MAC5755
Sistemas Operacionais Distribuídos
O modelo de componentes de CORBA (CCM)

Cesar Alison Monteiro Paixão

24/10/2001

Introdução

A necessidade de desenvolver, implantar e manter aplicações distribuídas complexas vem crescendo a cada dia. Tais aplicações requerem o uso de modelos técnicos para facilitar seu desenvolvimento e ambientes para aumentar sua confiabilidade. O *Common Object Request Broker Architecture* (CORBA) da *Object Management Group* (OMG) fornece um meio para criação de tais aplicações oferecendo ao mesmo tempo uma série de características interessantes como independência de linguagem, sistema operacional, hardware, localização e serviços como *naming*, *trading*, *event notification*, *security*, *transaction* [4, 5]. Infelizmente, a especificação do modelo de objetos CORBA até a versão 2.4 possui algumas limitações que podem ser descritas abaixo:

- **Não existe padrão para implantação de objetos:** As versões anteriores da especificação CORBA não definem um padrão para implantação das implementações de objetos nos processos servidores o que dificulta o processo de implantação ainda mais quando os objetos dependem uns dos outros.

- **Falta de suporte para o uso de subconjuntos de funcionalidades:** Um exemplo é o caso do (*Portable Object Adapter*) POA que é o mecanismo do ORB responsável por despachar as requisições dos clientes para os respectivos servidores. Existem um grande número de aplicações que utilizam só um pequeno subconjunto de funcionalidades do POA e mesmo assim o desenvolvedor deve conhecer muitas políticas deste para desenvolver tais aplicações.

- **Limitações ao estender as funcionalidades de objetos:** No modelo de objetos CORBA, os objetos só podem ser estendidos através de herança. Assim para suportar novas interfaces, os desenvolvedores de aplicações devem definir uma nova interface que herda todas as interfaces requeridas, implementar essa interface e finalmente implantar a implementação por todos os servidores desta aplicação. Assim estender a funcionalidade de um objeto é um processo lento e tedioso. Outra aspecto que deve ser considerado, é que herança múltipla utilizando a IDL (Interface Definition Language) CORBA não permite a exposição de uma mesma interface mais que uma vez, além de não suportar *overloading*, tendo assim sua aplicabilidade limitada.

- **A disponibilidade dos serviços de objetos CORBA não é definida antecipadamente:** A especificação CORBA não diz quais serviços de objetos são disponíveis em tempo de execução. Assim, os desenvolvedores de objetos devem usar algumas estratégias para configurar e ativar estes serviços quando implantam um sistema.

- **Não existe padrão no gerenciamento do ciclo de vida do objeto:** O CORBA Object Service define um serviço de ciclo de vida de objetos, mas não obriga sua utilização, fazendo com que os clientes sejam responsáveis por gerenciar o ciclo de vida do objeto explicitamente.

Essas limitações geram problemas como por exemplo a criação de aplicações com objetos difíceis de modelar, reutilizar, implantar, manter e estender. Com o objetivo de suprir essas limitações, a OMG adotou o CORBA Component Model (CCM) como parte da especificação do CORBA 3.0, previsto para ser lançado ainda neste ano.

CORBA Component Model (CCM)

O CCM é uma especificação que facilita a criação de aplicações escaláveis, independente de linguagem, multi-usuário, transacionais e seguras no lado do servidor. Para isso fornece uma estrutura consistente que estende o modelo de objetos do CORBA, definindo funcionalidades e serviços que permitem que o desenvolvedor implemente, gerencie, configure e implante componentes que integrem os serviços de CORBA em um ambiente padronizado, além de permitir maior reutilização de softwares para os servidores e fornecer maior flexibilidade para configuração dinâmica das aplicações [1, 5]. A especificação CCM é estruturada em cinco modelos (*Abstract Model*, *Programming Model*, *Packaging Model*, *Deployment Model*, *Execution Model* e um meta-modelo *Meta Model* [4]. Abaixo segue uma breve descrição de cada um desses modelos.

- ***Abstract model*** - fornece aos projetistas de aplicações um meio para definir as várias interfaces fornecidas e usadas pelo componente, tão bem como suas propriedades. A *Interface Definition Language* (IDL) da OMG foi estendinda para levar em conta os novos conceitos introduzidos no CCM, especialmente a definição de portas (*ports*) que é o meio para definir componente com múltiplas interfaces.

