Modelo de Componentes CORBA

Cláudia Brito L. N. da Silva

cblns@cin.ufpe.br

Centro de Informática U·F·P·E

Roteiro

1. Visão geral
2. Desenvolvendo componentes CCM
   - Definindo componentes
   - Implementando componentes
   - Empacotando componentes
   - Instalando componentes
   - Executando componentes
3. Utilizando Componentes CCM
4. Considerações finais

Visão Geral

- O que é CCM?

O CCM representa um modelo de componente do lado do servidor, cuja finalidade é facilitar o desenvolvimento e a instalação de aplicações distribuídas compostas por componentes heterogêneos.

Visão Geral

- Padrão OMG - Faz parte da especificação CORBA 3
- Supre as limitações do modelo de objetos CORBA
  - falta de um mecanismo padrão de implantação (*deployment*) dos objetos;
  - falta de flexibilidade para estender as funcionalidades dos objetos, sendo possível apenas através de herança;
  - dificuldade de utilizar e configurar aplicações e serviços CORBA;
  - as conexões entre os objetos não são visíveis
  - requisitos não funcionais da aplicação, como transações, segurança, persistência etc., são codificados juntamente com os métodos de negócio.

Visão Geral

- É dividido em dois níveis de componentes:
  - *básico*: provêem uma forma simples de transformar um objeto CORBA em componente;
  - *estendido*: fornecem um conjunto maior de funcionalidades, como as portas de comunicação, que representam pontos de conexão entre os componentes

Visão Geral

- É estruturado em cinco modelos:
  - *modelo abstrato*:
    - definição dos atributos, portas de comunicação e *homes* dos componentes
  - *modelo de programação*:
    - composto pela CIDL (*Component Implementation Definition Language*) e pelo CIF (*Component Implementation Framework*)

Visão Geral

- É estruturado em cinco modelos:
  - *modelo de empacotamento*:
    - especifica como os componentes e suas implementações devem ser empacotados
  - *modelo de instalação*:
    - define um mecanismo padrão para a instalação de aplicações
  - *modelo de execução*:
    - define o ambiente de execução para as instâncias do componente

⇩

container

Desenvolvendo componentes CCM

(Definindo Componentes)

- Definição do componente em IDL3
  - ILD3 estende a IDL2 de forma a suportar conceitos relacionados ao CCM (portas, *homes* etc.)
- A definição agrupa a declaração dos atributos e das interfaces (portas de comunicação)
- Portas:
  - *facetas*: interfaces oferecidas
  - *receptáculos*: referências para outros objetos
  - *produtores de eventos*: utilizados para produzir eventos
  - *consumidores de eventos*: utilizados para consumir eventos

