Projeto Baseado em Colaborações

João Carlos Vicente

Jaison Werlich

Kelton Rodrigo Zacchi

Projeto Baseado em Colaborações

- Metodologia que decompõe aplicações orientadas a objetos em conjunto de classes e conjunto de colaborações.

Colaborações

- Expressam aspectos da aplicação envolvendo diversos participantes (papéis);
- Uma classe pode participar de diferentes colaborações/ter diferentes papéis em uma colaboração;
- Cada papel representa um aspecto singular do comportamento da classe.

Projeto Baseado em Colaborações

- Representa aplicações orientadas a objeto de duas maneiras:
  - Em termos de participantes (classes) envolvidas;
  - Em termos de tarefas a serem executadas.
- Resulta em projetos mais compreensivos e reusáveis.

Problema Principal

- Linguagens de programação orientadas a objeto não tem um mapeamento direto com o projeto baseado em colaborações.
  - fluxo de controle do projeto fica espalhado por diversos pequenos métodos na hierarquia de classes;
  - “é como ler um mapa através de um canudinho de refrigerante”.

Solução

- Um componente que:
  - explicitamente capture um pedaço do comportamento do sistema (uma tarefa específica) que afete muitas classes, e que possa ser reusado;
  - Seja complementar à tecnologia de orientação a objetos existente, e não substituindo-a;
  - Suporte uma granularidade média(objetos<x<packages).

Sugestões de Componentes

- Frameworks;
- Adaptive plug-and-play components (APPC’s).

Frameworks

- Permitem reuso de interrelacionamentos entre classes (granularidade maior que a de objetos);
- Dependentes da linguagem de programação;
- Não expressam claramente as colaborações.

APPC’s

- Adaptive:
  - Expressam colaborações em uma família de aplicações (adaptive programming).
- Plug-and-Play:
  - Suportam composições black/white-box:
    - Podem ser refinados em outros componentes (tal como em herança);
    - Podem ser combinados para formar outros componentes;
    - Mantém o acoplamento baixo (facilitando o seu reuso).

Características

- Especificação genérica:
  - Permite à colaboração ser reusada na familia de aplicações;
  - Permite à mesma colaboração ser usada em muitos lugares em uma mesma aplicação, com participantes diferentes.
- Composição comportamental independente:
  - Suporte a reuso de componentes para colaborações mais complexas;
  - Acoplamento baixo;
  - Encapsulamento e independência das colaborações quando combinadas entre si.

Estrutura de um APPC

- Interface Class Graph – ICG:
  - Declara os participantes (papéis) da colaboração;
  - Declara o padrão dos seus relacionamentos (como em um programa adaptativo, mas ao invés de os nós e as pontas do grafo serem classes concretas, serão variáveis “class-valued” e “link-valued”);

Estrutura de um APPC

- Behavioral Interface:
  - Parte do ICG que define assinaturas de métodos que os participantes da colaboração deverão executar (variáveis “method-valued”);

Estrutura de um APPC

- Behavior Definition:
  - Parte do APPC que conterá a ação a ser executada pela colaboração para se executar a tarefa, envolvendo todos os papéis nele definidos;
  - Utiliza os grafos de estratégia de trajeto (traversal strategies graphs), bem como os métodos definidos na behavioral-interface;

Estrutura de um APPC

- Behavior Definition:
  - Como em JAVA, tem um método distinto que define um “ponto de entrada” para a invocação da colaboração definida pelo APPC (“main-entry”);
  - Pode ter outras colaborações aninhadas (“traversal-driven”) que ocorrem durante o trajeto do fluxo de controle.

```
APPC Pricing {
   Interface Class Graph:
     // structural interface

     LineItemParty = <item> ItemParty <pricer> PricerParty <customer> Customer
     ItemParty = <charges> ListOf(ChargerParty)

     // behavioral interface

     LineItemParty { int quantity();}
     PricerParty {
           float basicPrice(ItemParty item);
           float discount(ItemParty item, int qty, Customer customer); }
     ChargerParty { float cost(int qty, float unitP, ItemParty item);}

   Behavior Definition:
       LineItemParty {
           main-entry float price(){
              float basicPrice, unitPrice;
              int discount, qty;
              qty = this.quantity();
              basicPrice = pricer.basicPrice(item);
              discount = pricer.discount(item, qty, customer);
              unitPrice = basicPrice - (discount * basicPrice);
              return (unitPrice + additionalCharges(unitPrice, qty)); }

           private float additionalCharges(float unitP, int qty) {
              float total;
              from LineItemParty via itemInst:  ItemParty to ChargerParty {
                 init {total = 0; }
                 at ChargerParty {total += cost(qty, unitP, itemInst); } }
                 return {total;} }
     } }
```

"traversal-driven"

Instanciando um APPC

- Consiste em mapear os objetos reais da aplicação com as variáveis definidas no ICG (“class-valued”, “link-valued”, “method-valued”);
- A colaboração formada pelo APPC será amarrada à aplicação (operação denotada com o símbolo “::+”);
- A aplicação resultante será criada por um gerador automático de código.

```
HWAppl::+ [float regularPrice() = Pricing with [
        LineItemParty = Quote;
        PricePartY = HWProduct
                [basicPrice = regPrice;
                discount = regDiscount];
        ItemParty = HWProduct;
        ChargerParty = Tax [cost = taxCharge];] ]

HWAppl::+ [float negotiatedPrice() = Pricing with [
        LineItemParty = Quote;
        PricePartY = Customer
                [basicPrice = negProdPrice;
                discount = negProdDiscount];
        ItemParty = HWProduct;
        ChargerParty = Tax [cost = taxCharge];]]
```