RPC - CORBA

Tecnologias de Middleware

2006/2007

Fernando Martins - fmp.martins@gmail.com

Tópicos

- RPC
- CORBA
- Conclusões

RPC

- *Remote Procedure Call* (RPC)
- RFC 707 - “A High-Level Framework for Network-Based Resource Sharing”, 1976
- Cedar, Xerox, 1981
- É a forma mais simples de *middleware*.

Objectivo

- Invocar de uma rotina remota, localizada noutro computador, de forma transparente.

- Abstracção de definições e implementação:
  - Canal de comunicação.
  - Formato da informação.
  - Tolerância a falhas.

Paradigma e Conceitos

- Base dos sistemas 2-*tier*
- Camada de abstracção ao nível da comunicação.
- Conceito cliente/servidor em sistemas heterogéneos
- *Interface Definiton Language* (IDL).
- Invocação transparente de rotinas remotas, como se se tratassem de rotinas locais.

Implementação da Comunicação

- RPC construído por cima de camadas de comunicação já existentes.

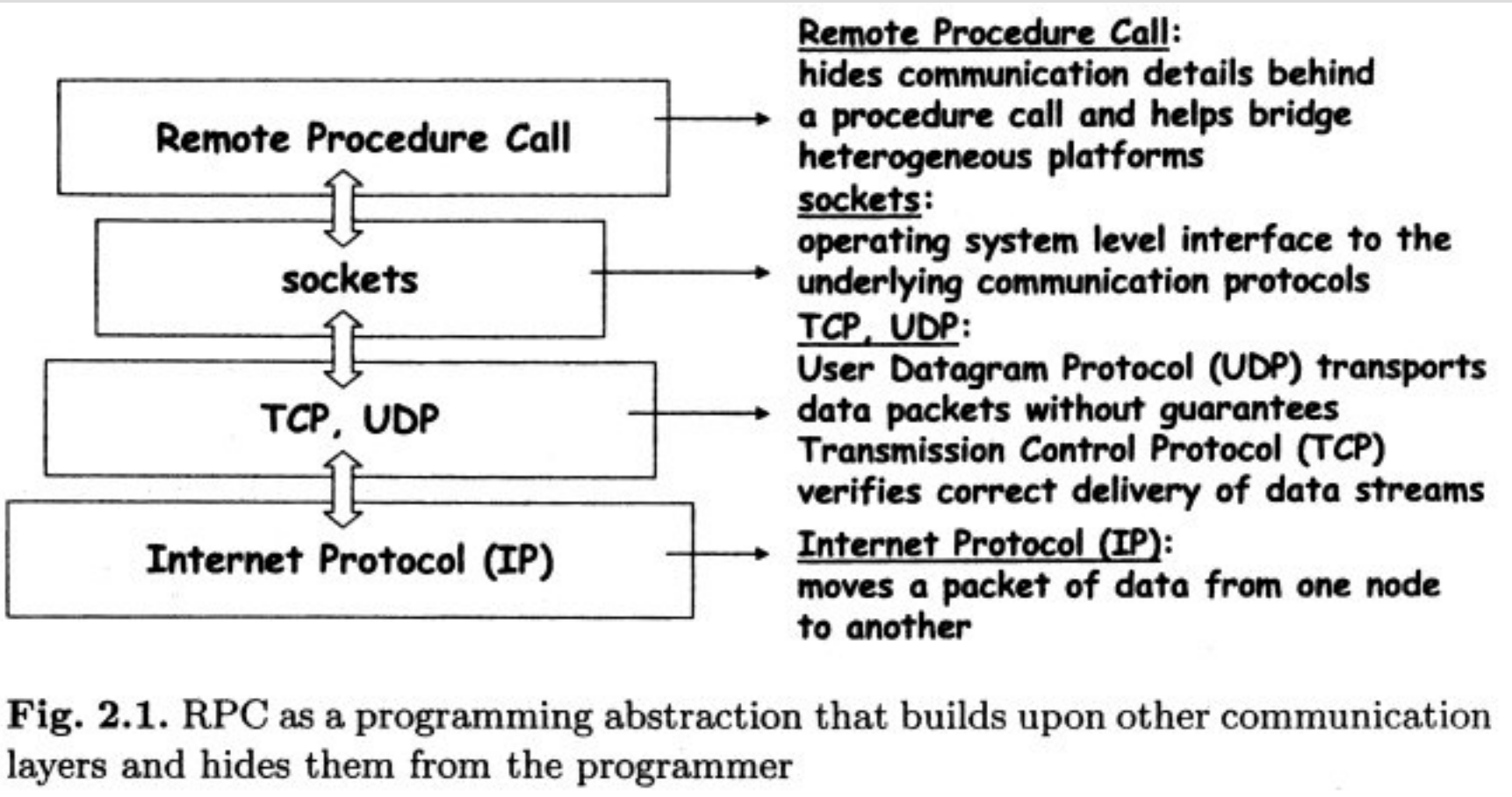

Fig. 2.1. RPC as a programming abstraction that builds upon other communication layers and hides them from the programmer

Especificação - IDL

- IDL é usada como forma de especificação da interface usada.
- Disponibiliza uma representação abstracta da rotina em termos de parâmetros de entrada e parâmetros de saída.
- Define a representação intermédia de troca de dados entre clientes e servidores.
- Pode ser considerada como a especificação do serviço disponibilizado pelo servidor.
- Permite ignorar diferenças de arquitectura, de linguagem implementação e de plataforma.

