3
Circuitos de objetos e simulação

Simulação (Ross, 2001) é a arte da imitação. Freqüentemente, torna-se muito caro, ou mesmo impraticável, estudar um sistema real em ação. Se adicionarmos os pré-requisitos de que o sistema deve estar sob total controle e que o experimento possa ser repetido indefinidamente, a tarefa torna-se ainda mais difícil. Por este motivo, pesquisadores das mais diversas áreas encontram abrigo em técnicas de simulação, que provê maneiras de capturar, dentro de um modelo executável, todas as particularidades relevantes do sistema em estudo.

Sob uma perspectiva histórica, orientação a objetos e simulação tem muito em comum, pois alguns conceitos-chave da primeira – incluindo classes e objetos, herança e *dynamic binding* – foram introduzidas por uma linguagem de simulação dos anos 60, Simula I (Birtwistle, 1979; Kirkerud, 1989). De fato, os bons resultados obtidos no campo da simulação fizeram com que Simula fosse reescrita mais tarde sob o nome de Simula 67: uma linguagem de programação completa e genérica, que influenciou muitas linguagens OO modernas.

Do outro lado, a teoria de circuitos também está intimamente ligada à simulação, uma vez que ambas tratam de entidades concorrentes capazes de se comunicar num ambiente dinâmico. Deste modo, a união de orientação a objetos e circuitos é perfeita: a primeira, como uma ferramenta poderosa e expressiva de modelagem; a segunda, como uma plataforma onde sistemas concorrentes têm sua descrição mais natural e elegante.

3.1.Modos de tempo

Uma simulação *time-stepped* (TS) é tal que o estado do sistema é atualizado num ritmo constante. Pode-se imaginar a existência de um relógio global cuja tarefa é periodicamente sinalizar a passagem de uma unidade de tempo. Já uma simulação *discrete-event* (DE) mantém o estado do sistema inalterado até que algum evento externo cause uma transição e leve o sistema a um novo estado.

Ambos os tipos possuem prós e contras. TS é geralmente mais simples e barata de implementar; entretanto, se o estado do sistema tem grandes chances de permanecer estável por longos períodos, a atualização freqüente torna-se dispendiosa. Nestes casos, uma simulação do tipo DE é mais adequada. Todavia, mecanismos eficientes de tratamento de eventos são mais difíceis de implementar.

Muitas das ferramentas específicas de simulação disponíveis estão atreladas a um modo particular de passagem do tempo. A generalidade de circuitos de objetos, entretanto, permite que sejam construídos modelos que integram TS e DE. Isto é possível porque não existe uma distinção global entre modos de passagem de tempo; pelo contrário, ela é emulada pelos próprios dispositivos que compõem o circuito. Para compreender como isto é possível, explicaremos a implementação de DE usando circuitos de objetos, pois em nossa abordagem, simulação TS é um subcaso de DE onde o avanço do tempo é também representado por um evento discreto.

Um evento é definido como uma alteração de potencial em uma ou mais portas de um barramento. Subdividem-se em três categorias distintas: queda, subida e mudança de potencial. Um dispositivo conectado a um barramento é capaz de detectar eventos, e reagir de acordo.

O diagrama do circuito da Ilustração 1 mostra dois dispositivos: F, representando uma função genérica Y0 = F(X0, X1, X2), e um relógio. À medida que o relógio gera novos eventos, eles são percebidos através da porta CK, determinando que F reavalie suas entradas e atualize a porta de saída, Y0. Isto ilustra como o modo TS é alcançado. Note, porém, que embora uma linha global de passagem do tempo possa existir, este não é sempre o caso, pois cada dispositivo de relógio existente define sua própria linha do tempo local e independente.

A Ilustração 2 mostra como implementar o modo DE utilizando uma versão modificada do dispositivo F, que é guiada por mudanças de potencial nas portas de entrada. Este novo dispositivo é implementado substituindo o relógio da versão original de F por um disparador ou *trigger*, cuja função é gerar um sinal na porta Y0 toda vez que uma mudança ocorrer em X0, X1 ou X2. O evento é então sentido na porta CK, e F refaz sua computação.

Finalmente, a Ilustração 3 mostra como combinar ambos os modos de passagem do tempo. Nós modificamos o circuito anterior para incluir também o

relógio do primeiro circuito. O resultado é que F atualiza seu estado tanto em decorrência de um sinal de relógio disparado, quanto por alteração no potencial de uma de suas postas de entrada.

