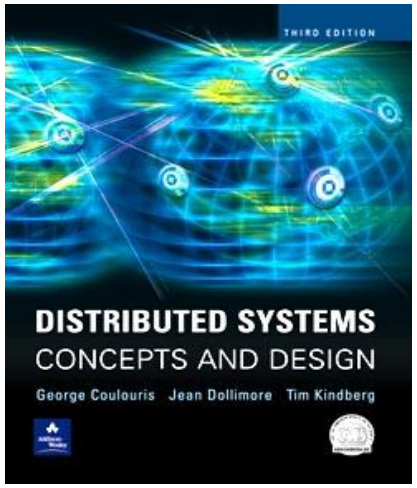
Capítulo 5: Objetos Distribuídos e Invocações Remotas



From **Coulouris, Dollimore and Kindberg**

Distributed Systems: Concepts and Design

Edition 3, © Addison-Wesley

Objetos Distribuídos e Invocação Remota

- Tópicos cobertos nesse capítulo
  - Comunicação entre objetos distribuídos
  - Invocação remota de procedimento
  - Eventos e notificações
  - Java RMI
- Problemas com objetos distribuídos?
  - Como identificar objetos
  - O que está envolvido na invocação de um método de uma classe
    - Quais métodos estão disponíveis ?
    - Como podemos passar e receber valores ?
  - É possível rastrear eventos em sistemas distribuídos ?

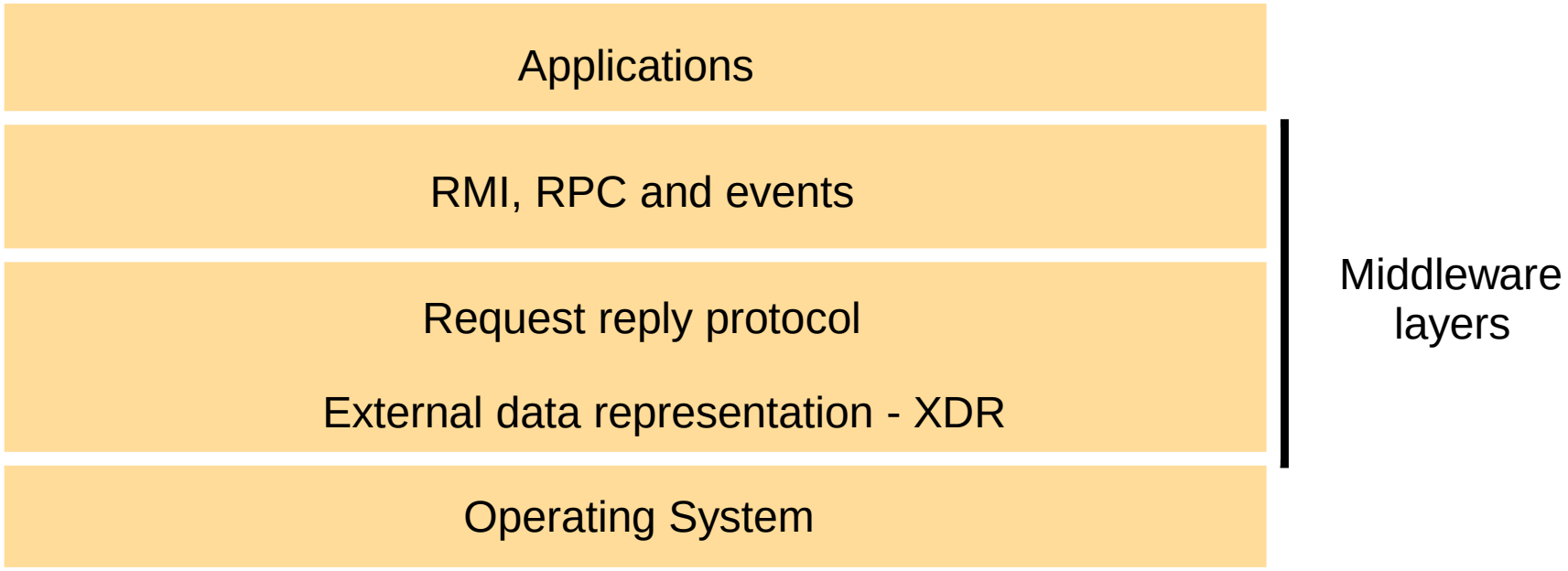
Modelos de Programação para Sistemas Distribuídos

- Remote Procedure Call – cliente chama procedimentos que executam em outro processo
- Remote Method Invocation (RMI) – um objeto em um processo invoca um método em um objeto em outro processo
- Notificação de eventos – objetos recebem a notificação de eventos de outros objetos onde eles se registraram
- Os mecanismos tem que ser transparentes em relação à localização.