Implementação

- Exemplo de um IDL, tecnologia Microsoft (MIDL):

```
// File Example1.idl
[
   // A unique identifier that distinguishes this
   // interface from other interfaces.
   uuid(00000001-EAF3-4A7A-A0F2-BCE4C30DA77E),

   // This is version 1.0 of this interface.
   version(1.0),

   // This interface will use an implicit binding
   // handle named hExample1Binding.
   implicit_handle(handle_t hExample1Binding)
]
interface Example1 // The interface is named Example1
{
   // A function that takes a zero-terminated string.
   void Output( [in, string] const char* szOutput );
}
```

Implementação

- Cliente

```
#include <iostream>
#include "Example1.h"
int main() {
   RPC_STATUS status; unsigned char* szStringBinding = NULL;

   // Creates a string binding handle. This function is nothing more than a printf.
   status = RpcStringBindingCompose( NULL, reinterpret_cast<unsigned char*>("ncacn_ip_tcp"),
   reinterpret_cast<unsigned char*>("localhost"), reinterpret_cast<unsigned char*>("4747"), NULL, &szStringBinding);

   // Validates the format of the string binding handle and converts it to a binding handle.
   status = RpcBindingFromStringBinding( szStringBinding, hExample1Binding);

   RpcTryExcept   {
      // Calls the RPC function. The hExample1Binding binding handle is used implicitly. Connection is done here.
      Output("Hello RPC World!");
   }  RpcExcept(1)  {
      std::cerr << "Runtime reported exception " << RpcExceptionCode() << std::endl;
   }  RpcEndExcept

   // Free the memory allocated by a string.
   status = RpcStringFree(&szStringBinding); // String to be freed.

   // Releases binding handle resources and disconnects from the server.
   status = RpcBindingFree(&hExample1Binding); // Frees the implicit binding handle defined inthe IDL file.
}
```

Implementação

- Servidor

```
#include <iostream>
#include "Example1.h"

// Server function.
void Output(const char* szOutput)
{
   std::cout << szOutput << std::endl;
}

int main()
{
   RPC_STATUS status;

   // Uses the protocol combined with the endpoint for receiving remote procedure calls.
   status = RpcServerUseProtseqEp( reinterpret_cast<unsigned char*>("ncacn_ip_tcp"),
    RPC_C_PROTSEQ_MAX_REQS_DEFAULT, reinterpret_cast<unsigned char*>("4747"), NULL);

   // Registers the Example1 interface.
   status = RpcServerRegisterIf(Example1_v1_0_s_ifspec, NULL, NULL);

   // Start to listen for remote procedure calls for all registered interfaces.
   // This call will not return until RpcMgmtStopServerListening is called.
   status = RpcServerListen(1, RPC_C_LISTEN_MAX_CALLS_DEFAULT, FALSE);
}
```

Stubs

- Suportam detalhes de implementação de rede, incluindo *timeouts* e retransmissões.
- ***Client Stub:*** assinatura da rotina no ficheiro IDL resulta num *stub* de cliente que é responsável pela implementação da invocação do lado do cliente. Funciona como um *proxy*, reencaminhando a chamada à rotina para o servidor que implementa a rotina.
- ***Server Stub:*** semelhante ao do cliente, mas implementa a invocação do lado do servidor.