- ***Programming model*** - inclue um *framework*, o *Component Implementation Framework* (CIF) que define como funcional o código que é manualmente programado e não-funcional o código que é gerado automaticamente. Este *framework* conta com o uso do *Component Implementation Definition Language* (CIDL) para definir a maneira que a instância do componente interage com o *container* ao qual foi associado.

- ***Packaging model*** - especifica como componentes e implementações devem ser empacotadas. Os pacotes de componentes são então distribuídos e implantados sobre a rede. Este modelo usa um vocabulário XML (*eXtended Markup Language*) para descrever o empacotamento do software e suas dependências.

- ***Deployment model*** - define um processo que permite implantar uma aplicação sobre muitos sítios de uma maneira simples e automática. Este processo usa pacotes para instalar, criar instâncias e interconectar implementações de componentes, com objetivo de fornecer uma aplicação em tempo de execução.

- ***Execution model*** - descreve um ambiente de execução das instâncias de componentes. Tem como elemento principal o *container* que esconde e gerencia aspectos não-funcionais de um cliente, como ciclo de vida e persistência, dessa maneira não é mais necessário programar o uso do serviços CORBA utilizados.

- ***Meta model*** - define o conceito de muitos dos modelos acima mencionados, como seus relacionamentos. Este modelo é definido usando esquemas UML e inclui definições como tipo de componente e vários novos conceitos tais como portas (*ports*).

Neste texto esses modelos não serão estudados separadamente, o termo CCM será empregado para descrever todos os 6 modelos apresentados de uma maneira geral e quando falarmos em componentes deixamos subentendido que estes componentes utilizam a especificação CCM. A seguir introduziremos alguns conceitos importantes em uma arquitetura típica CCM como componentes e seus mecanismos de comunicação e *containers* e sua interação com os componentes e serviços do ORB.

Componentes

Os componentes são as peças fundamentais na construção de um sistema CCM. Eles podem disponibilizar multíplas interfaces, publicar/consumir eventos e interagir entre si através de mecanismos de portas que serão mencionados mais adiante. Existem 4 tipos de componentes que são divididos como segue.

- ***Service Components:*** Seu tempo de vida é restrito a uma simples requisição de operação ou seja, uma simples chamada de um método. São criados e destruídos pelo cliente CCM ao qual estão associados, e não sobrevivem a uma queda do sistema.
- ***Session Components:*** Cada *Session Component* é criado e destruído pelo cliente CCM ao qual esta associado. Um *Session Component* pode guardar ou não estado. Apesar disso, *Session Components* não sobrevivem a uma queda no sistema.
- ***Process Components:*** *Process Components* sempre guardam estado. Cada *Process Component* pode ser compartilhado por muitos clients CCM. Seus estados podem persistir sendo mantidos após múltiplas invocações e consequentemente *Process Components* podem sobreviver a quedas no sistema.
- ***Entity Components:*** *Entity Components* sempre guardam estado e como os *Process Components* podem ser compartilhados por muitos clientes CCM. Seus estados podem persistir sendo mantidos após múltiplas invocações e portanto *Entity Components* podem sobreviver a quedas no sistema. Cada *Entity component* pode ser identificado unicamente por sua *Primary Key*.

Uma das maiores diferenças entre Process e Entity Components é que enquanto um Entity Component possue uma Primary Key para ser identificado unicamente por um cliente, um Process Component não expõe uma identidade para o cliente exceto através de operações definidas pelo desenvolvedor.

Estrutura dos Componentes

O CCM fornece quatro mecanismos que possibilitam aos componentes interagir com entidades externas, tais como serviços fornecidos por um ORB, outros componentes, ou clientes (figura 1). Estes mecanismos são chamados *ports* (portas) e especificam diferentes visões e interfaces que um componente expõe para os clientes [3, 2]. Os quatro tipos de portas são descritas abaixo.

- ***Facets:*** também conhecidas como *provided interfaces*, são as interfaces fornecidas pelo componente e usadas de forma síncrona pelos clientes. Essas interfaces não são necessariamente relacionadas com as interfaces suportadas através de herança, diferentemente do modelo tradicional. Assim as *Facets* permitem aos componentes expor diferentes visões para os clientes através do fornecimento de diferentes interfaces.

- ***Receptacles :*** Antes de um componente delegar ações a outros componentes, ele precisa obter as referências para as instâncias dos componentes que irá usar. No CCM, estas referências são chamada *"object connections"* e a porta destas conecções são chamados *receptacles*. Podemos dizer que os *receptacles* é o nome dado aos pontos de conecção que descrevem a capacidade de usar uma referência a um objeto e que são usados de forma síncrona pelos componentes.

