

## Objetos Distribuídos

- Java RMI
- CORBA

1

## Objetos Distribuídos

- Orientação a Objetos
  - Encapsulamento:
    - Parte interna (privada) dos objetos
      - Implementação: métodos
      - Estado: atributos, variáveis, constantes e tipos
    - Parte externa (pública) dos objetos
      - Interface: conjunto bem definido de métodos públicos que podem ser acessados externamente

2

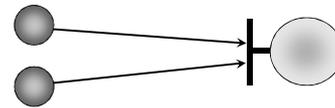
## Objetos Distribuídos

- Orientação a Objetos (cont.)
  - Herança: de interfaces e implementações
  - Polimorfismo: a mesma interface pode ter várias implementações
  - Interação entre objetos
    - Troca de mensagens (chamadas de métodos)
    - Mensagens podem ser locais ou remotas
      - Mensagens locais: objetos no mesmo espaço de endereçamento
      - Mensagens remotas: objetos em máquinas diferentes → distribuídos!

3

## Objetos Distribuídos

- Orientação a Objetos (cont.)
  - Referência do objeto → Ponteiro de memória
  - O acesso ao estado do objeto é feito através dos métodos da interface (única parte visível do objeto)
  - Implementação independente da interface
  - Métodos são acessados por outros objetos



4

## Objetos Distribuídos

- Objetos Distribuídos
  - Interagem através da rede
  - Colaboram para atingir um objetivo
  - Fornecem serviços (métodos) uns aos outros
  - Apenas a interface do objeto é visível
  - Referência do objeto possui endereço de rede



5

## Objetos Distribuídos

- Problemas
  - Como compartilhar referências de objetos?
  - Como gerenciar o ciclo de vida dos objetos?
  - Como gerenciar o acesso concorrente aos objetos?
  - Como trabalhar num ambiente heterogêneo?
    - Máquinas podem ter arquiteturas diferentes
    - Máquinas podem estar em redes diferentes
    - Máquinas podem rodar S.O.'s diferentes
    - Objetos podem ser implementados em linguagens diferentes

6

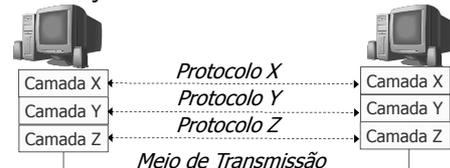
## Objetos Distribuídos

- Problemas (cont.)
  - Comunicação não confiável e não-determinista: depende da dinâmica do sistema e da rede
  - Custo da comunicação: latência e largura de banda são fatores críticos em aplicações de tempo real, multimídia, etc.
  - Comunicação insegura: sem controle de autorização e sem proteção das mensagens

7

## Objetos Distribuídos

- Protocolos de Comunicação
  - Estabelecem caminhos virtuais de comunicação entre duas máquinas
  - Devem usar os mesmos protocolos para trocar informações



8

## Objetos Distribuídos

- Protocolos de Comunicação (cont.)
  - Serviço sem Conexão: cada unidade de dados é enviada independentemente das demais



- Serviço com Conexão: dados são enviados através de um canal de comunicação



9

## Objetos Distribuídos

- Protocolos de Comunicação (cont.)
  - Protocolos de alto nível são necessários para interação entre objetos distribuídos
  - Escolha natural: usar TCP/IP
    - Cria conexões entre processos para trocar mensagens
    - Amplamente disponível, confiável e robusto
    - Relativamente simples e eficiente
    - Não mascara o uso da rede do programador

10

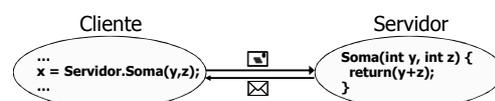
## Objetos Distribuídos

- Protocolo de Comunicação entre Objetos
  - Trata questões não resolvidas pelo TCP/IP
    - Formato comum dos dados
    - Localização de objetos
    - Segurança
  - Oferece ao programador abstrações próprias para aplicações orientadas a objetos
    - Chamada Remota de Procedimento (RPC) ou Invocação Remota de Métodos (RMI)
    - Notificação de Eventos

11

## Objetos Distribuídos

- RPC – Chamada Remota de Procedimento
  - Segue o modelo Cliente/Servidor
  - Muito usado na interação entre objetos
  - Objeto servidor possui interface com métodos que podem ser chamados remotamente
  - Objetos clientes usam serviços de servidores



12

## Objetos Distribuídos

- RPC – Características
  - Em geral as requisições são ponto-a-ponto e síncronas
  - Dados são tipados
    - Parâmetros da requisição
    - Retorno do procedimento/método
    - Exceções
  - Um objeto pode ser cliente e servidor em momentos diferentes

13

## Objetos Distribuídos

- RPC – Sincronismo
  - Chamada síncrona: cliente fica bloqueado aguardando o término da execução do método



- Chamadas semi-síncronas: sincronização é retardada; permitidas em alguns sistemas



14

## Objetos Distribuídos

- RPC – Sincronismo (cont.)
  - Chamadas assíncronas: cliente continua a execução sem aguardar o retorno do método; permitidas em alguns sistemas



15

## Objetos Distribuídos

- RPC – Funcionamento
  - Chamada é feita pelo cliente como se o método fosse de um objeto local
  - Comunicação é feita transparentemente por código gerado automaticamente pelo compilador (*stub*, *proxy*, *skeleton*, ...)
  - O código gerado faz a serialização e desserialização de dados usando um formato padrão, que compatibiliza o formato de dados usado por diferentes máquinas, linguagens e compiladores

16

## Objetos Distribuídos

- RPC – Funcionamento do Cliente
  - Acessa objeto local gerado automaticamente que implementa interface do servidor remoto

```

Public class HelloServerStub {
    public String hello(String nome) {
        // Envia pela rede o identificador do método e o valor dos ...
        // ... parâmetro(s) da chamada serializados para o servidor
        // Recebe do servidor o valor do retorno da chamada pela ...
        // ... rede, o deserializa e retorna o valor recebido ao cliente
    }
    // Outros métodos ...
}
    
```

17

## Objetos Distribuídos

- RPC – Funcionamento do Servidor
  - O código gerado automaticamente recebe as chamadas pela rede e as executa

```

while (true) {
    // Recebe pela rede o identificador do método chamado ...
    // ... pelo cliente e os parâmetros da chamada serializados
    // Desserializa os parâmetros enviados pelo cliente
    // Chama o método no objeto servidor e aguarda a execução
    // Serializa o valor do retorno da chamada e envia ao cliente
}
    
```

18

## Objetos Distribuídos

- RPC – Implementação
  - Descrição da interface do objeto remoto
    - Especificada na própria linguagem de programação
    - Especificada usando uma linguagem de descrição de interface (IDL)
  - Implementações de RPC de diferentes fabricantes (Sun RPC, DCE RPC, Microsoft RPC, etc.) são geralmente incompatíveis

19

## Objetos Distribuídos

- Notificação de Eventos
  - Eventos ocorridos são difundidos por produtores e entregues a consumidores
  - Canal de eventos permite o desacoplamento – produtor e consumidor não precisam se conhecer



20

## Objetos Distribuídos

- Notificação de Eventos – Características
  - Envio de eventos é completamente assíncrono
    - Produtor não precisa aguardar fim do envio
    - Evento é armazenado no canal de eventos
  - Comunicação pode ser feita através de UDP *multicast* ou fazendo múltiplos envios *unicast* com TCP, UDP ou com um suporte de RPC
  - Os eventos podem ter tamanho fixo ou variável, limitado ou ilimitado
  - Eventos podem ser tipados ou não

21

## Objetos Distribuídos

- Para suportar a interação entre objetos distribuídos, um *Middleware* deve prover:
  - Localização transparente dos objetos
  - Invocação de métodos local e remoto idêntica
  - Criação de objeto local e remoto idêntica
  - Migração de objetos transparente
  - Facilidades para ligação (*binding*) de interfaces dinamicamente
  - Diversos serviços de suporte:
    - Nomes, Transação, Tempo, etc.