Stubs - Desenvolvimento

- Desenvolvimento dos RPCs

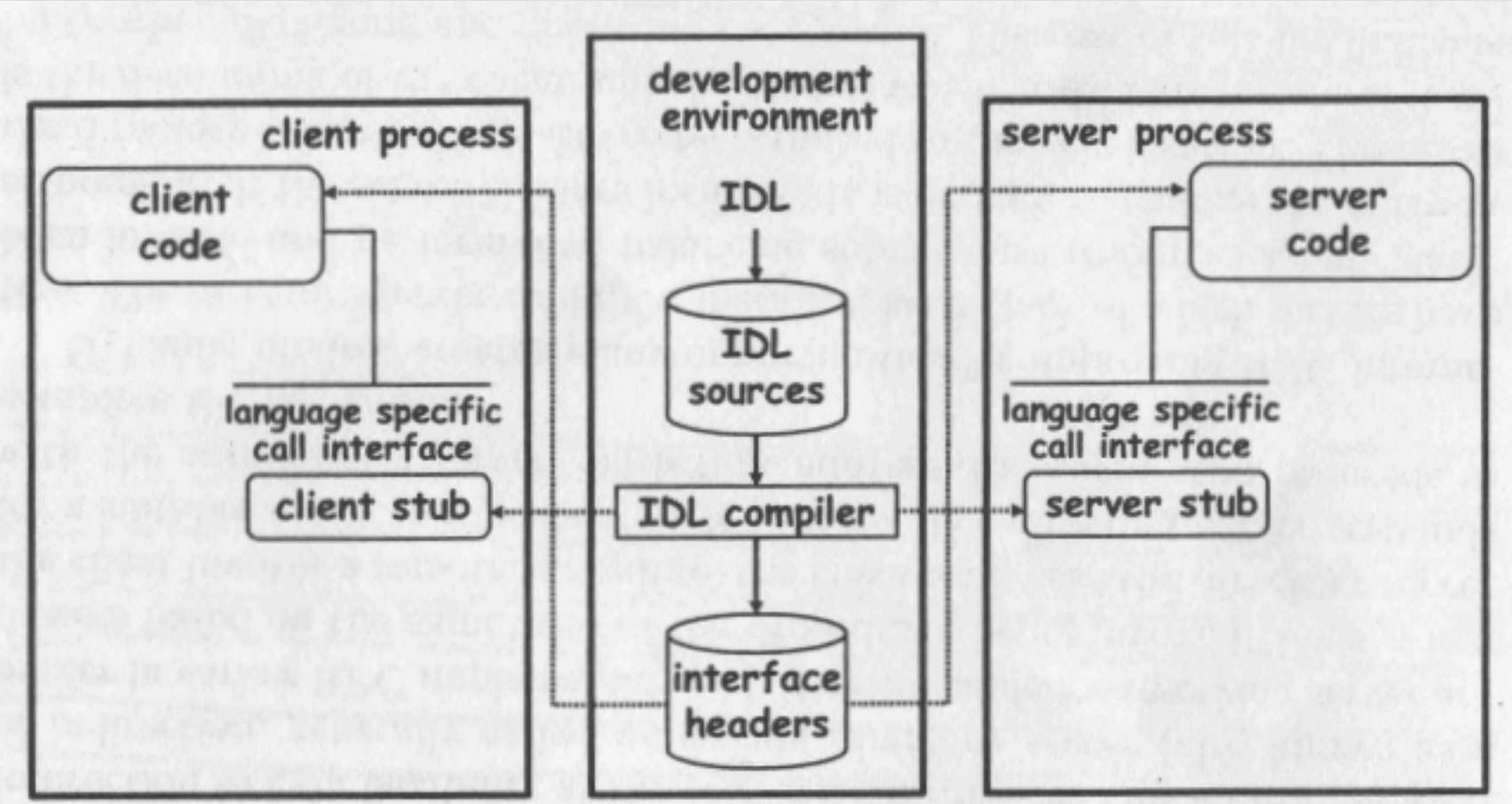

Fig. 2.2. Developing distributed applications with RPC

Heterogeneidade

- Distintas plataformas onde os clientes e os servidores podem ser executados.
- Implementação dos clientes e servidores em distintas linguagens.
- 2*N*M combinações de *stubs* cliente/servidor referentes a implementação e plataforma.
- Usar uma única forma de representação intermédia necessita de traduzir (N+M *stubs*)

Binding

- ***Estático:*** o *stub* do cliente possui toda a informação necessária para efectuar a invocação remota, i.e., sabe a localização do servidor e da rotina.
  - Vantagens:
    - Simples
    - Eficiente
  - Desvantagens:
    - Cliente e servidor com ligação demasiado estreita
    - Falhas do servidor não toleradas
    - Maior custo de manutenção

Binding

- ***Dinâmico:*** o *stub* cliente usa um serviço especial para localizar o servidor apropriado. Este serviço é uma adição de uma nova camada de localização, também conhecida como *name and directory server*.
  - Vantagens:
    - Flexibilidade
  - Desvantagens:
    - Performance

Segurança e Tolerância a Falhas

- RPC pode disponibilizar encriptação nas chamadas.

- RPC transaccional, suporte de gestão de erros e falhas.
- Não possui persistência *per se*, pelo que é necessário algum trabalho aplicacional.

Arquitectura

- Cliente invoca uma rotina remota.
- É efectuada uma chamada local à rotina disponibilizada pelo *stub*.
- O *stub*:
  - Localiza o servidor, i.e., efectua a ligação (*binding*) da chamada com o servidor;
  - Formata os dados através de:
    - *Marshaling*: empacotamento dos dados num formato de mensagem pronto para envio;
    - *Serialização*: transformação da mensagem numa *string* de *bytes*.

Arquitectura

- O *stub* (continuação):
  - Comunica com o servidor enviando a mensagem;
  - Recebe a resposta;
  - Formata os dados:
    - Deserialização;
    - *Unmarshaling;*
  - *Reencaminha a resposta como o parâmetro de saída da rotina invocada*
- *Cliente recebe a resposta da invocação da função.*