- ***Event sources:*** São as interfaces que descrevem a emissão de eventos de um determinado tipo. Os eventos emitidos são consumidos de forma assíncrona por componentes que tiverem interesse. Assim um componente declara seu interesse de emitir eventos especificando "events sources" em sua definição.

- ***Event sinks:*** São as interfaces que descrevem o "consumo" de eventos de um determinado tipo. Assim um componente declara seu interesse de consumir eventos especificando "events sinks" em sua definição.

Além dos mecanismos de portas um componente possui outras propriedades importantes , as quais podemos citar:

- ***Attributes :*** São os atributos ou propriedades configuráveis de um componente. Para possibilitar a configuração de componentes, o CCM estende o conceito de *attributes* definidas no modelo de objetos CORBA tradicional. Os atributos podem ser usados por ferramentas de configuração para alterar valores de configuração de um componente. O CCM também permite operações que acessam e

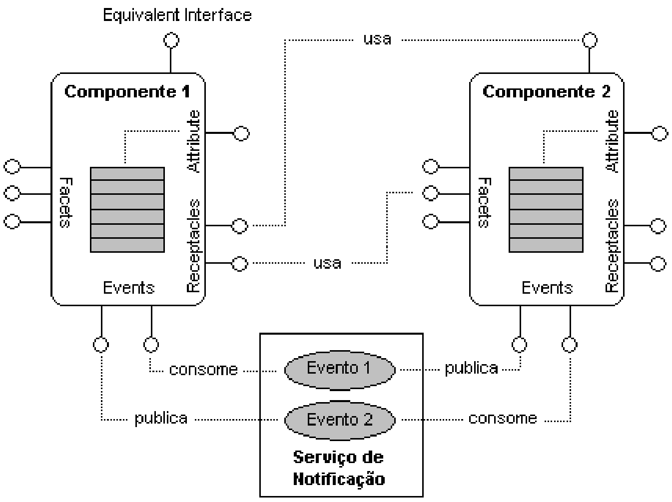

Figura 1: Comunicação entre componentes através do mecanismo de *ports*

modificam valores dos atributos para gerar exceções, permitindo que desenvolvedores criem aplicações que gerem exceções se ações indesejáveis ocorrerem.

- ***Primary Key:*** Uma *primary key* ou chave primária pode ser usada para disponibilizar um meio ao qual clientes podem localizar, criar e destruir instâncias de um componente, apenas associando o respectivo valor da chave primária deste.

- ***Home Interface:*** É a interface que gerencia instâncias de um componente. Fornece operações para gerenciar o ciclo de vida dos componentes e opcionalmente podem gerenciar associações entre instâncias de componentes e valores de chaves primárias (*primay keys*) [1]. Define a *equivalent interface* , que pode ser descrita em termos da IDL CORBA.

- ***Equivalent Interface:*** Uma *home interface* define implicitamente um conjunto de operações que possuem o mesmo nome em todas interfaces *home*, mas que são específicas ao componente gerenciado por esta home. O fato destas operações possuírem o mesmo nome em diferentes *homes* impossíbilita o processo

[1] Uma *home interface* que define uma *primary key* suporta operações de busca de uma instância de componente por parte do cliente, uma *home interface* onde não foi definida uma *primary key* não suporta este tipo de operação. Todas as *homes* suportam operações de criação e destruição de instâncias.

de herança destas interfaces . A *equivalent interface* quebra esta restrição identificando unicamente uma instância de um componente.

Estes novos mecanismos aumentam significativamente a reutilização de componentes quando comparados ao modelo tradicional de objetos do CORBA, facilitando o trabalho dos clientes que agora não precisam ter conhecimento explícito dos detalhes de implementação do componente para acessar uma interface deste e dos desenvolvedores através das padronizações trazidas pelo CCM, como a padronização dos mecanismos de configuração. Agora a funcionalidade de um componente pode ser estendida apenas adicionando novas interfaces a implementação deste sem afetar o uso de seus clientes. O CCM também define a *CCM Navigation interface* que enumera todas as *facets* fornecidas pelo componente , disponibilizando assim um meio ao qual um cliente pode verificar se uma certa interface é fornecida por um componente.

External Interfaces

As interfaces disponíveis para um cliente são conhecidas como *external interfaces* (interfaces externas). Existem dois tipos de interfaces externas as *home interfaces* e as *application interfaces* (interfaces da aplicação). O acesso a uma interface da aplicação é feito através de uma referência a *home interface* do componente, que dispõe de atributos e ferramentas para tal fim.