22

## Objetos Distribuídos

- Principais suportes de Middleware para Objetos Distribuídos
  - Java RMI (*Remote Method Invocation*), da Oracle/Sun Microsystems
  - COM (*Component Object Model*), da Microsoft Corporation
  - CORBA (*Common Object Request Broker Architecture*), especificado pela OMG (*Object Management Group*) e implementado por diversos fabricantes de software

23

## Java RMI

- Java
  - Orientada a objetos
  - Possui diversas APIs amigáveis
  - Multi-plataforma: *Java Virtual Machine* (JVM)
  - Integrada à Internet: *applet*, *servlet*, JSP, JSF
  - Suporte a componentes: *JavaBeans* e EJB
  - De fácil aprendizagem
  - Bem aceita pelos programadores
  - Suportada por diversos fabricantes de SW



24

## Java RMI

- Java é oferecida em três versões
  - Java ME (*Micro Edition*)
    - Para celulares, sist. embarcados, etc.
  - Java SE (*Standard Edition*)
    - Para desktops
  - Java EE (*Enterprise Edition*)
    - Para servidores
- Versões diferem nas APIs oferecidas
- Java SE e EE possuem suporte para invocação remota de métodos (RMI)

25

## Java RMI

- Java RMI (*Remote Method Invocation*)
  - Fornece um suporte simples para RPC/RMI
  - Permite que um objeto Java chame métodos de outro objeto Java rodando em outra JVM
  - Solução específica para a plataforma Java



26

## Java RMI

- Arquitetura RMI
  - *Stub* e *Skeleton*
  - Camada de referência remota
  - Camada de transporte



27

## Java RMI

- *Stub*
  - Representa o servidor para o cliente
  - Efetua serialização e envio dos parâmetros
  - Recebe a resposta do servidor, desserializa e entrega ao cliente
- *Skeleton*
  - Recebe a chamada e desserializa os parâmetros enviados pelo cliente
  - Faz a chamada no servidor e retorna o resultado ao cliente

28

## Java RMI

- Camada de Referência Remota
  - Responsável pela localização dos objetos nas máquinas da rede
  - Permite que referências para um objeto servidor remoto sejam usadas pelos clientes para chamar métodos
- Camada de Transporte
  - Cria e gerencia conexões de rede entre objetos remotos
  - Elimina a necessidade do código do cliente ou do servidor interagirem com o suporte de rede

29

## Java RMI

- Dinâmica da Chamada RMI
  - O servidor, ao iniciar, se registra no serviço de nomes (RMI *Registry*)
  - O cliente obtém uma referência para o objeto servidor no serviço de nomes e cria a *stub*
  - O cliente chama o método na *stub* fazendo uma chamada local
  - A *stub* serializa os parâmetros e transmite a chamada pela rede para o *skeleton* do servidor

30

## Java RMI

- Dinâmica da Chamada RMI (cont.)
  - O *skeleton* do servidor recebe a chamada pela rede, desserializa os parâmetros e faz a chamada do método no objeto servidor
  - O objeto servidor executa o método e retorna um valor para o *skeleton*, que o desserializa e o envia pela rede à *stub* do cliente
  - A *stub* recebe o valor do retorno serializado, o desserializa e por fim o repassa ao cliente

31

## Java RMI

- Serialização dos dados (*marshalling*)
  - É preciso serializar e desserializar os parâmetros da chamada e valores de retorno para transmiti-los através da rede
  - Utiliza o sistema de serialização de objetos da máquina virtual
    - Tipos predefinidos da linguagem
    - Objetos serializáveis: implementam interface `java.io.Serializable`

32

## Java RMI

- Desenvolvimento de Aplicações com RMI
  - Devemos definir a interface do servidor
    - A interface do servidor deve estender `java.rmi.Remote` ou uma classe dela derivada (ex.: `UnicastRemoteObject`)
    - Todos os métodos da interface devem prever a exceção `java.rmi.RemoteException`
    - O Servidor irá implementar esta interface
  - *Stubs* e *skeletons* são gerados pelo compilador RMI (`rmic`) com base na interface do servidor

33

## Java RMI

- RMI/IIOP
  - O RMI permite a utilização do protocolo IIOP (Internet Inter-ORB Protocol) do CORBA
  - IIOP também usa TCP/IP, mas ao efetuar a serialização (*marshalling*) converte os dados para um formato binário diferente do Java RMI
  - Com RMI/IIOP, objetos Java podem se comunicar com objetos CORBA escritos em outras linguagens

34

## Java RMI

- APIs úteis na comunicação remota
  - JNDI (*Java Naming and Directory Interface*)
    - Suporte para nomeação
    - Associa nomes e atributos a objetos Java
    - Objetos localizados por nome ou atributos
  - *JavaSecurity*
    - Suporte para segurança
    - Criptografa dados
    - Cria e manipula chaves e certificados
    - Emprega listas de controle de acesso

35

## CORBA

- OMG (*Object Management Group*):
  - Formada em 1989
  - Objetivos:
    - Promover a teoria e prática de tecnologias O.O. no desenvolvimento de software
    - Criar especificações gerais e proveitosas: definir interfaces, e não implementações
  - Composta por cerca de 800 empresas interessadas no desenvolvimento de software usando tecnologia de objetos distribuídos

36

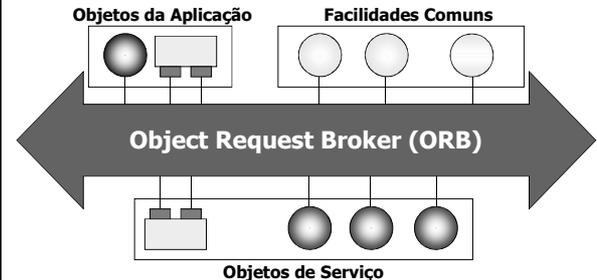
## CORBA

- **OMA (Object Management Architecture)**
  - Infra-estrutura sobre a qual todas as especificações da OMG estão baseadas
  - Define apenas aspectos arquiteturais
  - Permite interoperabilidade entre aplicações baseadas em objetos em sistemas abertos, distribuídos e heterogêneos
    - Diferentes máquinas
    - Diferentes sistemas operacionais
    - Diferentes linguagens de programação
  - Maior portabilidade e reusabilidade
  - Funcionalidade transparente para a aplicação

37

## CORBA

- **OMA**



38

## CORBA

- **OMA**
  - **Objetos da Aplicação**
    - Definidos pelos usuários/programadores
  - **Facilidades Comuns**
    - Grupos de objetos que fornecem serviços para determinadas áreas de aplicação
  - **Objetos de Serviço**
    - Serviços de propósito geral usados por objetos distribuídos
  - **Object Request Broker (ORB)**
    - Canal de comunicação entre objetos

39

## CORBA

- **CORBA (Common Object Request Broker Architecture)**
  - Define concretamente as interfaces do ORB, especificado de forma abstrata pela Arquitetura OMA
  - Permite a interação entre objetos distribuídos
  - Fornece um suporte completo para desenvolver aplicações distribuídas orientadas a objetos

40

## CORBA

- **Histórico**
  - A versão 1.0 do CORBA foi proposta em 1991
  - CORBA começou a se estabelecer a partir de 1993, com o surgimento das primeiras implementações de ORBs comerciais
  - CORBA 2.0 foi lançado em 1996
    - Interoperabilidade entre implementações
  - Versão 3.0 foi lançada em 2002
    - Acrescentou suporte a componentes (CCM), invocações assíncronas de métodos (AMI), mensagens (CORBA Messaging), ...