Arquitectura

- Funcionamento básico, ligação estática

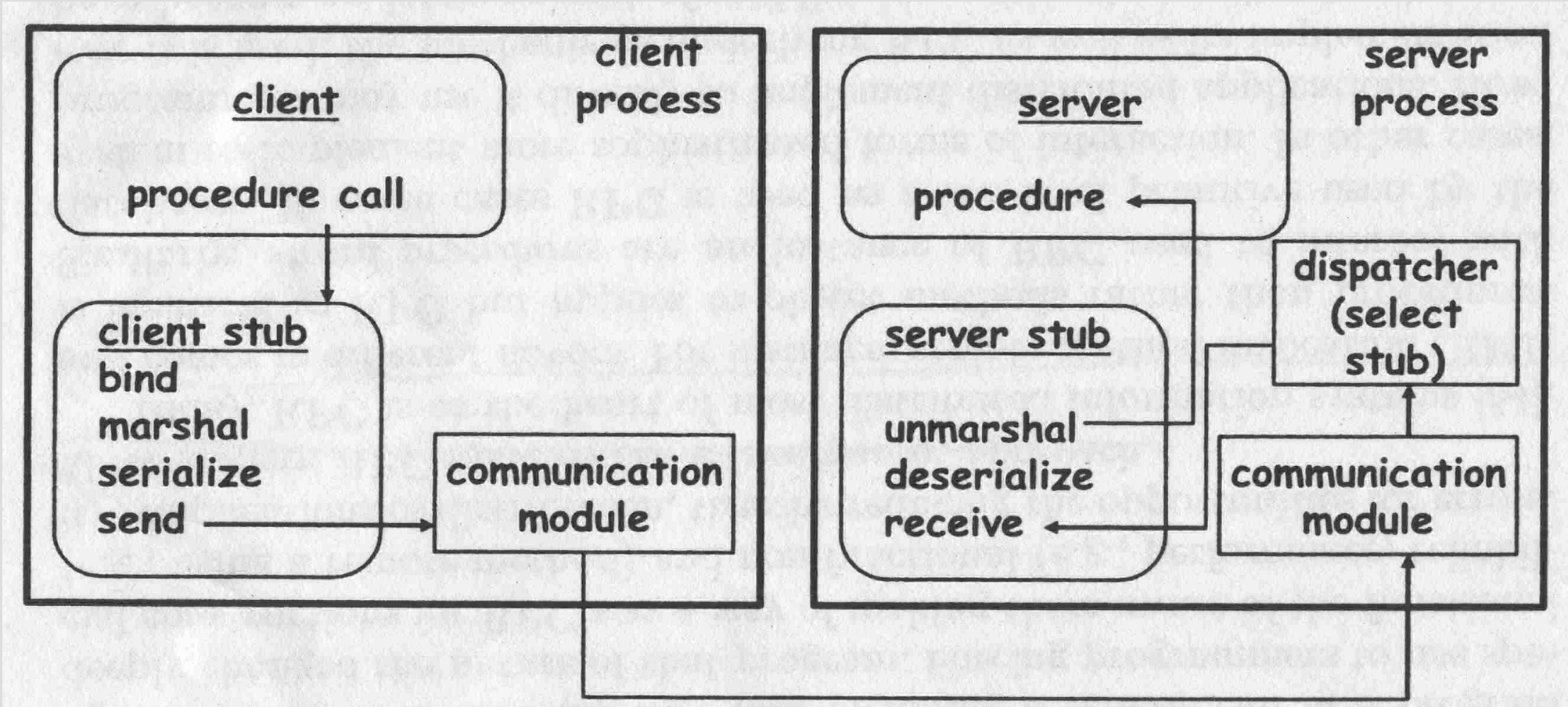

Fig. 2.3. Basic functioning of RPC

Arquitectura

- Funcionamento com ligação dinâmica

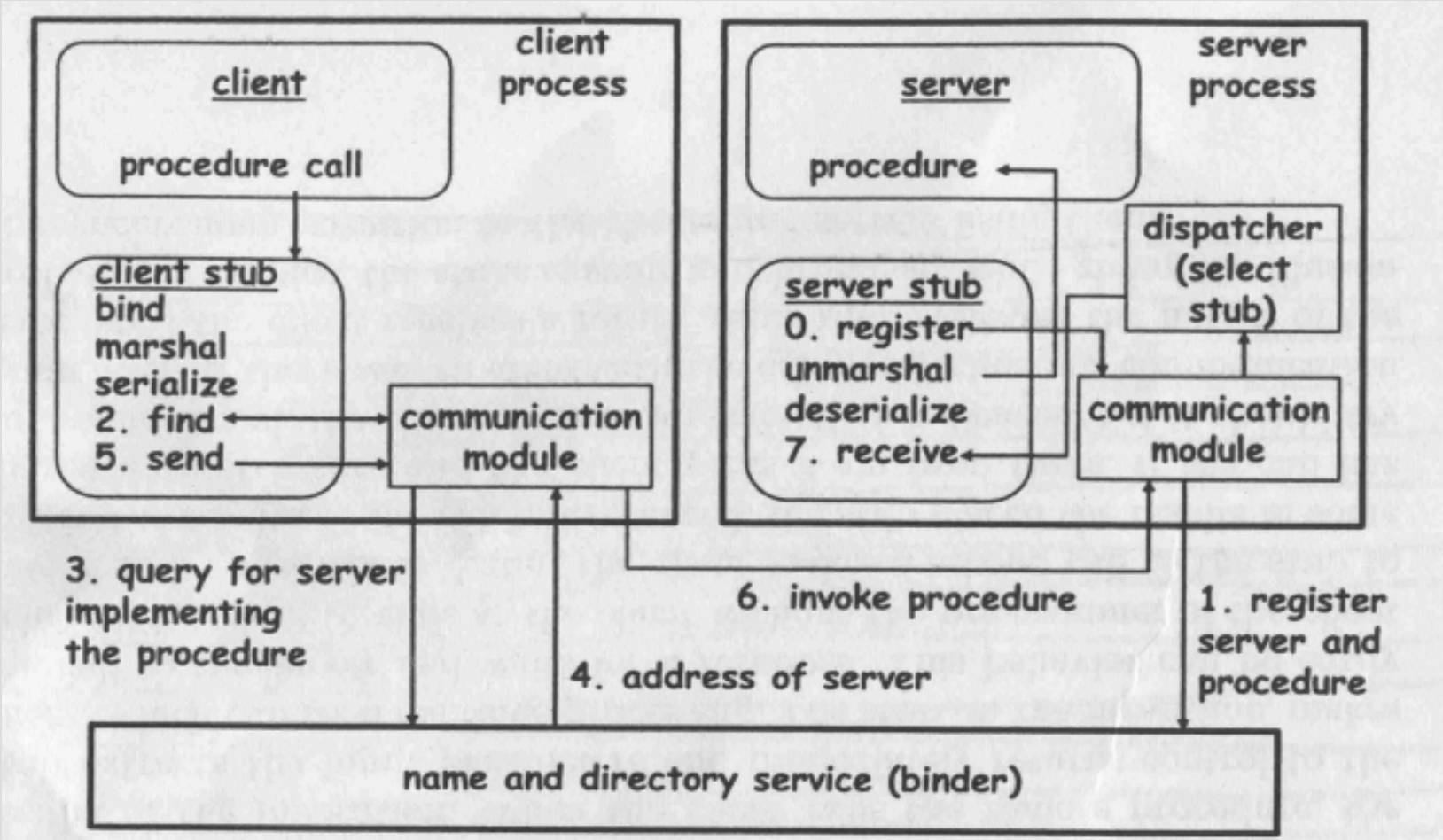

Fig. 2.4. Dynamic binding allows a server to register the procedures it implements

Extensões – RPC Assíncrono

- Chamadas remotas não bloqueantes, permitindo a execução do cliente sem que este tenha de esperar pela resposta:
  - Aplicação cliente efectua chamada remota e continua sem esperar pela resposta.
  - Mais tarde efectua nova chamada para obter a resposta, obtendo a resposta esperada ou um código de erro.
- Requer infraestrutura mais sofisticada.

Extensões – DCE

- *Distributed Computer Environment* (DCE) resulta da tentativa (falhada) de standarizar o RPC por parte da Open Software Foundation.
- Disponibiliza uma especificação completa do modo de funcionamento do RPC e de como devia ser implementado.

Extensões – DCE

- Disponibiliza serviços como:
  - *Cell directory service:* um *name and directory server* sofisticado para criar e gerir domínios RPC na mesma rede;
  - *Time service:* sincronização através de todos os nós da rede;
  - *Thread service:* suporte de *threads* e múltiplos processadores;
  - *Distributed file service:* disponibiliza a partilha de ficheiros de dados;
  - *Security service:* disponibiliza comunicação autenticada e segura

