2
Circuitos de objetos numa casca de noz

Num circuito de objetos, o papel do objeto é análogo ao do *potencial* (Quevedo, 1988) num circuito elétrico. A propriedade de interesse do potencial elétrico é que, dado um conjunto de linhas de transmissão interligadas, ele é propagado uniformemente por todos os pontos do conjunto. Desta forma, se uma fonte, capaz de manter o potencial num certo nível constante, é conectada a um ponto *P*, então o potencial de *P*, bem como o de todos os pontos a ele conectados, é o potencial mantido pela fonte.

O objeto propaga-se por linhas de transmissão da mesma forma que o potencial elétrico. A analogia entre objeto e potencial é tão forte que não mais os distinguiremos no âmbito de circuitos de objetos. Sempre que necessário, faremos distinção entre o potencial de um circuito elétrico e o de um circuito de objetos. Dependendo do contexto, poderemos ainda nos referir ao objeto como *sinal*, *mensagem*, *dado*, etc.

Um valor de potencial em particular, denominado *potencial nulo*, denota a ausência de informação numa linha de transmissão, e é representado pela constante *N*. A presença do potencial nulo numa linha é equivalente ao estado de *alta impedância* numa linha de um circuito elétrico, caracterizado pela ausência de fontes ativas conectadas à linha.

A idéia de utilizar o potencial como único veículo de informação não é nova, uma vez que circuitos digitais, que constituem a base dos computadores modernos, o fazem para representar a unidade elementar de informação: o *bit*. Um bit pode assumir dois estados – "0" ou "1", "ligado" ou "desligado", etc. – sendo cada estado associado a uma certa faixa de valores de potencial elétrico. Nos primórdios da computação, a programação de software era feita no mesmo nível dos circuitos digitais do computador, realizando-se operações com bits. Neste sentido, a programação com circuitos de objetos é uma volta ao passado, mas com um enorme ganho em *expressividade* – ou seja, eficiência em atribuir significado

a um modelo – pois o emprego de objetos traz a modelagem para o mesmo nível de abstração do problema em mãos.

Um *nó* é um ponto do circuito onde é possível mensurar o potencial. Ele pode ser conectado por *trilhas* a outros nós, de tal forma a criar caminhos de propagação do potencial. A um dado nó *n* está associado um *ramo*: o grafo formado por todos os nós para os quais existe uma seqüência de arestas que os interliga a *n* (Ilustração 1). Num ramo, o potencial é propagado uniformemente por todos os nós, ou seja, vigora o princípio da *equipotencialidade*.

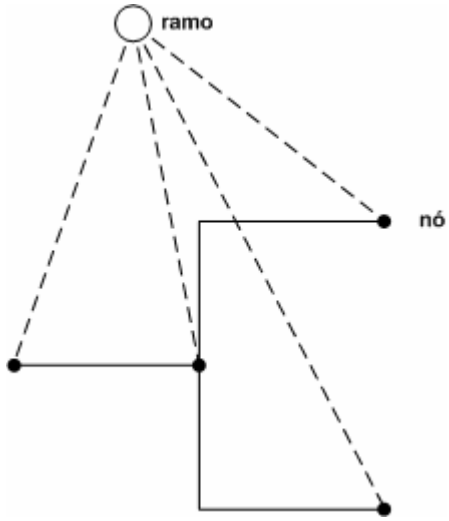

Ilustração 1 Nós interconectados e o respectivo ramo.

Todo o processamento da informação está concentrado nos *dispositivos*, similares aos parentes elétricos e eletrônicos. Um dispositivo é uma entidade com poder computacional com três características fundamentais: encapsulamento, autonomia e comunicabilidade.

Encapsulamento denota o fato de que um dispositivo é autocontido, isto é, possui fronteiras bem definidas. Isto significa que todo o ferramental necessário para que ele realize uma computação é interno ao próprio dispositivo. É diferente, por exemplo, de um objeto de uma linguagem OO tradicional, que pode depender de objetos externos. Embora um dispositivo nunca dependa de outros, externos, ele *pode* depender de dispositivos internos, em caso de composição, descrito mais adiante.