41

## CORBA

- **CORBA proporciona total transparência para os Objetos Distribuídos**
  - **Transparência de Linguagem**
    - Usa IDL (*Interface Definition Language*)
  - **Transparência de S.O. e Hardware**
    - ORB pode ser implementado em várias plataformas: Windows, UNIX, SO's embarcados e de tempo real, ...
  - **Transparência de Localização dos Objetos**
    - Objetos são localizados através de suas referências, que são resolvidas pelo ORB

42

## CORBA

- IDL (*Interface Definition Language*)
  - Usada para descrever as interfaces de objetos
  - Linguagem puramente declarativa, sem nenhuma estrutura algorítmica
  - Sintaxe e tipos de dados baseados em C/C++
  - Define seus próprios tipos de dados, que são mapeados nos tipos de dados de cada linguagem de programação suportada
  - Mapeada para diversas linguagens
    - C, C++, Java, Delphi, COBOL, Python, ADA, Smalltalk, LISP, ...

43

## CORBA

- Compilador IDL
  - Gera todo o código responsável por:
    - Fazer a comunicação entre objetos
    - Fazer o mapeamento dos tipos de dados definidos em IDL para a linguagem usada na implementação
    - Fazer as conversões de dados necessárias na comunicação (*serialização/ marshalling* dos dados)

44

## CORBA

- Interação entre objetos no CORBA
  - Segue o modelo Cliente-Servidor
    - Cliente: faz requisições em objs. remotos
    - Implementação de objeto: implementa os serviços descritos na sua interface



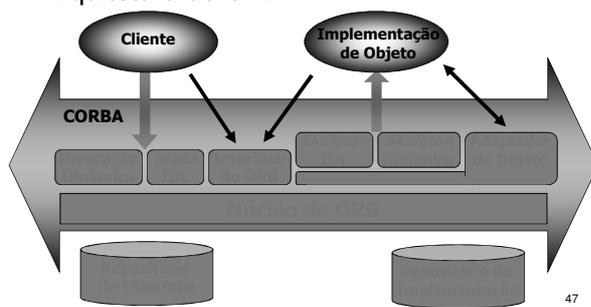
## CORBA

- Objetos CORBA possuem:
  - Atributos: dados encapsulados pelo objeto que podem ser lidos e ter seu valor modificado pelo cliente
  - Operações: serviços que podem ser requisitados pelos clientes de um objeto, que possuem:
    - Parâmetros: dados passados pelo cliente para a implementação do objeto ao chamar uma operação
    - Resultado: dado retornado pela operação
    - Exceções: retornadas quando detectada uma condição anormal na execução de uma operação
    - Contextos: carregam informação capaz de afetar a execução de uma operação

46

## CORBA

- Arquitetura do ORB



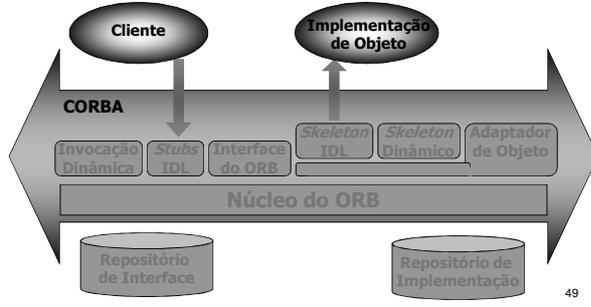
## CORBA

- Invocação de Operações Remotas
  - Formas de invocação:
    - Estática: através do código gerado com base na descrição da interface do servidor em IDL; ou
    - Dinâmica: através da interface de invocação dinâmica do CORBA
  - O servidor não percebe o tipo de invocação utilizado na requisição pelo cliente

48

## CORBA

### ■ Invocação Estática: *Stubs* e *Skeletons* IDL



49

## CORBA

### ■ *Stubs* IDL

- Geradas pelo compilador IDL com base na descrição da interface do objeto
- Usadas na invocação estática
- O cliente conhece a interface, o método e os parâmetros em tempo de compilação

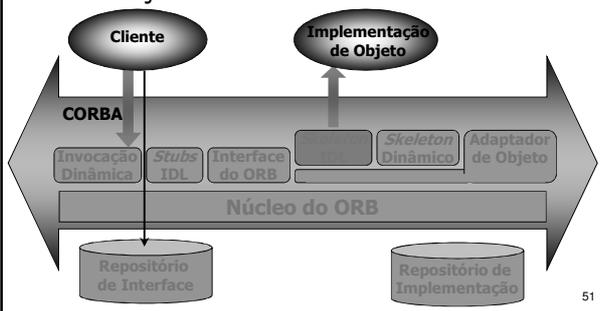
### ■ *Skeletons* IDL

- Geradas pelo compilador IDL
- Interface estática para os serviços (métodos) remotos executados pelo servidor

50

## CORBA

### ■ Invocação Dinâmica



51

## CORBA

### ■ Interface de Invocação Dinâmica (DII)

- Permite que o cliente construa uma invocação em tempo de execução
- Elimina a necessidade das *Stubs* IDL
- Com a DII, novos tipos de objetos podem ser adicionados ao sistema em tempo de execução
- O cliente especifica o objeto, o método e os parâmetros com uma seqüência de chamadas
- O servidor continua recebendo as requisições através de seu skeleton IDL

52

## CORBA

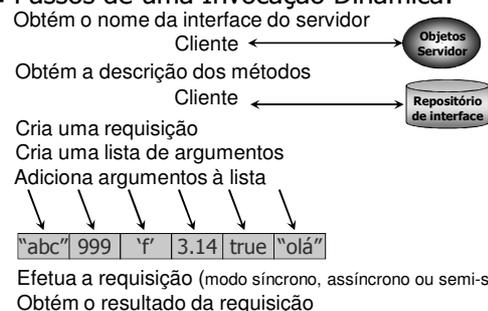
### ■ Repositório de Interface

- Contém informações a respeito das interfaces dos objetos gerenciados pelo ORB
- Permite que os serviços oferecidos pelo objeto sejam conhecidos dinamicamente por clientes
- Para usar a DII, a interface do objeto deve ser armazenada no repositório de interface