Ilustração 1 Time-stepped.

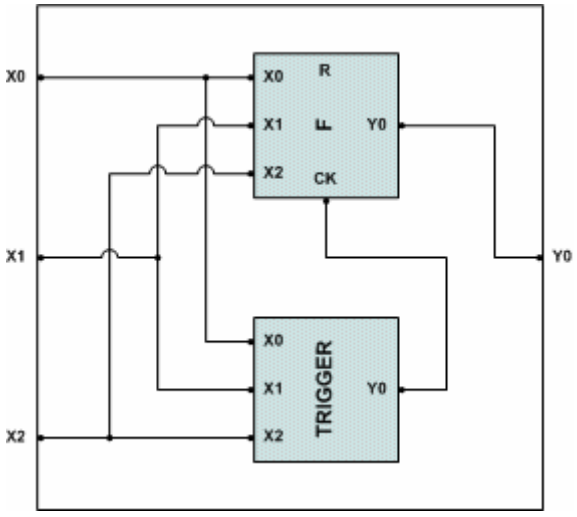

Ilustração 2 Discreve-event.

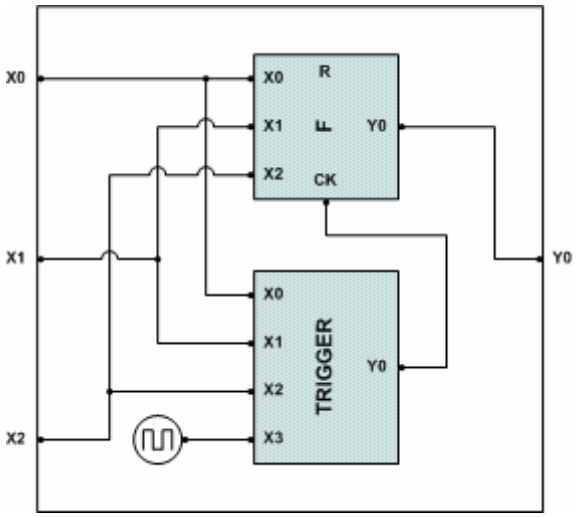

Ilustração 3 Modelo híbrido TS-DE.

3.2.Modelos de computação

Um modelo de computação (Savage, 1998) é uma representação abstrata de um computador. Toda linguagem de programação é baseada num modelo de computação. Com o modelo apropriado em mãos, é possível especificar um algoritmo que ressalta as partes de interesse de um problema, escondendo uma série de detalhes de implementação que tornariam seu estudo muito mais complexo. Alguns dos mais populares modelos de computação incluem os seguintes: seqüencial, funcional, relacional, concorrente e distribuído. Entretanto, é a natureza do problema que ditará qual modelo propicia a representação mais elegante.

Ferramentas de simulação não estão apenas interessadas em implementar um dado modelo de computação: problemas complexos podem ser freqüentemente subdivididos em problemas menores, que por sua vez são melhores representados em diferentes modelos. Então levantamos uma questão primordial: como simular sistemas heterogêneos, onde diversos modelos de computação são empregados em conjunto?

Uma possível resposta, tal como descrita em (Bhattacharyya, 2002), é a implementação explícita de uma série de modelos, além de mecanismos para integrá-los numa única simulação. Circuitos de objetos tomam um caminho diverso, uma vez que implementam um modelo único, cujo objetivo é ser genérico o bastante de forma que possa ser derivada, com relativa facilidade, uma gama de modelos mais simples. Esta não é uma solução perfeita, pois alguns modelos não podem ser derivados facilmente. Ainda assim, esta abordagem tem a vantagem de incorporar, sem custos adicionais, a integração de sistemas heterogêneos.

3.2.1.Dataflows

Um *dataflow* é um modelo de computação gráfico assíncrono e distribuído (Cheng, 1997). Existem diversas variações do tema; entretanto, aqui estaremos falando apenas do caso mais comum. Neste modelo, temos um grafo direcionado e acíclico, no qual vértices realizam computações baseadas nos dados provenientes das arestas de entrada. A computação num vértice é realizada tão logo todos os dados necessários para que ocorra estejam disponíveis. Finalizada a

computação, o resultado segue pelas arestas de saída do vértice, servindo de entrada para outros vértices do grafo.