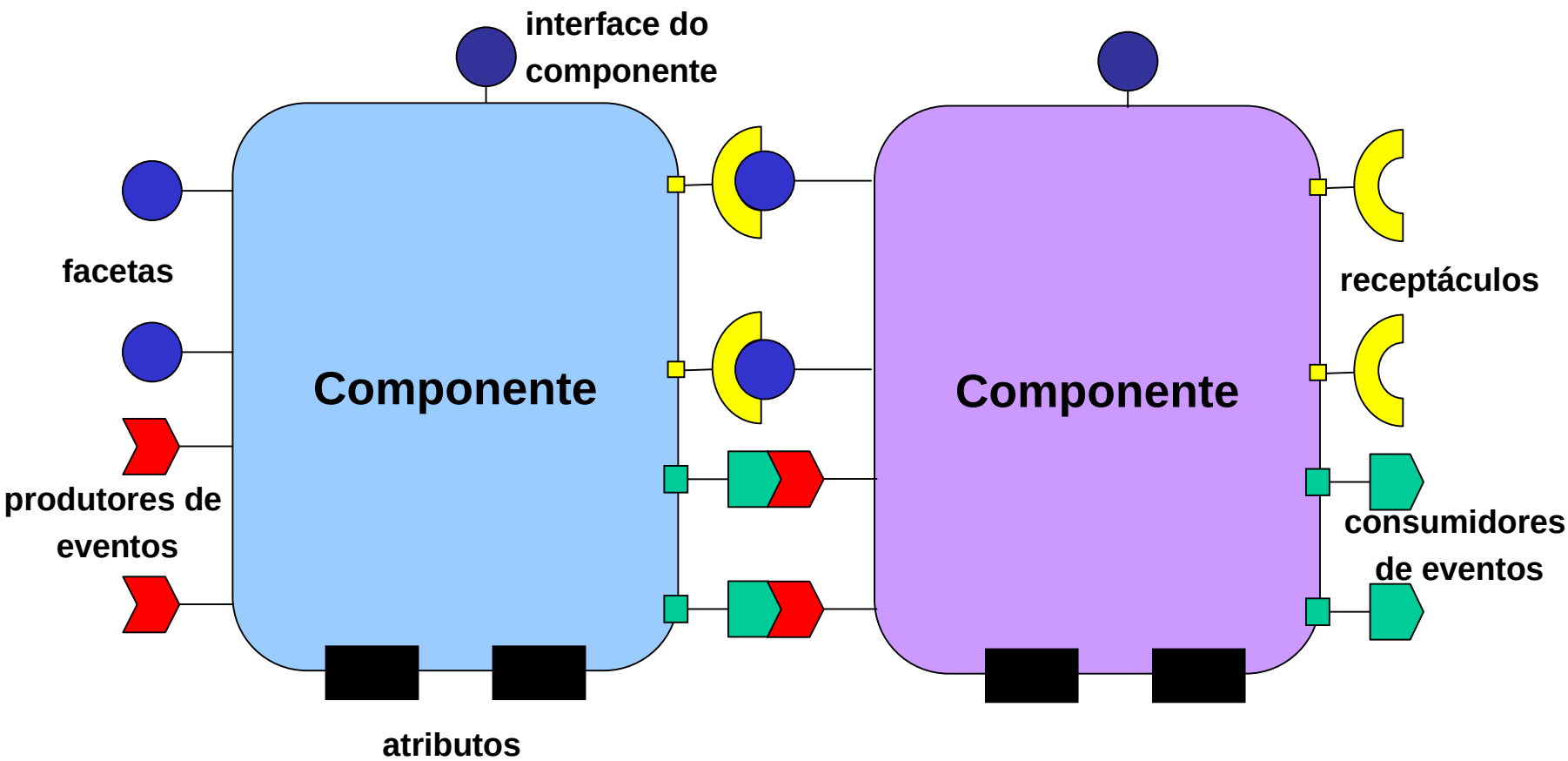
Desenvolvendo componentes CCM

(Definindo Componentes)

- Interface equivalente
  - Mapeamento de IDL3 para IDL2
  - Expõe as características do componente aos clientes

Desenvolvendo componentes CCM

(Definindo Componentes)



Desenvolvendo componentes CCM

(Definindo Componentes)

- Componente
  - Forma simples

```
component C1 {
  //definicao de portas e atributos
}
```

  - Interface suportada
    - Forma de definir um componente básico

objeto CORBA:

```
interface I1 {
  void methodX();
  string methodY();
}
```

componente CORBA:

```
component C1 supports I1 {
  //definicao de atributos
}
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Componente
  - Herança
    - Um componente pode herdar de um outro componente

```
component C1 supports I1 {
  //definicao de atributos
}
```

```
component C2: C1 {
  //definicao de portas e atributos
}
```

  - Interfaces suportadas e herança

```
interface I2 {
  void methodZ();
  string methodW();
}
```

```
component C3: C1 supports I2 {
  //definicao de portas e atributos
}
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Faceta
  - Serviço fornecido pelo componente
  - Declaração:

```
provides <tipoDaInterface> <nomeDaFaceta>;
```

    - Exemplo:

```
interface I3 {
  void methodA();
  int methodB(in string name);
}
```

```
provides I3 facetOne;
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Receptáculo
  - Referências para outros objetos com os quais o componente interage
  - Declaração:
    - *simples:*

```
uses <tipoDaInterface> <nomeDoReceptaculo>;
```

      - Exemplo:

```
interface I3 {
  void methodA();
  int methodB(in string name);
}
```

```
uses I3 receptacleOne;
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Receptáculo
  - *múltiplo:*

```
uses multiple <tipoDaInterface> < nomeDoReceptaculo >;
```

    - Exemplo

```
interface I3 {
  void methodA();
  int methodB(in string name);
}
```

```
uses multiple I3 receptacleOne;
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Eventos
  - Estabelecem uma comunicação assíncrona entre os componentes
  - Produtores de eventos
    - Expõem a capacidade do componente em produzir eventos
    - Dois tipos:
      - *publisher:* 1 para n (vários consumidores por vez)
      - *emitter:* 1 para 1 (apenas um consumidor)
  - Consumidores de eventos
    - Expõem a capacidade do componente em consumir eventos

Desenvolvendo componentes CCM

(Definindo Componentes)

- Eventos
  - Declaração:
    - Produtor (*publisher*)

```
publishes <tipoDoEvento> <nomeDoProdutor>;
```

    - Produtor (*emitter*)

```
emits <tipoDoEvento> <nomeDoEmissor>;
```

    - Consumidor

```
consumes <tipoDoEvento> <nomeDoConsumidor>;
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- Eventos
  - Exemplo:

```
eventType E1 {
  public string name;
}
```

    - » Produtor (*publisher*)

```
publishes E1 eventProdutor;
```

    - » Produtor (*emitter*)

```
emits E1 eventEmitter;
```

    - » Consumidor

```
consumes E1 eventConsumer;
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- *Homes*
  - Responsáveis pelo gerenciamento do ciclo de vida das instâncias de um determinado componente
  - Podem associar chaves primárias às instâncias do componente
  - Provêem operações para criar, localizar e remover instâncias de componentes
  - Gerencia apenas um único tipo de componente

Desenvolvendo componentes CCM

(Definindo Componentes)

- *Homes*
  - Declaração
    - *sem chave primária*

```
home <nomeDaHome> manages <tipoDoComponente> {
}
```

  - Exemplo:

```
component C4 {
   //definicao de atributos e portas
}
```

```
home H1 manages C4 { }
```

Desenvolvendo componentes CCM

(Definindo Componentes)

- *Homes*
  - Declaração
    - *com chave primária*

```
home <nomeDaHome> manages <tipoDoComponente> primaryKey <tipoDaChave>{ }
```

  - Exemplo:

```
component C4 {
  //definicao de atributos e portas
}
```

```
valuetype Id : Components::PrimaryKeyBase{
  public string value;
}
```

```
home H1 manages C4 primaryKey K1 { }
```

Desenvolvendo componentes CCM

(Implementando Componentes)

- CIF (*Component Implementation Framework*)
  - Automatiza o processo de desenvolvimento de componentes
  - *Skeletons* gerados a partir das descrições em CIDL
    - automatizam muito dos comportamentos básicos dos componentes (parte não-funcional)
- CIDL (*Component Implementation Definition Language*)
  - É uma linguagem descritiva assim como a IDL
  - Descreve composições (agregação de elementos)
    - *composition:* entidade que descreve todos os elementos requeridos para implementar um componente e sua *home*

Desenvolvendo componentes CCM

(Implementando Componentes)

- Elementos da *composition*:
  - Categoria do componente
    - *service*, *session*, *process*, *entity*
  - Nome do *skeleton* a ser gerado para o *home executor*
  - Tipo da *home* implementada
  - Nome do *skeleton* a ser gerado para o *component executor*
- Declaração de uma composição

```
composition <categoria> <nomeDaComposicao>{
  home executor <nomeDoHomeExecutor> {
    implements <tipoDaHome>;
    manages <nomeDoExecutor>;
  }
}
```