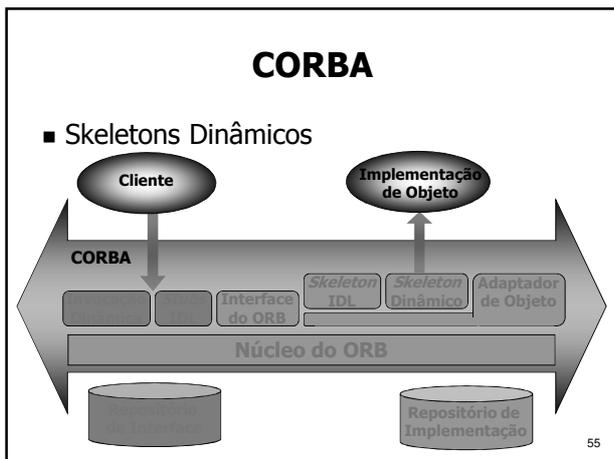
53

## CORBA

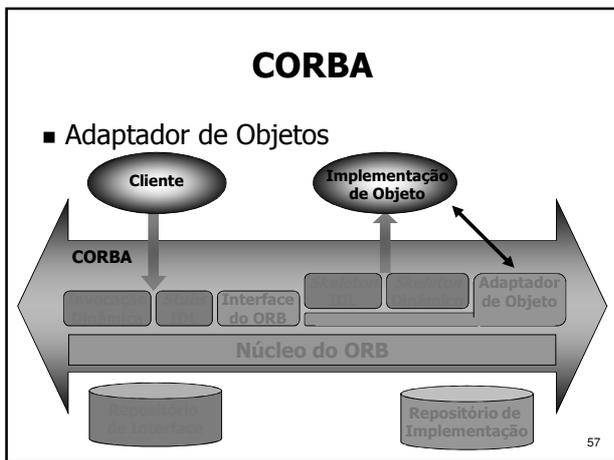
### ■ Passos de uma Invocação Dinâmica:



54

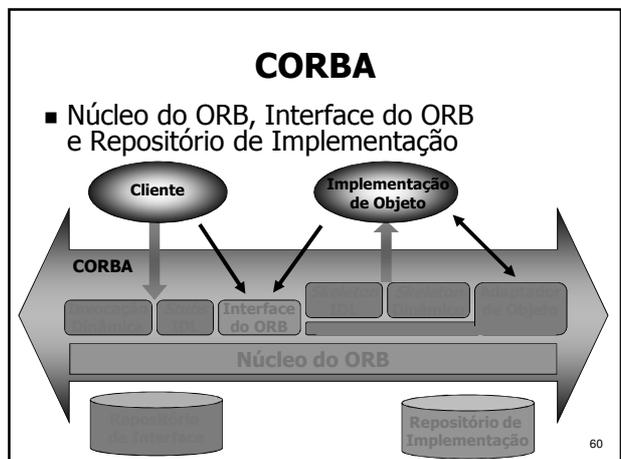


- ### CORBA
- *Skeletons* Dinâmicos
- Substituem os *Skeletons* IDL na ativação do objeto
  - Usados para manipular invocações de operações para as quais o servidor não possui *Skeletons* IDL
  - Fornece um mecanismo de ligação (*binding*) em tempo de execução
  - Uso: implementar pontes entre ORBs
- 56



- ### CORBA
- **Adaptador de Objetos**
- Interface entre o suporte e os objetos servidores
  - Transforma um objeto escrito em uma linguagem qualquer em um objeto CORBA
  - Usado para geração e interpretação de referências de objetos, invocação dos *Skeletons*, ativação e desativação de implementações de objetos, etc.
  - Existem vários tipos de adaptador de objeto
- 58

- ### CORBA
- *Portable Object Adapter* (POA)
- Adaptador padrão: torna o servidor portátil entre implementações diferentes
  - Abstrai a identidade do objeto da sua implementação
  - Implementa políticas de gerenciamento de *threads*:
    - uma *thread* por objeto
    - uma *thread* por requisição
    - grupo (*pool*) de *threads*
    - *etc.*
- 59



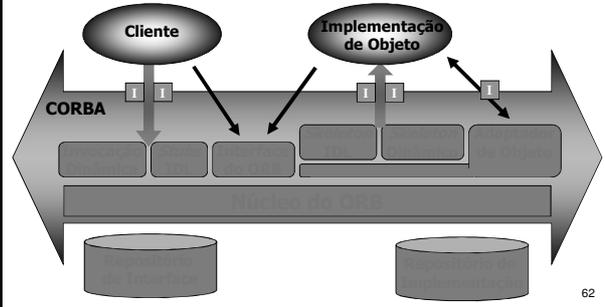
## CORBA

- Núcleo do ORB
  - Implementa os serviços básicos de comunicação
  - Utilizado pelos demais componentes do ORB
- Interface do ORB
  - Fornece serviços locais de propósito geral
  - Usado tanto pelo cliente quanto pelo servidor
- Repositório de Implementação
  - Contém informações para o ORB localizar e ativar as implementações de objetos

61

## CORBA

- Interceptadores



62

## CORBA

- Interceptadores
  - Dispositivos interpostos no caminho de invocação, entre Cliente e Servidor
  - Permitem executar código adicional para gerenciamento/controle/segurança, etc.
  - Há cinco pontos possíveis de interceptação
    - Dois pontos de interceptação no cliente: ao enviar a chamada e ao receber a resposta
    - Dois pontos de interceptação no servidor: ao receber a chamada e ao enviar a resposta
    - Um ponto de interceptação no POA: após a criação da referência do objeto (IOR)

63

## CORBA

- Interoperabilidade
  - CORBA garante a interoperabilidade entre objetos que usem diferentes implementações de ORB
  - Solução adotada a partir do CORBA 2.0
    - Padronizar o protocolo de comunicação e o formato das mensagens trocadas
    - Foi definido um protocolo geral, que é especializado para vários ambientes específicos

64

## CORBA

- Interoperabilidade (cont.)
  - Protocolo Inter-ORB Geral (GIOP)
    - Especifica um conjunto de mensagens e dados para a comunicação entre ORBs
  - Especializações do GIOP
    - Protocolo Inter-ORB para Internet (IIOP): especifica como mensagens GIOP são transmitidas numa rede TCP/IP
    - Protocolos Inter-ORB para Ambientes Específicos: permitem a interoperabilidade do ORB com outros ambientes (ex.: DCE, ATM nativo, etc.)

65

## CORBA

- Interoperabilidade entre CORBA e Java RMI
  - Une as vantagens das duas tecnologias
  - Applets, Servlets e aplicações Java podem ser clientes CORBA usando RMI/IIOP ou ORB Java
  - Mapeamentos: IDL → Java e Java → IDL
- Interoperabilidade entre CORBA e DCOM
  - Permite que objetos DCOM acessem serviços oferecidos por objetos CORBA e vice-versa
  - Bridges convertem mensagens entre os ambientes, integrando o DCOM a plataformas nas quais ele não está disponível

66

## CORBA

- Padrões Relacionados
  - CCM: modelo de componentes CORBA
  - CORBA AV *streams*: para fluxos de áudio/vídeo
  - *Minimum* CORBA: para sistemas embarcados
  - RT CORBA: para tempo-real
  - FT CORBA: para tolerância a falhas
  - CORBAMSec: serviço de segurança
  - CORBA Messaging: para troca de mensagens
  - AMI: para invocação assíncrona de métodos
  - Mapeamento de UML para IDL

67

## CORBA

- Padrões Relacionados (cont.)
  - *Model-Driven Architecture* (MDA)
  - *Unified Modeling Language* (UML)
  - *Common Warehouse Metamodel* (CWM)
  - *XML Metadata Interchange* (XMI)
- Em fase de padronização:
  - Integração de negócios, finanças, manufatura, ...
  - Integração com Web Services e .NET
  - Suporte para agentes móveis
  - Suporte para redes sem fio
  - ... e dezenas de outras especificações.