Component Implementation Definition Language (CIDL)

Afim de suportar a estrutura e funcionalidade dos componentes o CCM estende a IDL (*Interface Definition Language*) do CORBA introduzindo uma nova linguagem declarativa a *Component Implementation Definition Language* (CIDL). Esta linguagem é usada para descrever implementações, persistência de estado e *homes* dos componentes permitindo que estes utilizem uma combinação de diferentes interfaces não relacionadas [3, 2, 5]. Implementações geradas por um compilador CIDL são chamadas *executors* (figura 2). *Executors* suportam a automação das implementações e

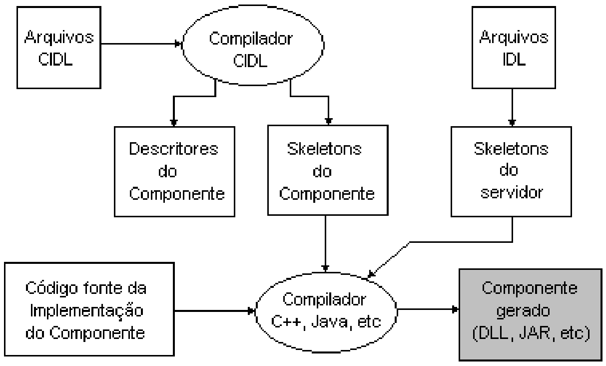

Figura 2: Processo de desenvolvimento de componentes utilizando a CIDL

fornece métodos que permitem desenvolvedores de componentes adicionar ações comuns a um componente específico, eles podem ser empacotados em DLLs e instalados em um servidor de componentes que roda um uma plataforma específica (Windows, Linux, etc). A CIDL é então responsável pela geração dos descritores do componente (*component descriptors*) que definem as características de um componente descrevendo as interfaces, políticas de threading ou transação e tipo de serviços requisitados por este [5].

Component Implementation Framework (CIF)

O *Component Implementation Framework* (CIF) define um modelo de programação para construção de implementações de componentes. Após os desenvolvedores declararem os componentes utilizando a CIDL eles utilizam o CIF para construir a implementação destes. O CIF usa as descrições da CIDL para gerar *programming skeletons* que automatizam o comportamento do núcleo do componente, como navegação, requisição de identidade, ativação e gerenciamento de estado (figura 2). O CIF define um conjunto de APIs que gerenciam o estado persistente dos componentes, assim um componente pode reconstituir seu último estado sempre que uma instância dele for ativada. Como já foi visto o CCM define muitas interfaces na criação de componentes.

Muitas dessas interfaces são construídas com a mesma estrutura independentemente do componente utilizado, o CIF alivia os desenvolvedores do trabalho de escreverem essas interfaces sempre que forem criar um componente permitindo a automação de atividades comuns a componentes.

Containers

Os *containers* atuam como interface entre um componente CORBA e o mundo exterior, simplificando as tarefas de desenvolvimento e/ou configuração das aplicações. Eles fornecem um ambiente em tempo de execução para gerenciar componentes que pode:

- Ativar ou desativar implementações de componentes, preservando assim recursos do sistema, tais como memória.
- Despachar requisições de clientes para os serviços CORBA como os serviços de transação, persistência, segurança e notificação, liberando o cliente de tal trabalho.
- Fornecer camadas de adaptação para *callbacks* usados pelo *container* e ORB para informar o componente sobre eventos interessantes, tais como mensagens de transação ou a notificação de serviços.
- Gerenciar políticas do POA determinando como criar referências a componentes.

Um *container* é composto basicamente por 2 tipos de interfaces : ***Internal Interfaces*** que disponibilizam as funcionalidades e serviços do *container* ao componente e ***Callback Interfaces*** que são invocadas pelo *container* e implementadas pelo desenvolvedor no componente (figura 3). Assim um componente acessa as funcionalidades do ORB através do *container* ao qual ele foi encapsulado.

Dependendo do tipo de componente que encapsula os *containers* podem ser divididos em *Service Containers*, *Session Containers*, *Process Containers* e *Entity Containers* e de acordo com o estado dos componentes suportados os *containers* são chamados de *Transient Containers* ou *Persistent Containers* (Tabela 1). Um *container* CCM pode conter também um componente EJB ("bean") , neste caso ele é chamado de EJB *container*.