```
composition session Comp1{
  home executor he1 {
    implements H1;
    manages ce1;
  }
}
```

Desenvolvendo componentes CCM

(Implementando Componentes)

- *Component Executors* e *Home executors*
  - elementos de programação que implementam o comportamento dos componentes e de suas *homes*, respectivamente
- O compilador CIDL gera:
  - *Skeleton* do componente: código para navegação, identificação, ativação e gerenciamento de estado
  - Descritor do componente em XML

Desenvolvendo componentes CCM

(Empacotando Componentes)

- Pacote
  - Facilita o processo de instalação do componente
  - Arquivo ZIP:
    - Implementações do componente
    - Descritores XML
- Descritores
  - Gerados automaticamente ou através de ferramentas
  - Tipos:
    - *Software Package Descriptor* (.csd)
      - Informações gerais sobre o pacote (autor, descrição etc.)
      - Informações sobre as implementações (SO, linguagem de programação etc.)

Desenvolvendo componentes CCM

(Empacotando Componentes)

- Descritores
  - Tipos (cont…):
    - *CORBA Component Descriptor* (.ccd)
      - Descreve um componente
        - » Portas e interfaces
        - » Tipo do componente (*service*, *session*, *process* e *entity*)
        - » Características de transação, persistência etc.
      - Determina o tipo de *container* no qual o componente precisa ser instalado

Desenvolvendo componentes CCM

(Empacotando Componentes)

- Descritores
  - Tipos (cont…):
    - *Component Assembly Descriptor* (.cad)
      - Descreve os componentes utilizados no *assembly*
      - Informações de conexões entre os componentes
    - *Property File Descriptor* (.cpf)
      - Define os parâmetros de configuração de componentes e *homes*
      - Utilizado para configurar uma instância de um componente ou de uma *home* (atribuindo valores aos atributos)

Desenvolvendo componentes CCM

(Instalando Componentes)

- Processo padronizado
- Ferramentas de instalação
  - Automatizar o processo de instalação
  - Tendo informações sobre os *hosts*, se encarrega do processo restante
- Passos para a instalação
  1. identificar os *hosts* onde os componentes serão instalados;
  2. instalar as implementações dos componentes;
  3. instanciar *homes* e componentes nos *hosts*;
  4. conectar os componentes como especificado nos descritores.

Desenvolvendo componentes CCM

(Executando Componentes)

- *Container*
  - Ambiente de execução das instâncias
  - Esconde a complexidade relacionada ao gerenciamento do POA, e à associada à utilização dos serviços de sistema (transação, persistência, segurança)
  - Interação com as instâncias do componente
    - Interfaces internas: providas pelo *container*
    - Interfaces *callback*: providas pelo componente
    - Interfaces externas: acessadas pelo cliente, implementadas pelo componente

Desenvolvendo componentes CCM

(Executando Componentes)

- *Container*
  - Tipos de API
    - Depende da categoria do componente
      - *Service:* sem estado e identidade, tempo de vida transiente
      - *Session:* estado, identidade e tempo de vida transiente
      - *Process:* estado, identidade e tempo de vida persistente
      - *Entity:* mesmas caracterísitcas do '*process'*, identidade exposta ao clientes através de uma chave primária
    - APIs:
      - *Session:* referências transientes (*service*, *session*)
      - *Entity:* referências persistentes (*process*, *entity*)

Desenvolvendo componentes CCM

(Utilizando Componentes)

- Obter referência para o componente
  - Localizar a *home* do componente
    - Interface *HomeFinder*
      - *CORBA::ORB::resolve_initial_references("ComponentHomeFinder")*
      - Utilizar operações da *HomeFinder* para localizar a *home* desejada
    - Serviço de Nomes
  - Utilizar operações da *home* para criar ou localizar a instância do componente
- Utilizar a referência para invocar operações no componente

Considerações Finais

	Processo de desenvolvimento	Aspectos não-funcionais	Distribuição e configuração
CORBA 2.X	***ad hoc***	**codificados com os métodos de negócio**	***ad hoc***
CCM	**padronizado**	**descritos utilizando descritores XML**	**padronizado**

Considerações Finais

- Implementações existentes
  - OpenCCM
    - Primeira implementação disponível
    - Java / Código aberto
  - EJCCM
    - Java / Código aberto
  - CIAO
    - Otimizada para sistemas distribuídos embarcados e de tempo-real
    - C++ / Código aberto
  - MicoCCM
    - C++/ Código aberto

FIM...

Perguntas?!?!?!?!