Consideremos agora um dispositivo como uma máquina rodando um processo único e interminável. Definimos então o conceito de *autonomia* como a capacidade da máquina – isto é, do dispositivo – de executar este processo

ininterrupta e indefinidamente, em concorrência com os processos das outras máquinas. Se porventura existir algum sincronismo na execução dos processos – e, em geral, ela é necessária e existe – isto se dá não por outro motivo que não a lógica do design do circuito.

Finalmente, *comunicabilidade* é a capacidade do dispositivo de interagir com o ambiente que o cerca. A comunicação é muito simples: um dispositivo "lê" sinais de entrada, processa-os e "escreve" os resultados na forma de sinais de saída. Por leitura, entenda-se a percepção do potencial num certo nó por algum sensor interno do dispositivo. Já por escrita, leia-se o fornecimento *contínuo* de potencial num dado nó, tal como uma fonte de tensão num circuito elétrico.

Leitura e escrita ocorrem através de um conjunto de *barramentos*, que constituem a interface do dispositivo com o exterior. Um barramento é formado por uma lista seqüencial de nós, aqui conhecidos como *portas* (Ilustração 2).Ao escrever num dado barramento *b*, o dispositivo associa um certo potencial a cada uma das portas de *b*. A partir daí, escritas subseqüentes realizadas pelo *mesmo* dispositivo apenas atualizam os potenciais das portas.

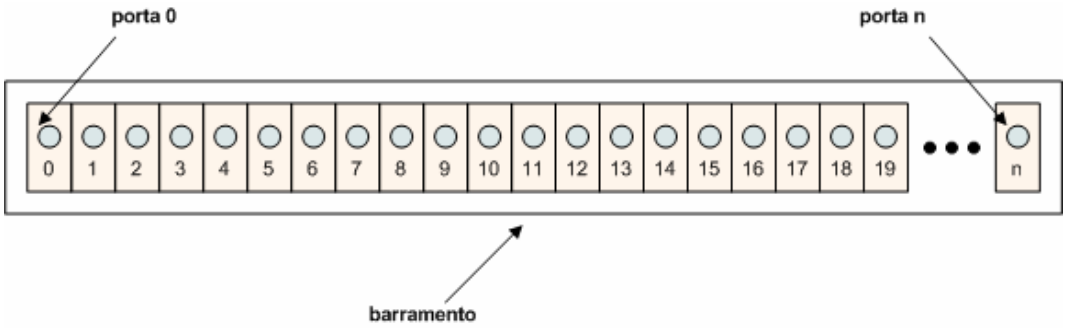

Ilustração 2 Barramento de comunicação.

Juntos, nós, dispositivos e barramentos constituem os *elementos* de um circuito de objetos. Através desta tríade, é possível construir qualquer circuito. A Ilustração 3 mostra o diagrama de um circuito com três dispositivos interconectados. Por simplicidade, os barramentos não são mostrados isoladamente, mas sim como parte integrante dos dispositivos. Nem sempre esta abordagem é adequada: em alguns casos, vários dispositivos podem compartilhar o mesmo barramento. Nestes casos, é necessária uma representação explícita do barramento compartilhado.