68

## Serviços CORBA

- Serviços CORBA
  - Coleção de serviços em nível de sistema
  - Oferecem funcionalidades básicas para utilizar e implementar os objetos de aplicações distribuídas
  - Especificam as interfaces e casos de uso, deixando a implementação de lado
  - Estendem ou complementam as funcionalidades do ORB
  - Independentes da aplicação

69

## Serviços CORBA



70

## Serviços CORBA

- Serviço de Nomes (*Naming Service*)
  - Define as interfaces necessárias para mapear um nome com uma referência de objeto
  - O objeto que implementa o serviço de nomes mantém a base de dados com o mapeamento entre referências e nomes
  - Uma referência para este serviço é obtida através do método:  
`resolve_initial_references("NameService")`
  - A referência do serviço de nomes é mantida pelo ORB ou em um servidor de diretório, http, ftp, etc.

71

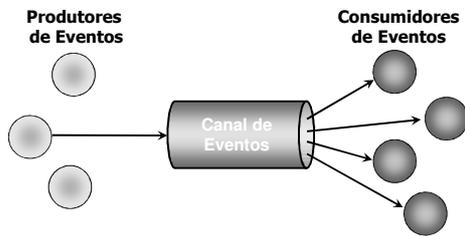
## Serviços CORBA

- Serviço de Eventos (*Event Service*)
  - Define como objetos podem registrar seu interesse dinamicamente em eventos enviados por outros objetos
  - Tipos de clientes do serviço
    - Produtores (ou *publishers*) de eventos
    - Consumidores (ou *subscribers*) de eventos
  - Canais de eventos (*Event Channels*) coletam e distribuem eventos entre objetos da aplicação
  - Três abordagens: *Push*, *Pull* e Mista

72

## Serviços CORBA

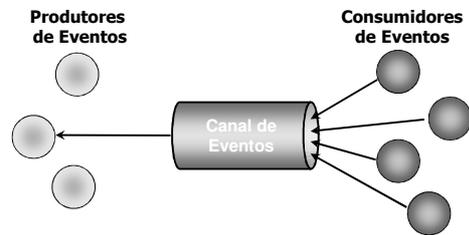
### ■ Serviço de Eventos – Abordagem *Push*



73

## Serviços CORBA

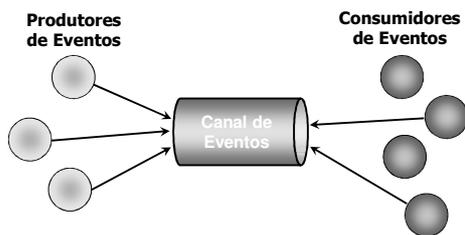
### ■ Serviço de Eventos – Abordagem *Pull*



74

## Serviços CORBA

### ■ Serviço de Eventos – Abordagem *Mista*



75

## Serviços CORBA

### ■ Serviço de Eventos – Vantagens

- Desacoplamento da comunicação
  - Um produtor não precisa saber quantos ou quem são os consumidores
  - Um consumidor não precisa saber quantos ou quem são os produtores
- Produtores e consumidores podem se conectar facilmente a vários canais de eventos

76

## Serviços CORBA

### ■ Serviço de Notificação (*Notification Service*)

- Extensão do serviço de eventos que oferece:
  - Qualidade de serviço: prioridade, prazo de validade dos eventos, ...
  - Persistência: armazena evento em um BD
  - Filtragem de eventos: limita os eventos recebidos usando um padrão
  - Eventos estruturados: possuem propriedades e dados para filtragem
  - Suporte a Multicast: reduz tráfego na rede

77

## Serviços CORBA

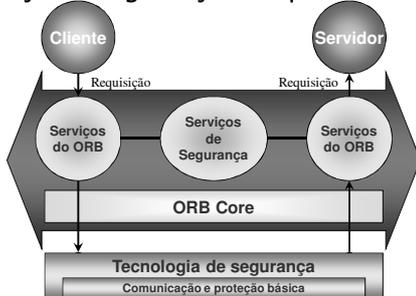
### ■ Serviço de Segurança (*Security Service*)

- Fornece serviços de segurança para objetos de aplicações CORBA
- Define a infraestrutura para garantir
  - Identificação e Autenticação
  - Autorização e Controle de Acesso
  - Auditoria
  - Segurança na Comunicação
  - Não Repúdio
- Independente da Tecnologia de Segurança usada: SSL, Kerberos, CSI-ECMA, etc.

78

## Serviços CORBA

### ■ Serviço de Segurança – Arquitetura



79

## Serviços CORBA

### ■ Serviço de Tempo (Time Service)

- Usado para ordenar, gerar ou computar intervalo entre eventos (temporização e alarmes) no sistema
- Baseado no padrão UTC (*Universal Time Coordinated*): fornece uma estimativa do erro de transmissão
- Pode ser usado para sincronização de relógios em ambiente heterogêneo



80

## Facilidades CORBA

- Facilidades CORBA
  - Coleções de serviços de propósito geral utilizados por aplicações distribuídas
- Facilidades Horizontais
  - São utilizadas por várias aplicações, independente da área da aplicação
  - São divididas segundo quatro categorias
    - Interface do usuário
    - Gerenciamento de informação
    - Gerenciamento de sistema
    - Gerenciamento de tarefa

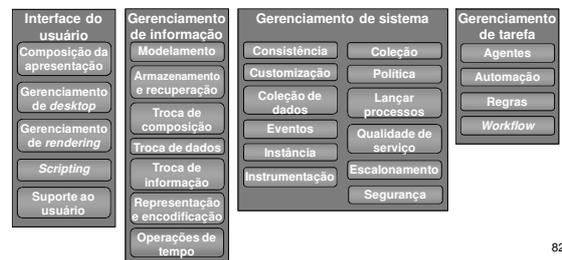
81

## Facilidades CORBA

### Aplicações Distribuídas



### Facilidades CORBA Horizontais



82

## Facilidades CORBA

- Facilidades Verticais
  - São utilizadas em áreas de aplicação específicas
  - Exemplos:
    - Processamento de Imagens
    - Supervias de informação
    - Manufatura integrada por computador
    - Simulação distribuída
    - Contabilidade
    - ...