Extensões – DCE

- Arquitectura

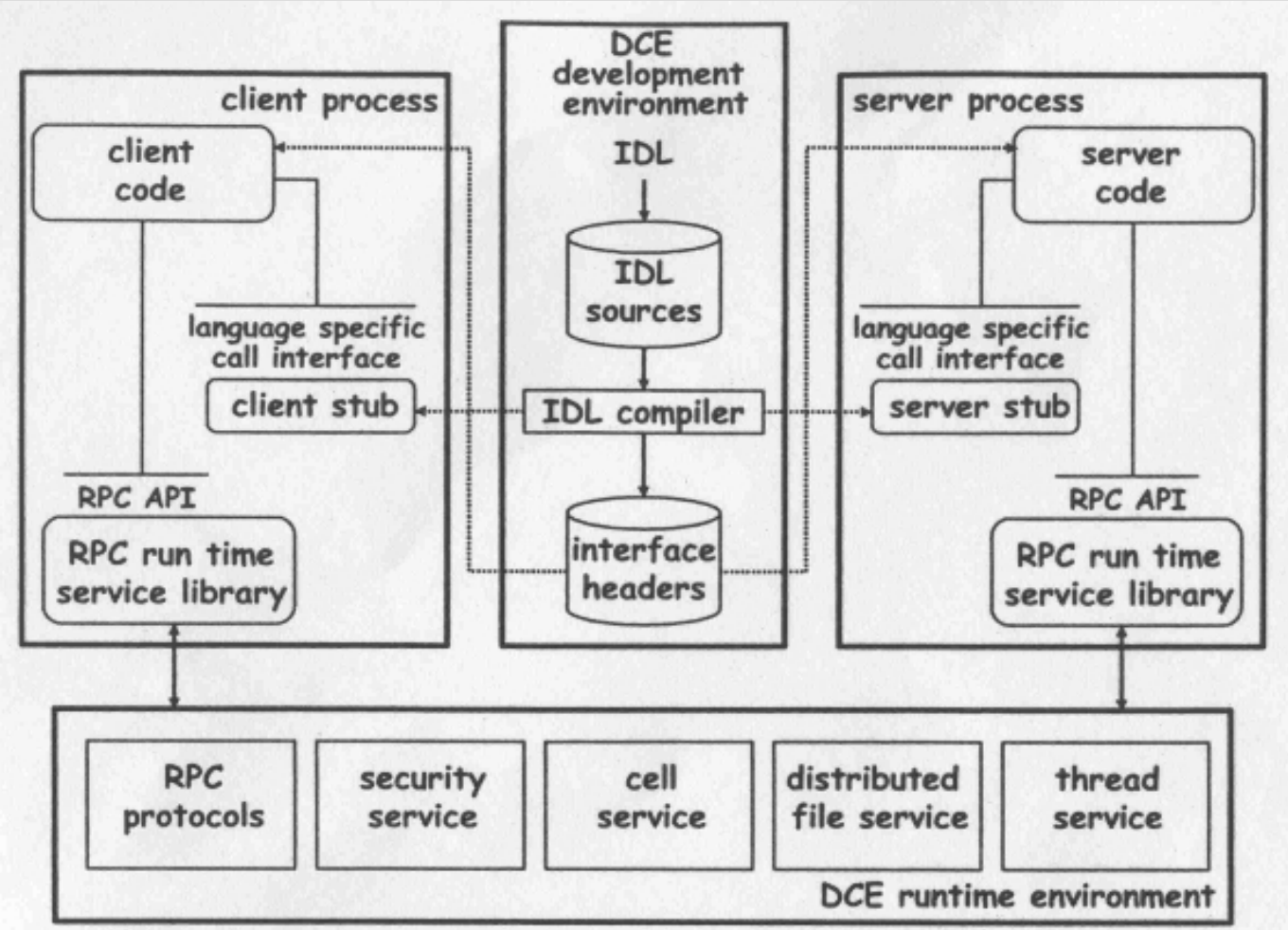

Fig. 2.5. The architecture of DCE

CORBA

- *Common Object Request Broker Arquitecture.*
- Produto do consorcio *Object Management Group* (OMG) de mais de 850 empresas:
  - Versão 1.0: 1991
  - Versão 2.0: 1996
  - Versão 2.3: 1998
  - Versão 3.0: 1999
- Microsoft possui o seu próprio *object broker*, *Distributed Component Object Model* (DCOM).

Objectivo

- Integração de sistemas e aplicações em plataformas heterogéneas.
- "*IDL-izar*" todo o middleware cliente/servidor e todos os componentes que vivem num *Object Request Broker* (ORB).

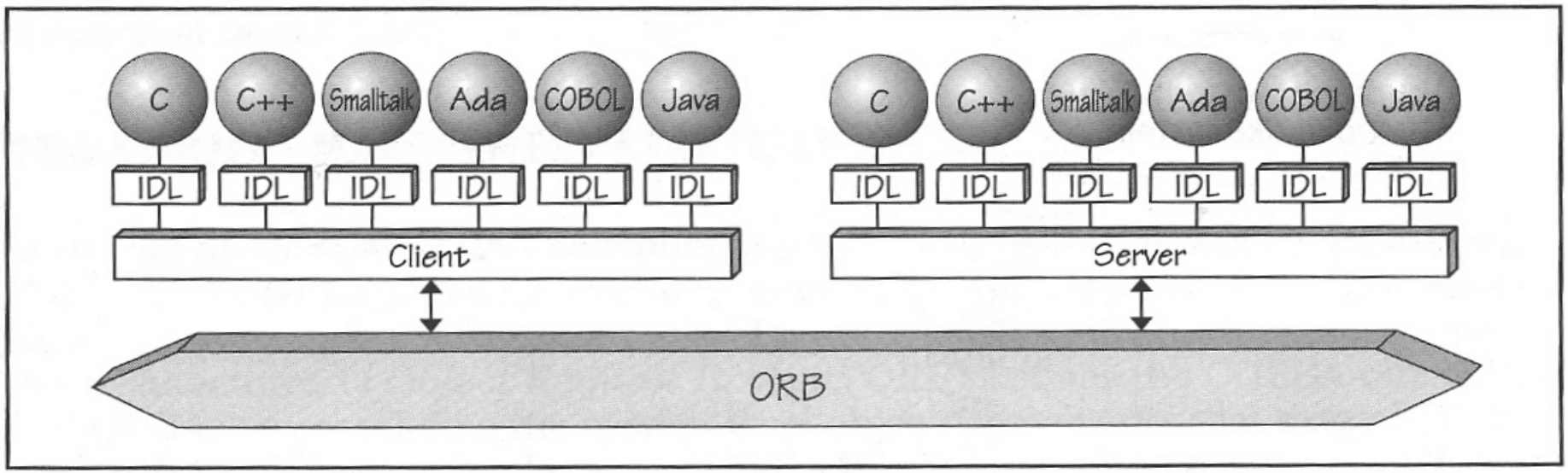

Figure 1-1. CORBA IDL Language Bindings Provide Client/Server Interoperability.

Paradigma e Conceitos

- Auto descritivo.
- Separa a implementação da especificação dos serviços.
- Usa objectos normais e transforma-os em:
  - Transaccionais
  - Seguros
  - Bloqueáveis
  - Persistentes
  - Disponíveis ao longo de uma rede
- ORB efectua a mediação entre o cliente e o objecto.

