A sentença SELECT possui a seguinte sintaxe:

sentença SELECT --- SELECT opção corpo ENDSELECT
opção --- parte de atribuição/FOREVER/MONITOR/vázio
corpo --- comando guardado comando guardado

OTHERWISE
comando guardado --- WHEN expressão lógica sentença guard
DO: seqüência de sentenças/

GUARD sentença guard ENDGUARD
seqüência de sentenças

sentença guard --- sentenças de comunicação
sentença de espera --- WAIT nome condição: WHEN
expressão lógica BY expressão
aritmética
sentença de sinalização --- SIGNAL nome de condição
TO expressão lógica.

O conceito de monitor de Hoare (1974), é implementado em Eletrical por uma sentença SELECT declarada usando-se a palavra reservada MONITOR. Isto define um escopo para os nomes de condições usados nas sentenças de ESPERA emitidas dentro de sentenças MONITOR. Uma invocação, suspensa numa fila de espera, será liberada para continuar a ser manipulada se for emitida numa sentença de SINALIZAÇÃO com o mesmo nome de condição (todas as invocações que satisfizerem a expressão lógica são liberadas). O uso da palavra chave FOREVER define uma sentença SELECT que será executada permanentemente. Enquanto que o uso da parte de atribuição permite que tal sentença seja executada até que a variável de controle seja maior que o limite, sendo que tal variável é visível no escopo da sentença. Isto possibilita que problemas que envolvam vetores de processos possam ser facilmente programados, usando-se a variável de controle como índice para obter a identidade desejada. Quando usada desta maneira tal sentença não define um novo escopo para os nomes de condição definidos em sentenças de ESPERA emitidas no seu interior. Desta forma uma sentença SELECT pode ser estruturada para se comportar como um monitor ou não, dependendo da aplicação a ser implementada.

A sintaxe é:

```
SELECT MONITOR
 WHEN ---------
 OR
 ACCEPT ---- THEN DO:
                 WAIT a: ----
 OR
 ACCEPT ---  THEN DO
               SIGNAL a:
 ENDSELECT
```

4. CONCLUSÕES

Neste trabalho procurou-se apresentar de forma sucinta as principais características da linguagem Eletrical, dando ênfase nos seus mecanismos para concorrência e facilidades para acesso ao 'hardware'. Vários aspectos da linguagem não foram tratados aqui por problema de espaço, mas podem ser visto com detalhes em Almeida (1986).

Pode-se considerar que os objetivos do projeto da linguagem são atingidos ao verificar-se que ela oferece as seguintes facilidades:

a) Modularidade

- O RECURSO permite que se modele módulos completamente funcionais oferecendo também uma parte para especificação de uma interface clara e independente para especificar as vias de comunicação e a visibilidade de objetos para outros RECURSOS.
- O tipo CLASS oferece uma estrutura à parte e independente para especificar tipos abstratos de dados, tornando possível determinar que atitudes devem ser tomadas ao entrar ou deixar o escopo onde objetos deste tipo foram declarados (parte inicial e final).
- Estrutura para especificação de unidades de execução concorrente.

b) Comunicação e Sincronização entre Processos

- Troca de mensagens usando procedimentos remotos e portas.
- Envio e recepção de mensagens através de chamada remota de procedimento, sentenças bloqueantes ou não.
- Designação do receptor ou emissor de mensagens de forma dinâmica.
- Liberação explícita de chamadas remotas.
- Possibilidade de adiar o atendimento de uma chamada remota já iniciada.

c) Gerenciamento de Processos

- Controle completo e explícito sobre o escalonamento para

acesso ao processador central.

- Controle explícito de criação e destruição.
- Alteração da prioridade em tempo de execução.
- Referências dinâmicas a processos.
- Execução de programas externos.

d) Acesso ao 'Hardware'

- Especificação do endereço de variáveis e código.
- Acesso a endereços de memória.
- Operações "bit" a "bit" e "byte" a "byte".
- Acesso a fila de mensagens de uma porta.
- Execução de código de máquina especificado em um programa.
- Definição de interrupção por "software".
- Vinculação entre o mecanismo de comunicação entre processos e a estrutura de interrupção da máquina alvo, possibilitando a programação dirigida por eventos.

e) Tipo de Dados e Vetores

- Linguagem com uso extensivo de tipo.
- Vetores dinâmicos.
- Operações sobre conjuntos de elementos de vetores.
- Definição dinâmica de cadeias de caracteres.
- Definição do tamanho de objetos apontados pelo tipo ponteiro de forma dinâmica.

f) Sentenças

- Sentença de execução não determinística, permitindo que nos guardas sejam especificadas tanto sentenças de envio quanto de recepção de mensagens, bem como apenas expressões lógicas.

Finalizando, deve-se ressaltar que não foram tratadas na linguagem as questões abaixo citadas, que deverão ser objeto de futuras pesquisas antes da proposição e incorporação de me

canismos próprios à linguagem.

- Compilação em separado;
- Temporização;
- Tratamento de exceções;
- Unidades genéricas;
- Ambiente de programação.

BIBLIOGRAFIA

ALMEIDA, F.A. - "Eletrical: Uma Linguagem para Programação Concorrente". Tese de Mestrado em Ciências. São José dos Campos, ITA, (1986).

ANDREWS, G.R. - "The Distributed Language SR, Mechanisms, Design and Implementation". Software Practice and Experience, 12: 719-753, (1982).

BALZER, R.M. - "Ports a Method for Dynamic Interprogram Communication on Job Control". Spring Join Computer Conference, (1971).

DIJKSTRA, E.W. - "Guarded Commands Nondeterminancy and Formal Derivations of Programs". Communication of ACM, 21(8):453-457, August, (1975).

HOARE, C.A. - "Monitors: An Operating Systems Concept". Communications ACM, 17(10):459-557, October, (1974).

HOARE, C.A. - "Communication Sequential Process. Communication of ACM, 21(8):66-677, (1978).

LISKOV, B.; SNYDER, A.; ATKINSON, R. & SCHAFFER, C. - "Abstractions Mechanisms in CLU". Communications of ACM, 20(8):564-576, August, (1977).

MILITARY STANDARD. "ADA Programming Language". ANIS/MIL-STD 18115A. American National Standards Institute, January, (1983).

SILBERSCHATZ, A. - "On the Sinchronization Mechanims of the ADA Language. Sigplan Notices, 16(2):96-103, February, (1981).

WIRTH, N. - "Modula: A Language for Modular Multiprogramming - Use of Modula - Design and Implementation of Modula". Software Practice and Experience, 7:3-84, (1977).

WIRTH, N. - "Programming in Modula 2". Springer Verlag, N.Y., (1983).