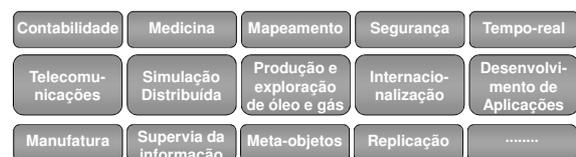
83

## Facilidades CORBA

### Aplicações Distribuídas



### Facilidades CORBA Verticais



84

## CORBA IDL

- IDL (Linguagem de Definição de Interface)
  - Usada para descrever as interfaces dos objetos CORBA
  - É uma linguagem declarativa, sem estruturas algorítmicas, que permite somente descrever tipos de dados, constantes e operações de um objeto CORBA
  - Uma interface descrita em IDL (arquivo .idl) especifica as operações providas pelo objeto e os parâmetros de cada operação

85

## CORBA IDL

- IDL (cont.)
  - De posse da IDL de um objeto, o cliente possui toda a informação necessária para utilizar os serviços deste objeto
  - Interfaces definidas em IDL podem ser acessadas através de *stubs* ou da interface de invocação dinâmica (DII)
  - As regras léxicas da IDL são iguais às do C++
  - As regras gramaticais da IDL são um subconjunto das regras do C++, acrescidas de construções para a declaração de operações

86

## CORBA IDL

- Tokens
  - Literais: 1, 2.37, 'a', "string", ...
  - Operadores: +, -, \*, =, ...
  - Separadores
    - Espaços
    - Tabulações
    - Quebras de linha
    - Comentários: // ou /\* \*/
  - Palavras-chave
  - Identificadores

87

## CORBA IDL

<u>Escopo</u> module interface abstract local <u>Definição de Tipos</u> const exception native typedef valuetype supports truncatable factory custom private public	<u>Tipos Básicos</u> any boolean char double fixed float long Object octet short string unsigned ValueBase void wchar wstring	<u>Tipos Construídos</u> enum sequence struct union switch case default <u>Dados e Operações</u> attribute readonly oneway in out inout context raises
---	---	--

88

## CORBA IDL

- Identificadores
  - São seqüências de caracteres do alfabeto, dígitos e *underscores* '\_'
  - O primeiro caractere deve ser uma letra
  - Todos os caracteres são significativos
  - Um identificador deve ser escrito exatamente como declarado, atentando para a diferença entre letra maiúsculas e minúsculas
  - Identificadores diferenciados apenas pelo *case*, como *MyIdent* e *myident*, causam erros de compilação

89

## CORBA IDL

- Elementos de uma especificação IDL
  - Módulos
  - Interfaces
  - Tipos de dados
  - Constantes
  - Exceções
  - Atributos
  - Operações
    - Parâmetros
    - Contextos

90

## CORBA IDL

- Módulos
  - Declaração de módulo:

```
module ident {  
    // lista de definições  
};
```
  - Pode conter declarações de tipos, constantes, exceções, interfaces ou outros módulos
  - O operador de escopo '::' pode ser usado para se referir a elementos com um mesmo nome em módulos diferentes

91

## CORBA IDL

- Interface
  - Declaração de interface:

```
interface ident : interfaces_herdadas {  
    // declarações de tipos  
    // declarações de constantes  
    // declarações de exceções  
    // declarações de atributos  
    // declarações de operações  
};
```
  - Pode conter declarações de tipos, constantes, exceções, atributos e operações

92

## CORBA IDL

- Interfaces Abstratas
  - Não podem ser instanciadas, servindo somente como base para outras interfaces

```
abstract interface ident { ... };
```
- Interfaces Locais
  - Não são acessíveis pela rede, recebendo somente chamadas locais

```
local interface ident { ... };
```

93

## CORBA IDL

- Herança de Interfaces
  - Os elementos herdados por uma interface podem ser acessados como se fossem elementos declarados explicitamente, a não ser que o identificador seja redefinido ou usados em mais de uma interface base
  - O operador de escopo '::' deve ser utilizado para referir-se a elementos das interfaces base que foram redefinidos ou que são usados em mais de uma interface base

94

## CORBA IDL

- Herança de Interfaces (cont.)
  - Uma interface pode herdar bases indiretamente, pois interfaces herdadas possuem suas próprias relações de herança
  - Uma interface não pode aparecer mais de uma vez na declaração de herança de uma outra interface, mas múltiplas ocorrências como base indireta são aceitas

95

## CORBA IDL

- Exemplo: Servidor de um Banco

```
module banco {  
    // ...  
    interface auto_atendimento {  
        // ...  
    };  
    interface caixa_eletronico: auto_atendimento {  
        // ...  
    };  
};
```

96

## CORBA IDL

- Tipos e Constantes
  - Novos nomes podem ser associados a tipos já existentes com a palavra-chave typedef  
typedef tipo ident;
  - Objetos descritos como valuetype podem ser enviados como parâmetros de chamadas  
valuetype ident { ... };

97

## CORBA IDL

- Constantes
  - Definidas com a seguinte sintaxe:  
const tipo ident = valor;
  - Operações aritméticas (+, -, \*, /, ...) e binárias (|, &, <<, ...) entre literais e constantes podem ser usadas para definir o valor de uma constante

98

## CORBA IDL

- Tipos Básicos
  - boolean: tipo booleano, valor TRUE ou FALSE
  - char: caractere de 8 bits, padrão ISO Latin-1
  - short: inteiro curto com sinal;  $-2^{15}$  a  $2^{15}-1$
  - long: inteiro longo com sinal;  $-2^{31}$  a  $2^{31}-1$
  - unsigned short: inteiro curto sem sinal; 0 a  $2^{16}-1$
  - unsigned long: inteiro longo sem sinal; 0 a  $2^{32}-1$
  - float: real curto, padrão IEEE 754/1985
  - double: real longo, padrão IEEE 754/1985
  - octet: 1 byte, nunca convertido na transmissão
  - any: corresponde a qualquer tipo IDL

99

## CORBA IDL

- Tipos Básicos (cont.)
  - Object: corresponde a um objeto CORBA
  - long long: inteiro de 64 bits;  $-2^{63}$  a  $2^{63}-1$
  - unsigned long long: inteiro de 64 bits sem sinal; 0 a  $2^{64}-1$
  - long double: real duplo longo padrão IEEE; base com sinal de 64 bits e 15 bits de expoente
  - wchar: caractere de 2 bytes, para suportar diversos alfabetos
  - fixed<n,d>: real de precisão fixa; n algarismos significativos e d casas decimais

100

## CORBA IDL

- Arrays
  - Array de tamanho fixo:  
tipo ident[tamanho];
  - Array de tamanho variável sem limite de tamanho (tamanho efetivo definido em tempo de execução)  
sequence <tipo> ident;
  - Array de tamanho variável com tamanho máximo:  
sequence <tipo,tamanho> ident;

101

## CORBA IDL

- Strings
  - Seqüência de caracteres sem limite de tamanho:  
string ident; // seqüência de char's  
wstring ident; // seqüência de wchar's
  - Seqüência de caracteres com tamanho máximo:  
string <tamanho> ident;  
wstring <tamanho> ident;

102

## CORBA IDL

### ■ Exemplo: Servidor de um Banco

```
module banco {
  typedef unsigned long conta;
  typedef double valor;
  const string nome_banco = "UFSC";
  const string moeda = "R$";
  // ...
};
```

103

## CORBA IDL

### ■ Tipos Complexos

- Estrutura de dados (registro)
  - Tipo composto por vários campos

```
struct ident {
  // lista de campos (tipos IDL)
};
```
- Lista enumerada
  - Lista com valores de um tipo

```
enum ident { /*lista de valores*/ };
```