Paradigma e Conceitos

- Clientes acedem aos componentes através da invocação dos seus métodos.
- Objectos podem actuar como clientes, servidores ou ambos.
- Clientes têm apenas noção da existência de um componente, não sabem:
  - A localização do componente;
  - Qual a implementação do componente;
  - O sistema operativo que o suporta;

Especificação - IDL

- A especificação é escrita numa linguagem própria, declarativa, baseada na sintaxe de C++ e neutra em relação à implementação: não possui informação de implementação.
- *Interface Definition Language* (IDL) define as fronteiras do componente, i.e., as interfaces disponibilizadas:
  - Atributos;
  - Informação de herança;
  - Excepções;
  - Eventos;
  - Métodos, incluindo parâmetros e os tipos;

Object Request Broker

- O *Object Request Broker* (ORB) é o “maestro” do CORBA, uma vez que é responsável por:
  - Intercepção de chamadas;
  - Descoberta de objectos;
  - Invocação de métodos;
  - Passagem de parâmetros;
  - Retorna resultado ou mensagem de erro;
- Permite assim que a comunicação entre objectos seja efectuada de forma transparente e independentemente da sua localização.

Basic Object Adapter

- Basic Object Adapter (BOA) faz a interface com o ORB e com a implementação do *esqueleto*.

- Define como um objecto é activado:
  - Novo processo;
  - Nova *thread*;
  - Reutilização do processo actual;
  - Reutilização da *thread* actual;

Implementação

- Compilador IDL faz a geração das seguintes interfaces:
  - *Stub*: actua como uma chamada a uma função local, disponibilizando a interface ao ORB.
  - *Esqueleto*: implementação da interface IDL do lado do servidor.

```
#ifndef _MONEY_IDL
#define _MONEY_IDL

module Money
{
    interface Accounting
    {
      float get_outstanding_balance();
    };
};

#endif
```

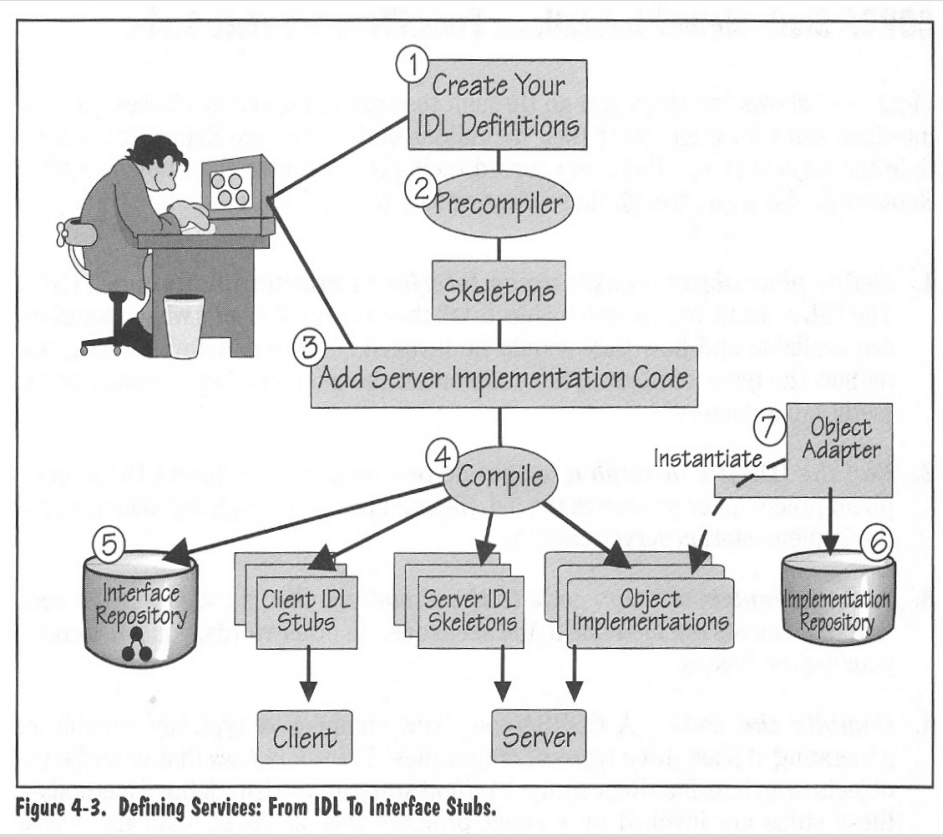

Figure 4-3. Defining Services: From IDL To Interface Stubs.

Implementação

- Cliente Java

```
import org.omg.CORBA.*;
public class Client
{
  public static void main(String args[]) {
    try {
      // Initialize the ORB.
      System.out.println("Initializing the ORB...");
      ORB orb = ORB.init(args, null);

      // bind to an Accounting Object named "Account"
      System.out.println("Binding...");
      Money.Accounting acc = Money.AccountingHelper.bind(orb, "Account");

      // Get the balance of the account.
      System.out.println("Making Remote Invocation...");
      float balance = acc.get_outstanding_balance();

      // Print out the balance.
      System.out.println("The balance is $" + balance);
    } catch(SystemException e) {
      System.err.println("Oops!  Caught: " + e);
    }
  }
}
```