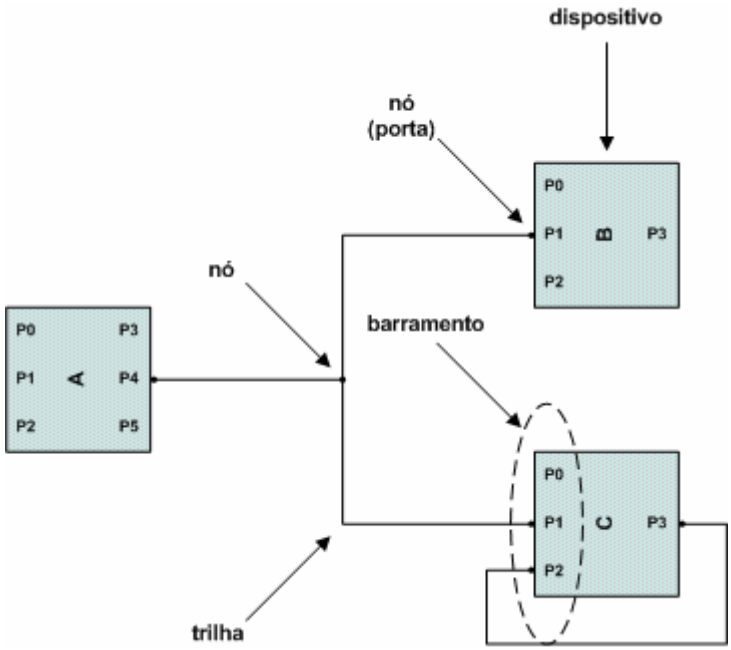

Ilustração 3 Um circuito simples e os elementos que o compõem: nós, barramentos e dispositivos.

A Ilustração 4 mostra o mesmo circuito em ação. A porta P4 do dispositivo A está ativa, sendo *f* o valor do potencial. Vemos então a propagação de *f* até as portas P1 de B e C.

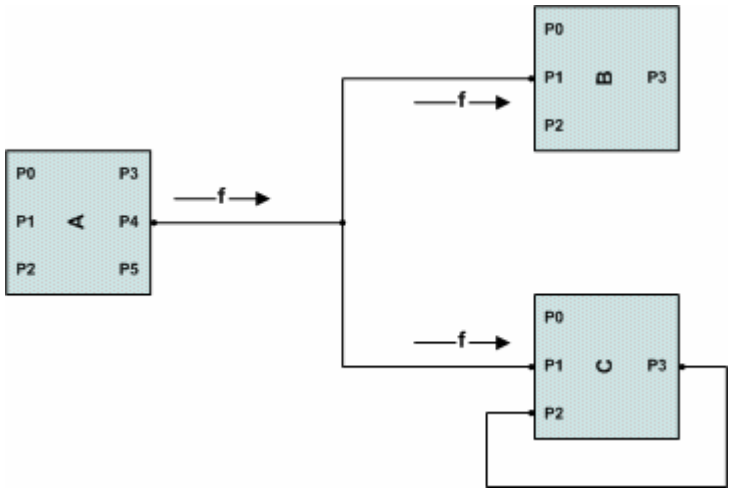

Ilustração 4 Propagação do potencial.

Ao escrever numa porta *p* um potencial não nulo, um dispositivo torna-se o *detentor* de *p*, que por sua vez torna-se uma porta *ativa*[1]. Mais ainda, o ramo de *p* também se torna ativo, sendo *p* denominada a *fonte* do ramo. Se, por outro lado, o

[1] Não confundir uma porta ativa com uma cujo potencial é não-nulo. Uma porta conectada a um ramo ativo possui potencial não-nulo, estando a porta ativa ou não.

detentor de *p* escreve *N* nesta porta, tanto *p* quanto seu ramo tornam-se *inativos*, e o dispositivo perde o status de detentor.

Decorre do princípio da equipotencialidade que um ramo tem no máximo uma fonte. A *tentativa* de violação deste princípio caracteriza a ocorrência de um *curto-circuito* (Ilustração 5). Tipicamente, um curto-circuito acontece quando um dispositivo tenta escrever em uma porta inativa de um ramo ativo, de tal forma que esta porta tornar-se-ia uma segunda fonte do ramo. É fundamental entender que, em caso de curto-circuito, a operação de escrita não é completada, de forma que o potencial original é mantido e equipotencialidade do ramo é preservada.