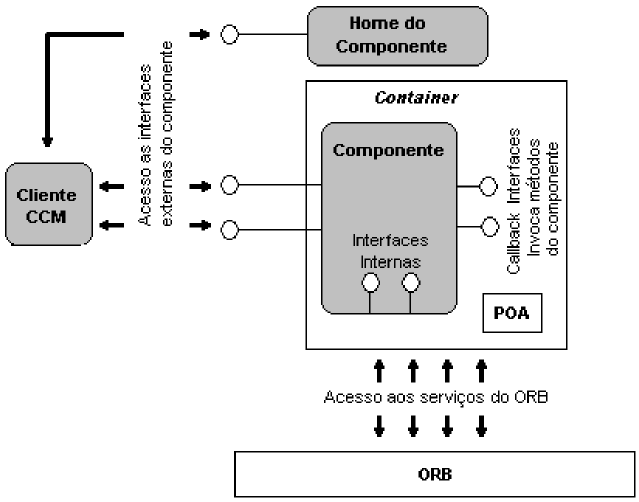

Figura 3: Comunicação (Cliente - *Container* - Componente - ORB)

Cada tipo de *container* gerencia uma implementação de componente definida pela CIF criando e configurando automaticamente seu próprio POA para todas as interfaces gerenciadas por ele livrando o desenvolvedor de tal tarefa. Os *containers* também gerenciam o tempo de vida dos componentes encapsulados através de políticas que controlam o momento de ativação e desativação destes componentes, como *Method*, *Session*, *Component* e *Container*. As políticas *Method*, *Session* ativam ou desativam um componente a cada chamada de método ou a cada início de sessão respectivamente. Já as políticas *Component*, *Container* delegam o gerenciamento do tempo de vida aos componentes e *containers* respectivamente.

Tipo de Componente	***Container***	**Identidade Externa**
Service	*Transient*	Sem chave
Session	*Transient*	Sem chave
Process	*Persistent*	Sem chave
Entity	*Persistent*	Com chave

Tabela 1: Relação Tipo de Componente - Container - Identidade

CCM sob a perspectiva do cliente

Os clientes fazem uso dos componentes CCM para realizar suas tarefas, eles podem acessar diretamente as interfaces externas do componente, tais como as *equivalent interface* e *facets* (figura 3). Para que o acesso às interfaces da aplicação sejam feitos diretamente pelos clientes, estes precisam obter uma referência para *Home Interface* do componente, que suporta operações que permitem ao cliente obter uma referência à interface de aplicação desejada. Um cliente pode acessar a *Home Interface* também para controlar o ciclo de vida de cada instância do componente que usa, ou seja através da *Home Interface* um cliente pode criar e remover instâncias de componentes. Para localizar a *Home Interface* de um componente específico, o cliente utiliza um mecanismo chamado *Home Finder* que localiza a referência para a *home* do componente em uma base de dados centralizada similar ao CORBA *Interoperable Naming Service*. Após obter a referência para a *Home Interface* do componente desejado, um cliente pode invocar operações sobre as instâncias deste componente.

CCM sob a perspectiva do desenvolvedor

Os desenvolvedores de componentes usam a CIDL para definir componentes e especificar operações que estes componentes suportam, da mesma maneira que as interfaces dos objetos são definidas tradicionalmente no modelo de objetos CORBA. Neste processo o desenvolvedor deve definir uma *home interface* para cada componente. Em seguida, eles utilizam o CIF na implementação do componente que pode ser empacotada em um arquivo como uma biblioteca compartilhada , um arquivo JAR, ou uma DLL e assim ativada dinâmicamente. Os desenvolvedores podem gerenciar o ciclo de vida de um componente especificando uma estratégia para cada componente e através das portas os desenvolvedores podem iniciar conecções entre objetos, consumir ou publicar eventos, e estabelecer atributos aos componentes permitindo a configuração dos mesmos. Após ter sido implementado o componente é acoplado a um *container* associado ao seu tipo e implantado em um servidor de componentes (que pode suportar outros *containers*) através do carregamento de seus arquivos, DLL, etc. Um servidor de componentes muitas vezes não tem conhecimento prévio de como configurar e instanciar os componentes que serão implantados nele, e por essa razão os desenvolvedores devem criar interfaces genéricas para facilitar a instalação e gerenciamento dos componentes. Além disso, o CCM define uma interface padrão de configuração chamada Components::StandardConfigurator que ajuda os servidores de componentes na configuração destes. Finalmente, os componentes hospedados em um servidor de componentes são capazes de processar requisições de clientes.