Papel do Middleware

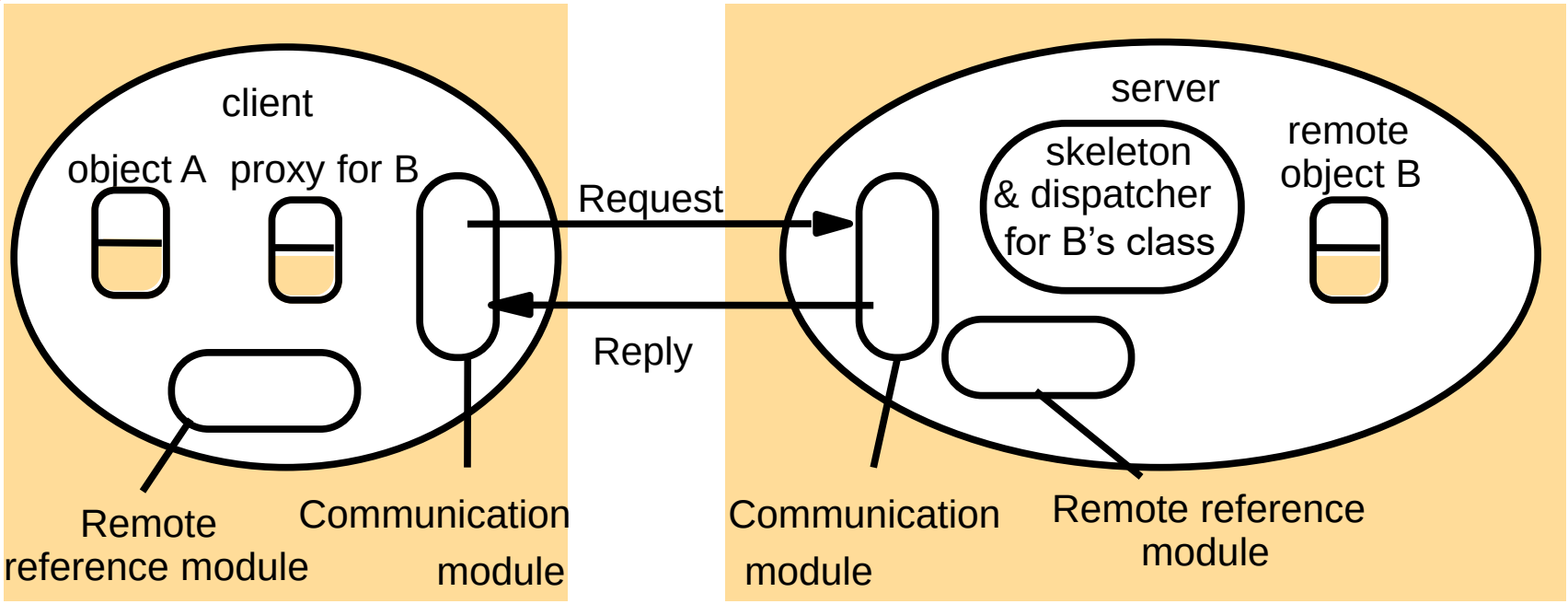
- Papel do Middleware
  - Prover abstrações de alto nível como RMI
  - Permitir transparência de localização
  - Ser livre de aspectos específicos como
    - Protocolos de comunicação
    - Sistemas Operacionais e hardwares de comunicação
  - Garantir interoperabilidade