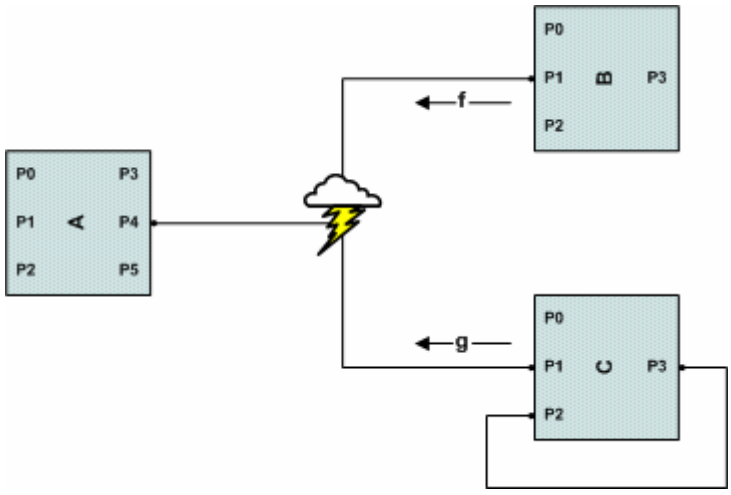

Ilustração 5 Dois dispositivos escrevem sobre portas pertencentes ao mesmo ramo, originando um curto-circuito.

2.1.Sincronização de leitura e escrita

Circuitos de objetos impõem algumas restrições sobre operações de leitura e escrita, cujo funcionamento é descrito a seguir.

Seja *d* um dispositivo, *b* um barramento de *d*, e sejam g_0, g_1,..., g_n todos os ramos das portas de *b*. Suponha que *d* inicie uma operação de leitura em *b*. Então, até que a leitura seja concluída, g_0, g_1,..., g_n estão *bloqueados para escrita*, isto é, nenhuma outra operação de escrita poderá ser iniciada caso envolva qualquer um dentre g_0, g_1,..., g_n. Outras operações de leitura, no entanto, são permitidas.

Suponha agora que *d* inicie uma operação de escrita em *b*. Então, até que a escrita seja concluída, g_0, g_1,..., g_n estão *bloqueados para leitura e escrita*, isto é,

nenhuma outra operação, seja de leitura ou de escrita, poderá ser iniciada caso envolva qualquer um dentre g_0, g_1,..., g_n.

Estas restrições permitem que dispositivos façam escritas atômicas de grupos de dados, ou seja, aquelas que ocorrem no escopo de uma *transação*. Isto evita que outros dispositivos realizem leituras "sujas" e escritas concorrentes, o que deixaria o sistema num estado inconsistente.

Uma questão interessante é analisar o que ocorre numa situação de "disputa", isto é, uma em que é preciso decidir qual dentre duas operações mutuamente exclusivas será efetuada em detrimento da outra. Uma situação ainda mais complexa de disputa pode envolver um conjunto de operações, algumas delas mutuamente exclusivas. A teoria de circuitos de objetos, entretanto, não entra no mérito da definição do algoritmo de disputa: diferentes implementações são livres para adotar a estratégia que lhes for conveniente.

2.2.O Legado de Von Neumann

Embora o avanço tecnológico tenha sido capaz de aumentar o poder dos processadores em algumas ordens de grandeza desde sua criação, na primeira metade do séc. XX, ele não foi – nem há expectativa de que eventualmente seja – capaz de produzir processadores velozes o bastante para computar uma vasta gama de problemas interessantes num horizonte de tempo razoável. Daí o motivo do estudo de concorrência sempre ter sido um tema popular entre pesquisadores. Se for possível modelar a solução de um problema – isto é, escrever um programa – de tal forma que diferentes partes do programa fiquem a cargo de diferentes processadores, reduzindo o esforço médio de cada processador a níveis satisfatórios, então a esperança de sucesso na resolução de problemas difíceis é renovada.

Dependendo do problema em mãos e do modelo de computação adotado, esta pode ser uma tarefa mais ou menos difícil. Nem todo problema é facilmente separável, e nem todo modelo se presta ao design de sistemas concorrentes.

De uma forma geral, e para a infelicidade dos programadores, há um consenso de que programas concorrentes são mais difíceis de construir que aqueles puramente seqüenciais (Alpern, 1994). Muitas vezes, a origem desta dificuldade está no próprio modelo adotado, e não na natureza do problema – o

que chega a ser surpreendente, pois temos opção de escolher o modelo de computação, mas raramente podemos escolher o problema resolver.

Esta questão tem raízes históricas. A maioria das linguagens de programação são versões de alto nível da máquina de Von Neumann, ainda hoje largamente adotada na construção de computadores. (Backus, 1978) faz uma extensa análise dos problemas decorrentes da adoção desta arquitetura e sugere alternativas; aqui, estamos apenas interessados em discutir os aspectos relativos à concorrência.