104

## CORBA IDL

### ■ Tipos Complexos (cont.)

- União discriminada
  - Tipo composto com seleção de campo por cláusula switch/case; o seletor deve ser tipo IDL inteiro, char, boolean ou enum

```
union ident switch (seletor){
  case valor: tipo ident;
  // mais campos
  default: tipo ident;
};
```

105

## CORBA IDL

### ■ Exemplo: Servidor de um Banco

```
module banco {
  // ...
  enum aplicacao { poupanca, CDB, renda_fixa };
  struct transacao {
    unsigned long data; // formato ddmmyyyy
    string<12> descricao;
    valor quantia;
  };
  sequence < transacao > transacoes;
  // ...
};
```

106

## CORBA IDL

### ■ Exceções

- São estruturas de dados retornadas por uma operação para indicar que uma situação anormal ocorreu durante sua execução
- Cada exceção possui um identificador e uma lista de membros que informam as condições nas quais a exceção ocorreu

```
exception ident {
  // lista de membros
};
```
- Exceções padrão do CORBA: CONCLUDED\_YES, CONCLUDED\_NO, CONCLUDED\_MAYBE

107

## CORBA IDL

### ■ Atributos

- São dados de um objeto que podem ter seu valor lido e/ou modificado remotamente
- Declarados usando a sintaxe:

```
attribute tipo ident;
```
- Caso a palavra-chave readonly seja utilizada, o valor do atributo pode ser somente lido

```
readonly attribute tipo ident;
```

108

## CORBA IDL

### ■ Exemplo: Servidor de um Banco

```
module banco {  
  // ...  
  exception conta_invalida { conta c; };  
  exception saldo_insuficiente { valor saldo; };  
  
  interface auto_atendimento {  
    readonly attribute string boas_vindas;  
    // ...  
  };  
  // ...  
};
```

109

## CORBA IDL

### ■ Operações

- Declaradas em IDL na forma:  
tipo ident (/\* lista de parâmetros \*/)  
[ raises ( /\* lista de exceções \*/ )  
[ context ( /\* lista de contextos \*/ ) ] ;
- Parâmetros
  - Seguem a forma: {in|out|inout} tipo ident
    - in: parâmetro de entrada
    - out: parâmetro de saída
    - inout: parâmetro de entrada e saída
  - Separados por vírgulas

110

## CORBA IDL

### ■ Contextos

- São strings que, ao serem passadas para o servidor em uma chamada, podem interferir de alguma forma na execução da operação
- Um asterisco, ao aparecer como o último caractere de um contexto, representa qualquer seqüência de zero ou mais caracteres

111

## CORBA IDL

### ■ Operações Oneway (assíncronas)

- Declaradas em IDL na forma:  
oneway void ident (/\* lista de parâmetros \*/);
- Uma operação oneway é assíncrona, ou seja, o cliente não aguarda seu término.
- Operações oneway não possuem retorno (o tipo retornado é sempre void) e as exceções possíveis são somente as padrão.

112

## CORBA IDL

### ■ Exemplo: Servidor de um Banco

```
interface auto_atendimento {  
  readonly attribute string boas_vindas;  
  valor saldo ( in conta c ) raises (conta_invalida);  
  void extrato ( in conta c, out transacoes t,  
    out valor saldo ) raises (conta_invalida);  
  void transferencia ( in conta origem,  
    in conta destino, in valor v )  
    raises (conta_invalida, saldo_insuficiente);  
  void investimento ( in conta c,  
    in aplicacao apl, in valor v )  
    raises (conta_invalida, saldo_insuficiente);  
};
```

113

## CORBA IDL

### ■ Exemplo: Servidor de um Banco

```
interface caixa_eletronico : auto_atendimento {  
  void saque ( in conta c, in valor v )  
    raises ( conta_invalida, saldo_insuficiente );  
};
```

114

## CORBA IDL

- Mapeamento IDL para C++
  - Definido no documento OMG/99-07-41, disponível em <http://www.omg.org>
  - O mapeamento define a forma como são representados em C++ os tipos, interfaces, atributos e operações definidos em IDL

115

## CORBA IDL

- Mapeamento de Módulos IDL para C++
  - Módulos são mapeados em namespaces
  - Se o compilador não suportar namespaces, o módulo é mapeado como uma classe
- Mapeamento de Interfaces IDL para C++
  - Interfaces são mapeadas como classes C++
    - `Interface_var`: libera a memória automaticamente quando sai do escopo
    - `Interface_ptr`: não a libera memória

116

## CORBA IDL

Tipo IDL	Equivalente em C++
<code>boolean</code>	<code>CORBA::Boolean</code>
<code>char</code>	<code>CORBA::Char</code>
<code>wchar</code>	<code>CORBA::WChar</code>
<code>short</code>	<code>CORBA::Short</code>
<code>long</code>	<code>CORBA::Long</code>
<code>long long</code>	<code>CORBA::LongLong</code>
<code>unsigned short</code>	<code>CORBA::Ushort</code>
<code>unsigned long</code>	<code>CORBA::Ulong</code>
<code>unsigned long long</code>	<code>CORBA::ULongLong</code>
<code>float</code>	<code>CORBA::Float</code>
<code>double</code>	<code>CORBA::Double</code>
<code>long double</code>	<code>CORBA::LongDouble</code>
<code>octet</code>	<code>CORBA::Octet</code>
<code>any</code>	<code>CORBA::Any</code> (classe)
<code>fixed</code>	<code>CORBA::Fixed</code> (classe)
<code>Object</code>	<code>CORBA::Object</code> (classe)

## CORBA IDL

- Mapeamento de Tipos IDL para C++
  - São idênticos em C++ e IDL, e portanto não precisam de mapeamento:
    - Constantes
    - Estruturas de dados
    - Listas enumeradas
    - Arrays
  - Unions IDL são mapeadas como classes C++, pois o tipo union de C++ não possui seletor
  - Sequências são mapeadas em classes C++
  - Strings são mapeadas como `char *` e `Wchar *`<sup>118</sup>

## CORBA IDL

- Mapeamento de Atributos IDL para C++
  - Um método com o mesmo nome do atributo retorna o seu valor
  - Se o atributo não for somente de leitura, um método de mesmo nome permite modificar o seu valor
- Mapeamento de Exceções IDL para C++
  - São mapeadas como classes C++

119

## CORBA IDL

- Mapeamento de Operações IDL para C++
  - Operações de interfaces IDL são mapeadas como métodos da classe C++ correspondente
  - Contextos são mapeados como um parâmetro implícito no final da lista de parâmetros (classe `Context_ptr`)
  - Se o compilador não suportar exceções, outro parâmetro implícito é criado ao final da lista de parâmetros (classe `Exception`)
  - Os parâmetros implícitos têm valores *default* nulos, permitindo que a operação seja chamada sem especificar estes parâmetros

