A abordagem R-RIO para concepção de aplicações distribuídas

Alexandre Sztajnberg

IME-UERJ/COPPE-UFRJ
Rua São Francisco Xavier, 524 – 6º andar
22550-000, Rio de Janeiro RJ
alexszt@ime.uerj.br

Marcelo Lobosco ***Orlando Loques***

Computação Aplicada e Automação – CAA
Universidade Federal Fluminense – UFF
Rua Passo da Pátria, 156 - 24210-240, Niterói, RJ
{lobosco, loques}@caa.uff.br

1 Introdução

Aplicações distribuídas possuem natureza dinâmica e necessitam de suporte para se adaptarem a novas demandas funcionais e não-funcionais. Novas demandas funcionais são criadas por usuários das aplicações, que precisam de novas funções ou necessitam alterar funções existentes. Além disso, podem surgir novos requisitos não-funcionais que, de uma forma geral, não estão associados diretamente com a aplicação, como protocolos de comunicação, qualidade de serviço, disponibilidade ou tolerância a falhas, por exemplo. Modelos tradicionais para a concepção de *software* não oferecem as facilidades necessárias para a produção de aplicações que se adaptem dinamicamente a novos requisitos. Na maioria dos casos as aplicações são estáticas e fruto de um grande esforço de programação, resultando em um código complicado, com vários aspectos entrelaçados (*code tangling*). É comum, por exemplo, um mesmo bloco de código conter instruções para a comunicação via *sockets*, sincronização via semáforos, além das instruções que executam a computação intrínseca da aplicação, criando assim dificuldades para a reutilização de código. Para facilitar a concepção dessa classe de aplicações, uma nova abordagem se faz necessária. Esta abordagem deve considerar que todas aplicações precisam evoluir dinamicamente em seus aspectos funcionais e não-funcionais ou mesmo em sua estrutura.

2 Abordagem R-RIO

A proposta de R-RIO (*Reflective, Reconfigurable Interconnectable Objects*) é oferecer uma nova alternativa para a construção de aplicações distribuídas baseada em configuração. Esta proposta difere de outras existentes por integrar explicitamente o paradigma de separação de interesses. Em R-RIO o conceito de configuração está baseado em: (**a**) uma metodologia para a concepção de aplicações, por composição, a partir de módulos componentes, atendendo a aspectos funcionais básicos das mesmas; (**b**) uma linguagem de configuração (LC), que permite especificar a interconexão de módulos, que define a arquitetura básica das aplicações, e descrever características ou aspectos não-funcionais das mesmas, como por exemplo, sincronização, distribuição, tolerância a falhas ou parâmetros de qualidade de serviço; e (**c**) um sistema de suporte, chamado de configurador, que permite a criação, conexão e remoção de componentes das aplicações.

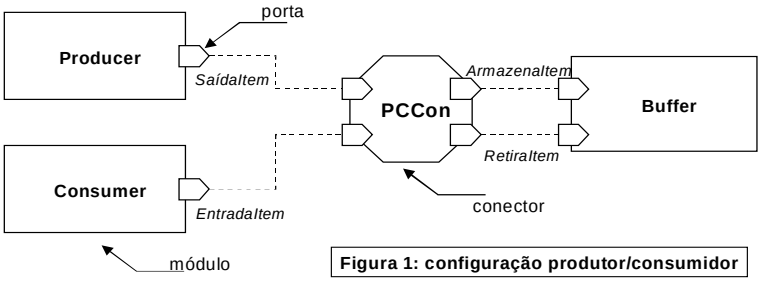

Figura 1: configuração produtor/consumidor

A metodologia de R-RIO incentiva a modularidade na concepção dos elementos básicos e a separação explicita entre a criação de componentes e as interações destes componentes. Isto permite, em princípio, retirar dos módulos a responsabilidade de implementar e gerenciar as interações, e concentrar esta responsabilidade em elementos de conexão, chamados em R-RIO de conectores. Esta separação de aspectos aumenta a possibilidade de reuso dos módulos funcionais e facilita a adequação dos conectores para diferentes estilos de interação. O modelo de R-RIO está baseado em três elementos básicos: (**1) módulos**, componentes de uma aplicação que, basicamente, encapsulam sua funcionalidade; (**2) conectores**, usados, no nível de configuração, para interligar e selecionar a forma de interação de módulos. No nível de operação os conectores intermediam e executam de forma

reflexiva a interação entre os módulos de acordo com o que foi configurado. Por exemplo, conectores podem encapsular padrões de interação entre módulos no mesmo sentido que é feito em *design patterns*; (**3) portas**, que identificam os pontos de acesso pelos quais módulos e conectores oferecem ou solicitam serviços.