Figure 5.1
Middleware layers



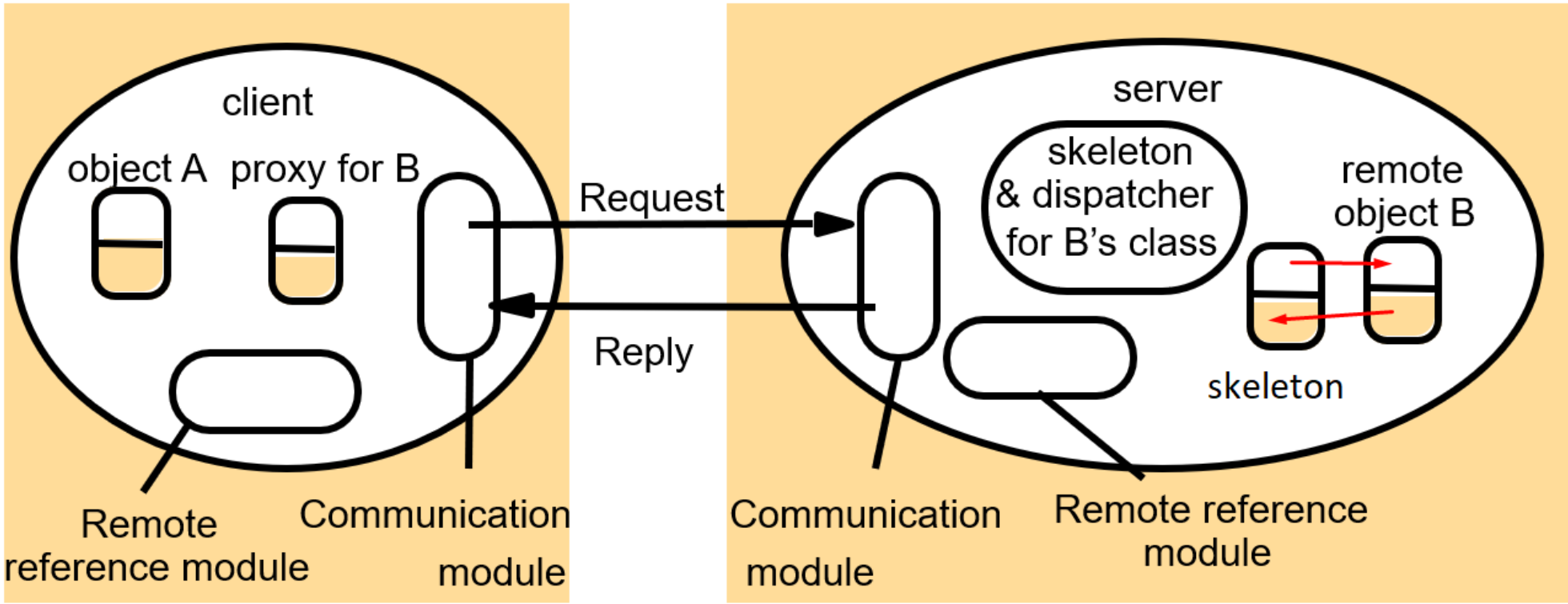
Arquitetura RMI

Figure 5.6



Arquitetura RMI

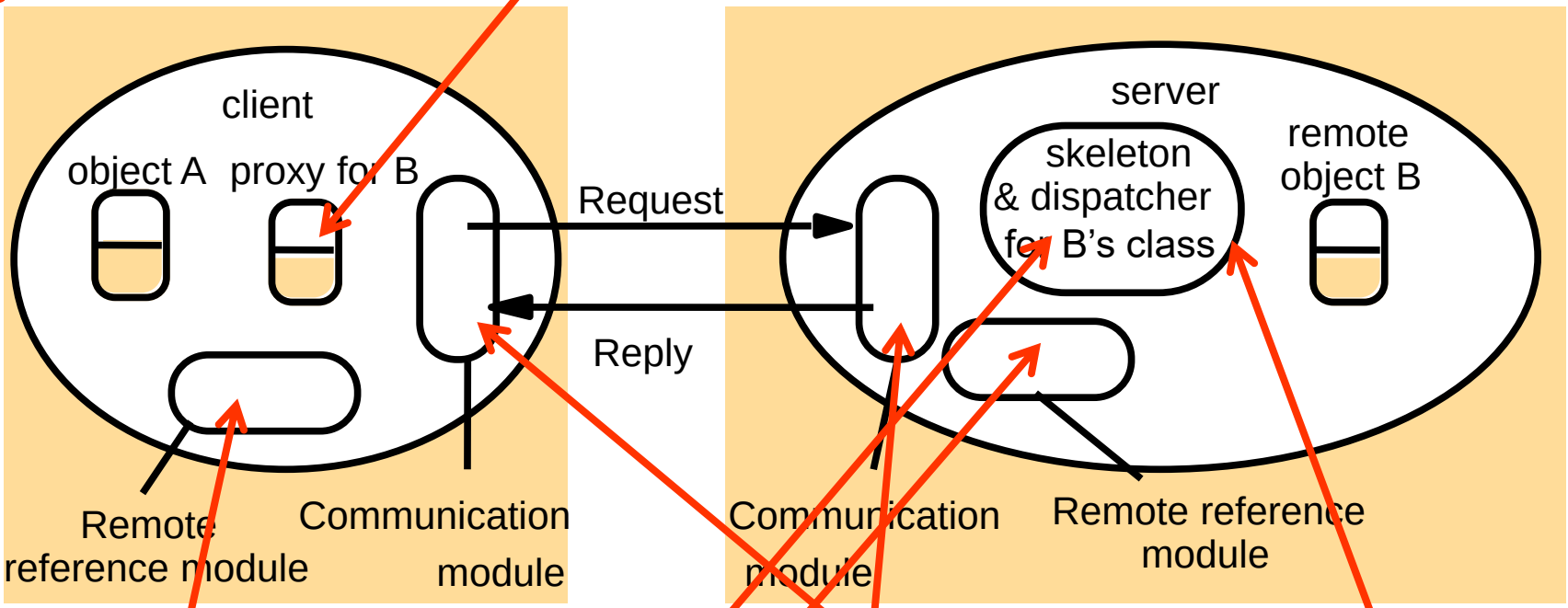
Destacando o Skeleton



Arquitetura RMI

Proxy – torna RMI transparente para cliente. Classe implementa interface remota. Converte requisições e desconverte respostas.

Figure 5.6



Tradução de referências de objetos remotas ou locais

Carrega protocolo de requisição-resposta

Dispatcher – pega requisição do módulo de comunicação e invoca método no skeleton (usando *methodID* na mensagem).

Skeleton – implementa métodos da interface remota. Desconverte requisições e converte resultados. Invoca método no objeto remoto.

Software RMI

- **Proxy** – provê transparência para invocação remota
  - Converte e desconverte argumentos
  - Envia e recebe mensagens
- **Dispatcher** – transfere requisições ao objeto correto
  - Recebe requisição, seleciona objeto correto e transfere mensagem
- **Skeleton** – implementa metodos da interface remota
  - Desconverte argumentos, invoca método do objeto remoto e converte resultados