Na máquina proposta por Von Neumann, uma central de processamento, denominada CPU, comunica-se com uma memória por um canal. Através deste canal, a CPU envia *símbolos* para a memória, e obtém outros símbolos nela previamente armazenados. A computação é seqüencial por excelência, visto que o canal só permite a passagem de um símbolo por vez.

Uma linguagem de programação baseada no modelo de Von Neumann utiliza o conceito de *variáveis* para representar as células da memória, e *atribuições* de valores a variáveis como operações de armazenamento nas células, exatamente como a máquina original. É fácil ver que estas linguagens carregam a marca da seqüencialidade na execução das instruções, e o design de sistemas concorrentes fica claramente prejudicado.

Muito embora se admita a existência de mais de uma linha de execução compartilhando o mesmo código, qualquer uma das linhas é, em última análise, um mecanismo seqüencial de execução. Isto significa que a próxima instrução não poderá ser executada antes do término da atual, mesmo que o resultado da primeira não seja usado pela segunda, ou ainda, que a execução de uma delas não influencie, de algum modo, o resultado da outra. Esta abordagem tende a esconder a independência natural que muitas vezes ocorre entre partes de um algoritmo. Como ilustração, considere o seguinte trecho de código que calcula, iterativa e indefinidamente, o fatorial dos números inteiros.

```
1.      i <- 1
2.      f <- 1
3.
4.      while ( i <= x )
5.      {
6.         f  <- f * i
7.         i  <- i + 1
8.      }
```

As instruções (6) e (6) executam seqüencialmente; entretanto, uma rápida investigação mostra que o resultado de (6) não tem efeito, nem é utilizado por (7). Embora a execução de (7) – que altera o valor da variável `i` – certamente modifique o resultado de (6) – que faz uso deste valor – os lados direitos de ambas as instruções não tem correlação entre si, e poderiam ser executados em paralelo, resultando numa redução de tempo de até 50%.

O design de concorrência em paradigmas como o de circuitos de objetos é enormemente facilitado. A razão é que os elementos que compõem o programa estão espacialmente dispersos, o que resulta numa clara separação de blocos de código logicamente independentes. Embora sincronização seja possível, a norma é a execução concorrente. Esta abordagem possui uma elegância natural, pois induz o programador a pensar de forma concorrente, sem, no entanto, introduzir um nível desnecessário de complexidade.

Uma demonstração de como circuitos de objetos realizam esta separação de código, no caso do cálculo do fatorial, é dada no Apêndice.

2.3.Evolução dinâmica

Um dos principais objetivos do desenvolvimento de software orientado a componentes é o suporte à evolução dinâmica: a capacidade de alterar um sistema em execução através da adição ou remoção de componentes. Tipicamente, sua implementação acarreta um esforço considerável em paradigmas tradicionais. Não obstante, é fácil perceber que até os circuitos elétricos mais elementares fazem uso natural dela. Não fosse este o caso, seria necessário desligar a chave geral de energia de uma casa antes de trocar uma única lâmpada queimada. Esta "mágica" só é possível porque um circuito elétrico é governado por um conjunto rígido de leis – as leis da física – que garantem, continuamente, a integridade do sistema.

A força desta comparação nos leva a acreditar que as fundações sobre as quais a teoria de circuitos de objetos foi construída são ideais para o desenvolvimento de sistemas com capacidade de evolução dinâmica. Entretanto, encontrar um conjunto de leis de integridade apropriado é apenas parte do problema, pois estas são apenas descritivas, e não versam sobre *como* as modificações podem ser realizadas. De fato, é necessário criar um mecanismo que permita introduzir modificações no sistema, que, juntamente com um conjunto de leis de integridade, forma uma solução completa de evolução dinâmica.