EJB X CCM

A especificação do CCM foi modelada usando como base a especificação do *Enterprise Java Beans* (EJB) [5]. Aproveitando o fato de que já existem muitos produtos que utilizam EJB no mercado, os criadores do CCM também definiram na especificação um padrão de mapeamento entre os dois modelos. Assim, um componente CCM pode se comportar como um "bean" EJB para os clientes EJB e "beans" do EJB podem funcionar como componentes CCM, desde que se faça um bom uso das técnicas de mapeamento. Como o CCM usa CORBA como arquitetura de interoperabilidade de objetos, ele não fica limitado a uma linguagem em particular enquanto o EJB é limitado ao uso de Java. Outro aspecto que pode ser levantado é que as atuais (e poucas) implementações de CCM possuem código aberto diferentemente das implementações do EJB, mas deixamos a discussão de qual desses processos é melhor para o futuro de uma tecnologia a cargo do leitor.

Alguns criadores do CCM questionam o EJB como um modelo distribuído desde que o EJB dá suporte a um único servidor enquanto o CCM suporta múltiplos servidores. Em compensação podemos citar o fato de que existem poucas e imcompletas implementações do CCM e portanto ainda não se pode garantir a eficiência nas aplicações que utilizam este modelo, ou seja não se sabe como o CCM se comportará nas primeiras implementações do modelo completo, enquanto que o EJB já esta consolidado neste aspecto.

Analisando superficialmente as duas tecnologias (Tabela 2) podemos dizer que o

CCM é mais flexível e poderoso que o EJB, mais ainda, o EJB pode ser visto como um subconjunto do CCM desde que um *container* do CCM pode hospedar componentes do EJB. Para concluir podemos dizer que o EJB e o CCM por serem tecnologias similares podem agir em conjunto na construção de aplicações distribuídas complexas.

Aspecto	**EJB**	**CCM**
Disponibilidade	Muitas implementações	Poucas implementações
Conecções	Implementadas no Java Beans	Quatro tipos de portas
Programação	Java	Qualquer linguagem
Implantação	Único Servidor	Múltiplos Servidores

Tabela 2: Comparação entre EJB e CCM

Conclusão

Com o progressivo aumento no tamanho e complexidade das aplicações distribuídas e devido ao fato da natureza de muitas aplicações estar diretamente relacionada com a construção e interação de componentes, os desenvolvedores de aplicações necessitam cada vez mais de um modelo de componentes que seja flexível e poderoso. O *CORBA Component Model* (CCM) traz significativas melhorias na organização dos objetos em componentes, facilitando o trabalho de modelagem, desenvolvimento, empacotamento, implantação e execução destes além de manter a interoperabilidade e neutralidade de linguagem fornecidas pelo CORBA. Apesar de ainda não ter implementações completas disponíveis, as perpectivas em relação ao futuro do CCM são muito boas devido a alguns aspectos dentre os quais podemos citar:

- A especificação CCM nasceu baseada em um modelo já bem sucedido que é o EJB.
- A crescente utilização do CORBA que vem se estabilizando como um padrão industrial.

- A compatibilidade com componentes EJB, dado que existe um grande número de aplicações no mercado utilizando esses componentes.
- Suporte a múltiplos servidores.

Considerando esses aspectos, percebemos que o CCM é um modelo interessante na criação de aplicações distribuídas mais complexas. Acreditamos que com o lançamento do CORBA 3.0, o interesse das empresas e pesquisadores aumente e com isso boas e completas implementações sejam desenvolvidas. Entretanto pensamos que devemos esperar o aparecimento destas versões para avaliarmos mais profundamente se todas as vantagens apresentadas aqui são realmente viáveis na prática, ou melhor, se as novas características trazidas pelo CCM não ficarão encobertas por um mau desempenho das aplicações escritas segundo esta especificação.

Referências Bibliográficas

[1] Golapan Suresh Raj. The corba component model (ccm). *Gopalan Suresh Raj's Web Cornucopia , www.execpc.com/˜ gopalan/corba/ccm.html.*

[2] OMG, Object Management Group. Corba 3 new components chapters. *OMG TC Document*, Outubro 1999.

[3] OMG, Object Management Group. Corba components: Joint revised. *OMG TC Document orbos*, Agosto 1999.

[4] Raphaël Marvie, Philippe Merle. Corba component model : Discussion and use with open ccm. *Laboratorie d'Informatique Fondamentale de Lille.*

[5] Wang, Schimidt, O'Ryan. Overview of the corba component model. *Department of Computer Science - Washington University in St. Louis , www.cs.wustl.edu/ schmidt/PDF/CBSE.pdf.*