120

## CORBA IDL

Data Type	In	Inout	Out	Return
short	Short	Short	Short	Short
long	Long	Long	Long	Long
long long	LongLong	LongLong	LongLong	LongLong
unsigned short	UShort	UShort	UShort	UShort
unsigned long	ULong	ULong	ULong	ULong
unsigned long long	ULongLong	ULongLong	ULongLong	ULongLong
float	Float	Float	Float	Float
double	Double	Double	Double	Double
long double	LongDouble	LongDouble	LongDouble	LongDouble
boolean	Boolean	Boolean	Boolean	Boolean
char	Char	Char	Char	Char
wchar	WChar	WChar	WChar	WChar
octet	Octet	Octet	Octet	Octet
enum	enum	enum	enum	enum
object	objref_ptr	objref_ptr	objref_ptr	objref_ptr
struct, fixed const	struct	struct	struct	struct
struct, variable const	struct	struct	struct	struct
union, fixed const	union	union	union	union
union, variable const	union	union	union	union
string const	char*	char*	char*	char*
wstring const	WChar*	WChar*	WChar*	WChar*
sequence const	sequence	sequence	sequence	sequence
array, fixed const	array	array	array	array slice
array, variable const	array	array	array slice	array slice
any const	any	any	any	any
fixed const	fixed	fixed	fixed	fixed

121

## CORBA IDL

- Mapeamento IDL para Java
  - Definido pelo documento formal/01-06-06, disponível em <http://www.omg.org/>
  - O mapeamento define a forma como são representados em Java os tipos, interfaces, atributos e operações definidos em IDL

122

## CORBA IDL

- Mapeamento de IDL para Java
  - Módulos são mapeados em packages Java
  - Interfaces, Exceções e Arrays e Strings são idênticos em Java
  - Sequências são mapeadas como Arrays Java
  - Constantes são mapeadas para atributos estáticos
  - Estruturas de dados, Unions e Enums são mapeadas como classes Java

123

## CORBA IDL

Tipo IDL	Equivalente em Java
boolean	boolean
char	char
wchar	char
short	short
long	int
long long	long
unsigned short	short
unsigned long	int
unsigned long long	long
float	float
double	double
long double	(não disponível)
octet	byte
any	CORBA.Any
fixed	Math.BigDecimal
Object	CORBA.Object

124

## CORBA IDL

- Mapeamento de Atributos IDL para Java
  - É criado um método com o nome do atributo
  - Se o atributo não for readonly, um método de mesmo nome permite modificar o seu valor
- Mapeamento de Operações IDL para Java
  - São criados métodos na interface correspondente, com os mesmos parâmetros e exceções
  - Contexto inserido no final da lista de parâmetros

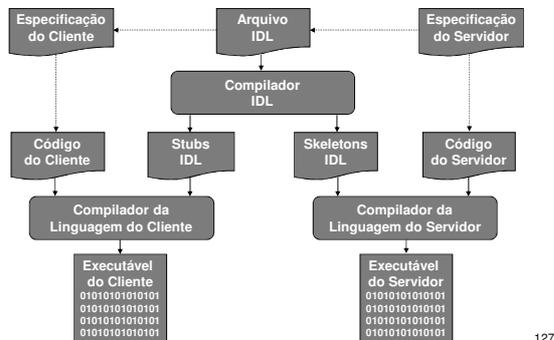
125

## Desenvolvimento de Aplicações

- Passos para desenvolver um servidor CORBA
  - Definir a interface IDL do servidor
  - Compilar a IDL para gerar o *skeleton*
  - Implementar os métodos do servidor
  - Compilar
  - Executar
- Passos para desenvolver um cliente CORBA
  - Compilar a IDL do servidor para gerar a *stub*
  - Implementar o código do cliente
  - Compilar
  - Executar

126

## Desenvolvimento de Aplicações



127

## Desenvolvimento de Aplicações

- O código pode ser implementado em qualquer linguagem mapeada para IDL

```
public class AutoAtendimentoImpl
    extends AutoAtendimentoPOA {
    public String boas_vindas() {
        return "Bem-vindo ao Banco";
    }
    ...
};
```

Java

```
class auto_atendimentoImpl:
    auto_atendimentoPOA { ... };
char* banco auto_atendimentoImpl::boas_vindas()
    throws (CORBA::SystemException) {
    return CORBA::string_dup("Bem-vindo ao Banco");
};
```

C++ 128

## Desenvolvimento de Aplicações

- Implementação do Servidor
  - O servidor deve iniciar o ORB e o POA, e disponibilizar sua referência para os clientes
  - Referências podem ser disponibilizadas através do serviço de nomes, impressas na tela ou escritas em um arquivo acessado pelos clientes usando o sistema de arquivos distribuído, um servidor HTTP ou FTP
  - Feito isso, o servidor deve ficar ouvindo requisições e as executando

129

## Desenvolvimento de Aplicações

- Implementação do Servidor
 

```
package banco;
import org.omg.*;
import java.io.*;

public class servidor
{
    public static void main(String args[]) {
        try{
            // Cria e inicializa o ORB
            ORB orb = ORB.init(args, null);

            // Cria a implementação e registra no ORB
            auto_atendimentoImpl impl = new
            auto_atendimentoImpl();
```

130

## Desenvolvimento de Aplicações

```
// Ativa o POA
POA rootpoa = POAHelper.narrow(
orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();

// Pega a referência do servidor
org.omg.CORBA.Object ref =
    rootpoa.servant_to_reference(impl);
auto_atendimento href =
    auto_atendimentoHelper.narrow(ref);

// Obtém uma referência para o serv. de nomes
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
NamingContextExt ncRef =
    NamingContextExtHelper.narrow(objRef);
```

131

## Desenvolvimento de Aplicações

```
// Registra o servidor no serviço de nomes
String name = "AutoAtendimento";
NameComponent path[] = ncRef.to_name( name );
ncRef.rebind(path, href);

System.out.println("Servidor em execução");

// Aguarda chamadas dos clientes
orb.run();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

132

## Desenvolvimento de Aplicações

```
package banco;
import org.omg.*;
public class auto_atendimentoImpl
    extends auto_atendimentoPOA {
    public String boas_vindas () {
        return "Bem-vindo ao banco " + banco.nome_banco.value;
    }
    public double saldo (int c) throws conta_invalida {
        return CadastroBanco.getConta(c).getSaldo();
    }
    // ...
}
```

133

## Desenvolvimento de Aplicações

- Implementação do Cliente
  - Um cliente deve sempre iniciar o ORB e obter uma referência para o objeto servidor
  - Referências podem ser obtidas através do serviço de nomes, da linha de comando ou lendo um arquivo que contenha a referência
  - De posse da referência, o cliente pode chamar os métodos implementados pelo servidor

134

## Desenvolvimento de Aplicações

### ■ Implementação do Cliente

```
import banco.*;
import org.omg.*;
import java.io.*;
public class cliente {
    public static void main(String args[]) {
        try {
            // Cria e inicializa o ORB
            ORB orb = ORB.init(args, null);
```

135

## Desenvolvimento de Aplicações

```
// Obtém referência para o serviço de nomes
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);

// Obtém referência para o servidor
auto_atendimento server =
auto_atendimentoHelper.narrow(
ncRef.resolve_str("AutoAtendimento"));

// Imprime mensagem de boas-vindas
System.out.println(server.boas_vindas());
```

136

## Desenvolvimento de Aplicações

```
// Obtém o número da conta
System.out.print("Entre o número da sua conta: ");
String conta = new BufferedReader(new
InputStreamReader(System.in)).readLine();

// Imprime o saldo atual
System.out.println("Seu saldo é de R$" +
server.saldo(Integer.parseInt(conta)));
} catch (Exception e) {
    e.printStackTrace(System.out);
}
}
```

137