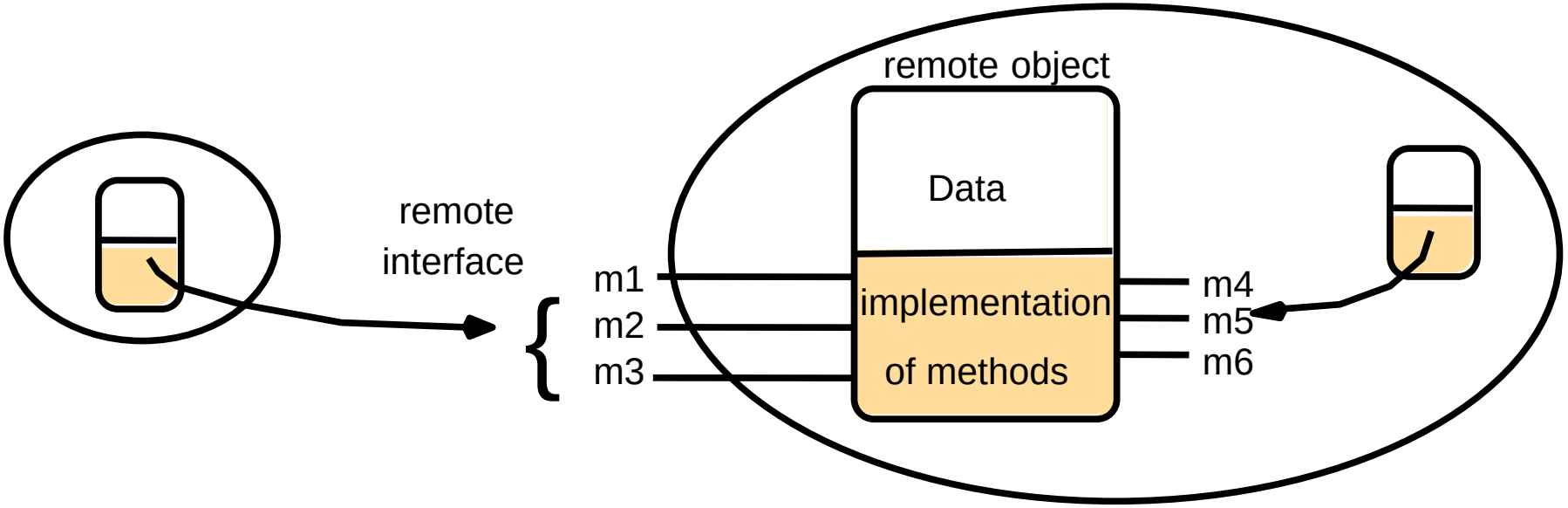
Software RMI

- **Servidor**
  - Classes para Dispatchers, skeletons e objetos remotos
  - Inicialização cria objetos remotos
  - Registro de objetos no “Binder”
- **Cliente**
  - classes para proxies de todos os objetos remotos
  - Binder procura por referências remotas de objetos
  - Não se cria objetos remotos através dos construtores
  - É preciso utilizar padrões de projeto criacionais (“fábricas”)

Software RMI

- **Modulo de Referência Remota**
  - Responsabilidades
    - Tradução entre referências locais e remotas
      - Tabela de objetos remotos
        - uma linha para cada objeto remoto do processo
        - uma linha para cada proxy
    - Criação de referências remotas

Figure 5.4
Objeto Remoto e sua interface remota



Interfaces

- Interface – esqueleto de classe pública (skeleton)
  - Define como será a interação
  - Abstração útil que remove dependências de aspectos internos
  - exemplos
    - Interface de Procedimentos
    - Interface de Classes: Remota
    - Interface de Módulo
- Interfaces de Sistemas Distribuídos
  - Qual é a maior diferença comparado com processo local?
    - Processos em diferentes nós
    - Pode passar apenas informações declaradas na interface

Interface de Procedimentos (RPC)

- Interface de Serviço
  - Especifíca um conjunto de procedimentos disponíveis para o cliente
  - Parâmetros de entrada e saída
  - Remote Procedure Call
    - argumentos são convertidos
    - Envia-se argumentos para o servidor
    - Servidor ‘desconverte’ pacote, executa procedimento, converte resposta e envia para o cliente
    - Cliente desconverte o valor de retorno
  - Todos estes detalhes são transparentes

Interface Remota (RMI)

- Interface Remota
  - Especifica métodos disponíveis em um objeto para invocação remota
  - Parâmetros de entrada e saída podem ser objetos
    - Bom uso para Serialização em Java
  - Remote Method Invocation
    - Funciona da mesma forma que a RPC
    - Deve haver um formato comum sobre como passar objetos(e.x., CORBA IDL or Java RMI)

Interface Definition Language (IDL)

- Interface Definition Language
  - Notação para interfaces independentes de linguagem
    - especifica tipo e seqüência dos argumentos
  - Exemplos
    - CORBA IDL for RMI
    - Sun XDR for RPC
    - Java Remote Interface e Serialization
    - DCOM IDL
  - Compilador IDL permite interoperabilidade entre linguagens diferentes