A implementação de dataflows com circuitos de objetos é feita com o auxílio de dispositivos de sincronização (SYNCs), como mostrado na Ilustração 4. Tais dispositivos possuem dois barramentos – um de entrada, outro de saída – e operam de uma forma tal que, sempre que todas as portas do barramento de entrada tiverem potenciais não nulos, eles são espelhados no barramento de saída. Caso uma das portas de entrada sofra uma queda de potencial, o espelhamento cessa, e o dispositivo escreve o potencial nulo *N* em todas as portas do barramento de saída.

Como pode ser visto no circuito da esquerda, o dispositivo SYNC é ele próprio implementado como um circuito de objetos, composto de um detector de potencial, um chaveador, e um dispositivo que implementa a função lógica de conjunção, denominado AND. O detector escreve valores booleanos T e F, especificando se a respectiva porta de entrada está ativa ou não. O dispositivo AND, por sua vez, utiliza estes sinais para descobrir se todas as suas portas de entrada estão simultaneamente ativas. Ele também controla as portas de seleção do chaveador, representadas por Si. Já o chaveador trabalha da seguinte forma: sempre que Si possui um valor T, ele escreve em Yi o mesmo que Xi. Se, por outro lado, Si tem o valor F, então Yi permanece inativa.

O dispositivo SYNC comporta-se como uma "barreira" – neste exemplo, uma barreira associada à função lógica "E". Entretanto, uma vez que foi construído com base em outros dispositivos, poderia ser facilmente modificado para comportar-se como uma barreira do tipo "OU", ou mesmo uma barreira com lógica mais complexa.

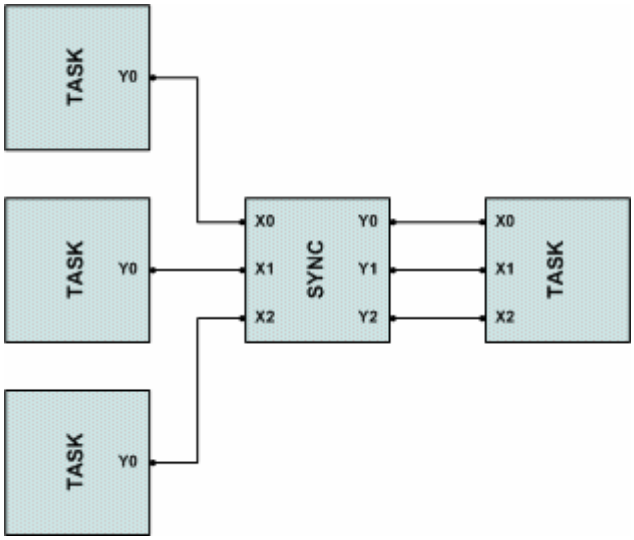

Ilustração 4 Modelando um dataflow. SYNC permite a propagação do potencial das portas de entrada para as de saída, desde que todas as portas de entrada tenham potencial não nulo.

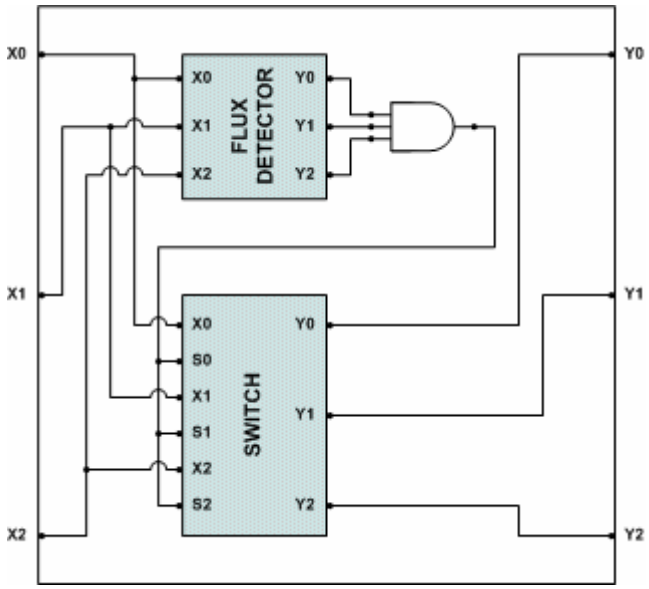

Ilustração 5 SYNC como um dispositivo composto.