Inicia-se a construção de uma aplicação descrevendo-se os módulos que irão compor a mesma e selecionando-se os conectores que interligarão estes módulos. Em seguida, o relacionamento destes elementos é determinado através das ligações entre módulos e conectores. Adicionalmente, podem ser descritos requisitos não-funcionais da aplicação, permitindo que esta composição de módulos e conectores seja ajustada em diferentes aspectos. O exemplo de uma configuração produtor-consumidor com *buffer* limitado é apresentado na Figura 1. O trecho de código descreve, com CL, partes da arquitetura do exemplo, incluindo o aspecto de sincronização e distribuição. Outros aspectos como comunicação podem ser descritos de forma semelhante.

```
module BufferApplication  {
  module BufferT {
    inport PutT ArmazenaItem;
    inport GetT RetiraItem;
    map CLASS Java "buffer"
  } Buffer;
  module  ProducerT { ... } Producer;
  module  ConsumerT { ... } Consumer;
  instantiate  Buffer    at estação1
  instantiate  Producer  at estação2;
  instantiate  Consumer  at estação3;
  instantiate  PCCon;
  link Producer, Consumer to Buffer by PCCon;
}
```

```
connector Cguard{
  condition vazio = true, cheio = false;
  statereq int n_itens;  int MAX_ITENS = 10;
  inport GetT;  inport PutT;
  exclusive {
    outport GetT { ... }
    outport PutT {
      inguard  (vazio) {
        after { if (n_itens == MAX_ITENS)
                  {vazio = false; cheio = true;}}
  } } }
  map CLASS Java "GuardGCon";
} PCCon;
```

```
module BufferAplication example;   instantiate example;   start example;
```

Os módulos são declarados indicando-se instâncias de portas e o mapeamento para um implementação (neste caso o módulo *Buffer* é implementado por *buffer.class*, programado em Java). Em seguida os conectores que interligarão os módulos são definidos. O aspecto de sincronização é configurado no conector *Cguard*. As portas de saída do conector são declaradas como mutuamente exclusivas e monitoradas por guardas. Uma requisição não chegará ao método *ArmazenaItem* do módulo *Buffer* se o guarda ***vazio*** for falso. Ao completar o método e antes de enviar a confirmação, o guarda ***vazio*** recebe o valor false se *n_itens* for igual a *MAX_ITENS*. Neste ponto os módulos e conectores são instanciados. O aspecto de distribuição é programado indicando-se em que estações os módulos serão instanciados. A configuração é concluída interligando-se os módulos através do conector. Em seguida cria-se uma instância da aplicação e inicia-se sua execução.

3 Conclusão

O modelo de R-RIO está sendo aperfeiçoado em vários pontos. A possibilidade da inclusão de novos aspectos será experimentada. A capacidade de composição e reflexão do modelo serão exploradas visando ampliar sua utilidade e abrangência em domínios diferentes de aplicações distribuídas e não distribuídas. As características que destacam R-RIO são: (a) flexibilidade e modularidade permitindo a rápida prototipação de aplicações. (b) separação de interesses é obtida com o projeto separado de aspectos diferentes da aplicação, integrados através da linguagem de configuração. (c) a evolução dinâmica é obtida pela capacidade de reconfiguração da aplicação durante sua operação e pela característica reflexiva dos conectores que permitem ajustes dinâmicos em aspectos não-funcionais. Um protótipo de um ambiente para programação distribuída em Java, baseado nos conceitos de R-RIO é descrito em [Lob 99]. Neste texto é também introduzido o conceito de *conector genérico*, que permite que conectores sejam adaptados automaticamente a diferentes interfaces de módulos em tempo de instanciação.

4 Referências Bibliográficas

[Lob 99] Lobosco, M., "R-RIO: Um Ambiente de Suporte à Construção e Evolução de Sistemas Distribuídos", dissertação de mestrado, CAA/UFF, Março, 1999.

[Szt 99] Sztajnberg, A. “Flexibilidade e Separação de Interesses em Sistemas Distribuídos”, proposta de tese de doutorado, em preparação, COPPE/UFRJ, 1999.