Exemplo CORBA IDL

```
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

Figure 5.2

Estrutura CORBA

Interface remota

Métodos da interface remota (RMI)

Parâmetros de entrada ou/e saída

- Interface Remota:
  - Especifica quais métodos de um objeto estão disponíveis para invocação
  - Uma linguagem de definição de interfaces (or IDL) é usada. E.g. the above in CORBA IDL ou interfaces Java que extendem de java.rmi.Remote
  - Java RMI teria uma classe para Pessoa, mas CORBA tem uma estrutura

Modelo Orientado a Objetos

- Objeto encapsula dados e métodos.
- Objetos podem ser acessados por referências.
- Uma **interface** provê a definição das assinaturas dos métodos disponíveis.
- As ações são realizadas por invocações aos métodos.
  - Alguma entidade deve prover as funcionalidades da interface
  - O estado de quem provê a funcionalidade pode mudar

Modelo Orientado a Objetos

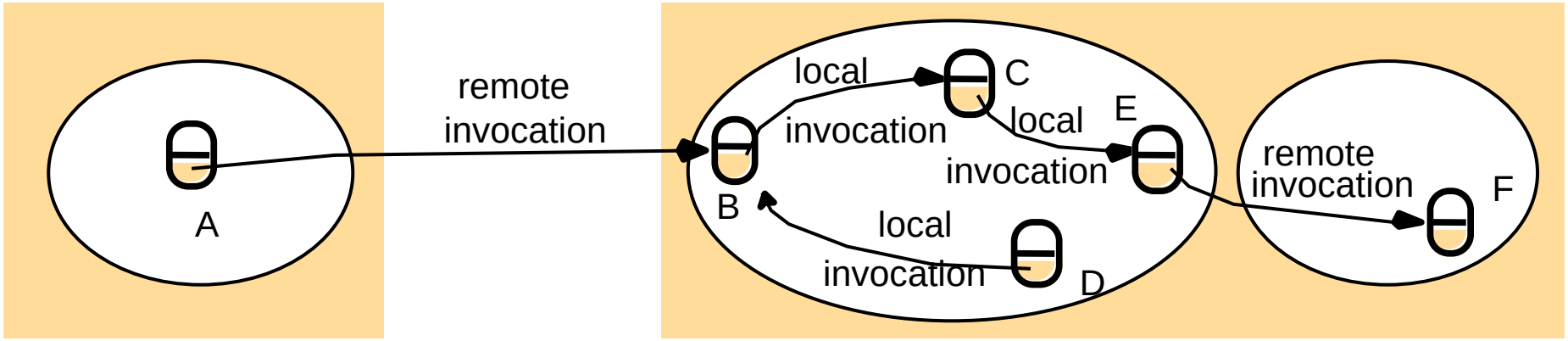
- **Exceções** podem ser lançadas para reportar erros.
- **Garbage collection** libera espaço ocupado por objetos não mais usados.

Modelo OO Distribuído

- RMI –Objetos em processos distintos.
  - É possível que haja LMI também (L == Local).
- Objeto Remoto – objetos podem receber invocações remotas.

Modelo de Objetos Distribuídos

Figure 5.3



- Cada processo contém objetos que podem receber invocações remotas. Outros apenas invocações locais
- É necessário que haja o conceito de referência de objeto remota para que um objeto use o outro remotamente. Como obter essas referências ?
- A interface remota especifica quais métodos podem ser invocados remotamente

Modelo OO Distribuído

- Referência Remota de Objeto
  - Permite que objetos em processos diferentes se usem
  - Podem ser passadas como argumentos em invocações de métodos
- Interface Remota
  - CORBA – usa IDL
  - JAVA – interface extende java.rmi.**Remote**

Problemas de Projeto com RMI

- Dois problemas em usar RMI ao invés de LMI
  - **Número de invocações**
    - O ideal é invocar poucas vezes devido ao custo de comunicação
  - **Transparência de Localização**
    - Uso deve permanecer simples como invocações locais

Propriedades Semânticas de Invocação

- **Semântica Incerta**:
  - O invocador não pode saber se o método ou procedimento remoto foi executado.
  - Só se falhar (Timeout?).
  - Tipos de falhas:
    - Falhas de Omissão - ex. Perda de pacotes
    - Falhas de Parada – Servidor trava
  - Portanto deve ser usado somente quando as falhas de invocação são aceitáveis
  - Exemplo: Protocolo HTTP

Propriedades Semânticas de Invocação

- **Semântica Ao-menos-um (mas pode chamar mais)**:
  - É garantido que o invocador/cliente receba um resultado ou uma exceção/erro.
  - Tipos de falhas:
    - Retransmitir requisição mascara falhas de omissão.
    - Ocorrem falhas de travamento quando o servidor do objeto remoto pára.
    - Falhas arbitrárias quando o método é invocado mais que uma vez
      - Uma **operação idempotente** pode ser executada repetidamente produzindo sempre o mesmo efeito pode ser a solução.
  - SUN RPC usa essa semântica

Propriedades Semânticas de Invocação