Uma abordagem ingênua em relação à questão de como realizar modificações é deixar que usuários do sistema – sejam eles humanos, outros sistemas, etc. – cuidem do problema manualmente, de acordo com sua conveniência. Infelizmente, isto pode não ser suficiente em alguns domínios. É necessária uma solução mais genérica e robusta: uma que permita que as próprias entidades que compõem o circuito executem mudanças estruturais. Esta idéia não é nova: se pensarmos em circuitos biológicos – a rede de células que forma os seres vivos – concluiremos que a Natureza vem adotando-a por alguns milhões de anos.

Tais mecanismos auto-referentes (Hofstadter, 1999) são para circuitos o que introspecção é para programas OO: a capacidade de referenciar a si mesmo, o que é freqüentemente considerado como uma característica poderosa.

2.4.Composição

Dada a sua natureza de componente, um dispositivo comporta-se como uma caixa preta, capaz de receber e enviar dados. Do ponto de vista de quem o utiliza, o essencial é conhecer sua interface, sendo irrelevantes os mecanismos internos de implementação.

A abordagem da caixa-preta permite que vários dispositivos sejam agrupados, de forma a criar outros mais complexos. São os chamados *dispositivos compostos*. Um dispositivo composto é construído definindo-se uma nova interface – isto é, um novo conjunto de barramentos – para um circuito de objetos existente, de tal forma que o circuito passe a ser visto como um único dispositivo. Em contrapartida, os dispositivos não compostos são denominados *atômicos*.

Existe uma restrição quanto à composição: dois dispositivos compostos não podem ter interseção, isto é, os circuitos que os compõem não podem ter elementos em comum – sejam eles nós, barramentos ou outros dispositivos. Esta regra visa preservar a propriedade de encapsulamento.

O processo de composição é recursivo, o que permite a existência de *árvores de dispositivos*. No nível das folhas, residem os dispositivos atômicos, também conhecidos como *circuitos integrados de objetos*, ou CIOs. A razão do nome é que, tal como ocorre com os circuitos integrados da eletrônica, não é possível "abrir" um CIO e encontrar partes reutilizáveis no seu interior. Tipicamente, um CIO é construído sobre uma camada de abstração mais baixa, do mesmo modo que código de montagem é utilizado para escrever as funções básicas de uma linguagem de alto nível.

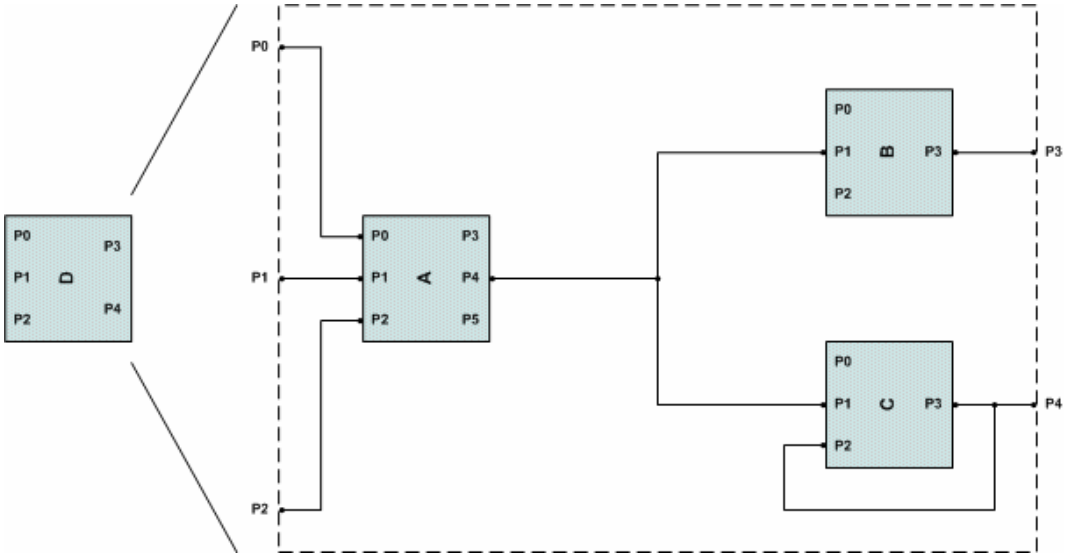

Ilustração 6 Visão interna de um dispositivo composto.