- **No-máximo-uma-vez**:
  - O invocador recebe um resultado ou uma exceção.
  - Falhas de omissão (rede?) podem ser eliminadas **retransmitindo** a requisição.
  - Falhas arbitrárias são evitadas pois um método é executado no máximo uma vez.
    - Viabiliza a invocação de operações não-idempotentes

- JAVA RMI e CORBA usam essa semântica
  - Isso é transparente para o desenvolvedor
- Exemplo: Uso de Tokens em uma aplicação WEB
  - Ao renderizar interface, gera-se um token
  - Ao submeter próxima request, envia token
  - O Token não pode ser consumido uma segunda vez

Técnicas vs Semânticas de Invocação

- **Técnicas para melhorar o protocolo requisição-resposta**
- **Torná-lo mais tolerante à falhas:**
  - **Mensagem de Retry**: retransmitir a requisição até que uma resposta chegue ou o servidor é dito fora-do-ar.
  - **Filtrar duplicatas**: quando retransmissões são usadas, ajuda a filtrar requisições recebidas em dobro pelo servidor.
  - **Retransmissão de resultados**: manter um histórico de mensagens enviadas, permitindo retransmissão (sem ter que executar operação no servidor novamente).
    - Maior complexidade

Técnicas vs Semânticas de Invocação

- A combinação dessas medidas levam à diferentes semânticas
- A figura 5.5 mostra as medidas com os nomes correspondentes para a semântica que elas produzem.

Técnicas vs Semânticas de Invocação

- Invocações locais são executadas 1 vez exatamente
- Invocações remotas podem não funcionar assim. :

Figure 5.5 *Medidas de tolerância a falhas*

Retransmitir msg requisição	*Filtrar duplicadas*	*Re-executar procedimento Retransmitir resposta*	*Semântica de invocação*
Não	Não aplicável	Não aplicável	*Incerta*
Sim	Não	Re-executar proc.	*Ao-menos-um*
Sim	Sim	Retransmit. Resp. (opcional)	*No-max.-um*

(Localização) Problemas de Transparência

- Objetivo RMI == LMI.
- Problemas
  - Mesmo que a sintaxe seja parecida existem diferenças de comportamento. Causa de erro: defeito no método ou comunicação
  - Lida-se com exceções lançadas devido à erros de comunicação

Implementação de RMI

- Figura 5.6 mostra como um objeto A invoca um método no objeto remoto B.
- Existe um módulo de Comunicação
  - Executa protocolo Requisição-Resposta
  - Responsável por implementar a semântica de invocação
  - O servidor seleciona o despachador de métodos do lado do servidor

“Binder” RMI e Threads do Servidor

- Um **binder** é um serviço separado que mantém uma lista contendo mapeamentos de nomes (texto) e objetos.
- Threads de Servidor
  - Cada chamada remota pode criar um novo thread para responder à chamada
  - Servidores com vários objetos remotos pode alocar threads para diferentes objetos
    - Quais as vantagens de cada abordagem?

“Binder” RMI e Threads de Servidor

- Ativação de objetos remotos
  - Objeto remoto é dito ativo quando ele habita um processo em execução.
  - É passivo quando ele pode ser ativado se chegar uma requisição.
- Um objeto que pode sobreviver entre ativações de processos é chamado objeto persistente.
- Um serviço de localização ajuda clientes a encontrar objetos remotos (ex: Java JNDI).