Implementação

- Servidor Java

```
import Money.*;
import org.omg.CORBA.*;
class AccountingImpl extends _AccountingImplBase
{
   public float get_outstanding_balance() {
    float bal = (float)14100.00; // Implement function here
    return bal;
   }
   public static void main(String[] args)
   {
    try {
       ORB orb = ORB.init(args, null); // Initialize the ORB.
       BOA boa = orb.BOA_init();       // Initialize the BOA.

       System.out.println("Instantiating an AccountingImpl.");
       AccountingImpl impl = new AccountingImpl("Account");

       boa.obj_is_ready(impl);
       System.out.println("Entering event loop."); // Wait for incoming requests
       boa.impl_is_ready();
    } catch(SystemException e) {
       System.err.println("Oops!  Caught: " + e);
    }
  }
}
```

Serviços

- ***Object life cycle*:** define como os objectos são criados, removidos, movidos e copiados.
- ***Naming*:** define os nomes simbólicos dos objectos.
- ***Events*:** separa a comunicação entre objectos distribuidos.
- ***Relationships:*** disponibiliza relações *n*-árias tipificadas entre objectos.
- ***Externalization:*** coordena a transformação de objectos de e para meios externos

Serviços

- ***Transactions:*** coordena o acesso atómico aos objectos.
- ***Concurrency Control:*** disponibiliza serviço de *lock* para os objectos garantindo o acesso serializado.
- ***Property:*** suporta a associação de pares nome-valor com os objectos.
- ***Trader:*** suporta a descoberta de objectos baseado em propriedades que descrevem o serviço disponibilizado pelo objeco.
- ***Query:*** suporta pesquisas sobre objectos.

Facilities

- Construídas por cima dos serviços, são:
  - Verticais: orientados a um ramo específico de indústria/mercado/negócio: medicina, direito, logística, etc..
  - Horizontais: transversais ao sistema, disponibilização de serviços de alto nível: internacionalização, gestão de informação, suporte a agentes móveis, etc..

Invocação Estática

- Invocação estática é a funcionalidade mais básica, parecido com RPC ou invocação de um método em Java.
- Necessita da referência do objecto CORBA antes de invocar os seus métodos.

Invocação Dinâmica

- Invocação dinâmica permite invocar a operação a efectuar e os parâmetros da mesma através da especificação de:
  - Informação da operação a executar;
  - Tipos de parâmetros passados (possivelmente obtidos através de um repositório de interfaces);

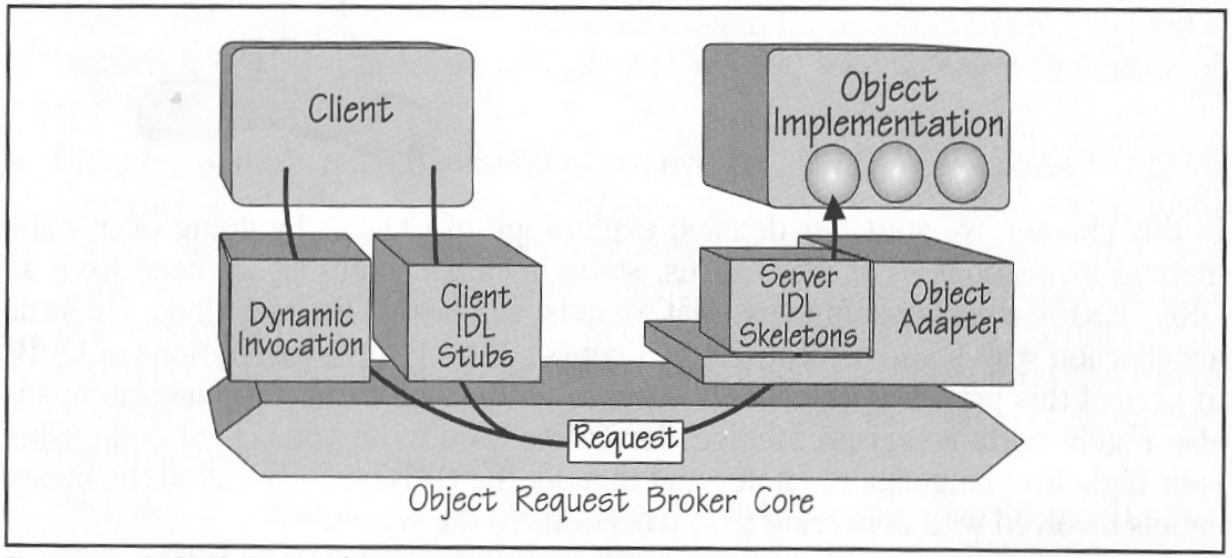

Figure 4-1. CORBA's Static and Dynamic Method Invocations.

Segurança e Tolerância a Falhas

- ORB possui informação de contexto de forma a garantir segurança e transacções.
- Um ORB pode substituir outro caso em caso de falha.

Arquitectura

- O serviço disponibilizado por um objecto é dado pela sua interface.
- A interface dos objectos é defina em IDL.
- Os objectos distribuídos são identificados por referência, tipificada pela interface IDL.

Arquitectura

- Chamada ao método do objecto desejado com os seus parâmetros.
- ORB procura o objecto em causa e invoca-o com os parâmetros especificados.
- Objecto efectua o seu processamento e retorna o resultado.
- ORB entrega o resultado.

Arquitectura

- Visão global

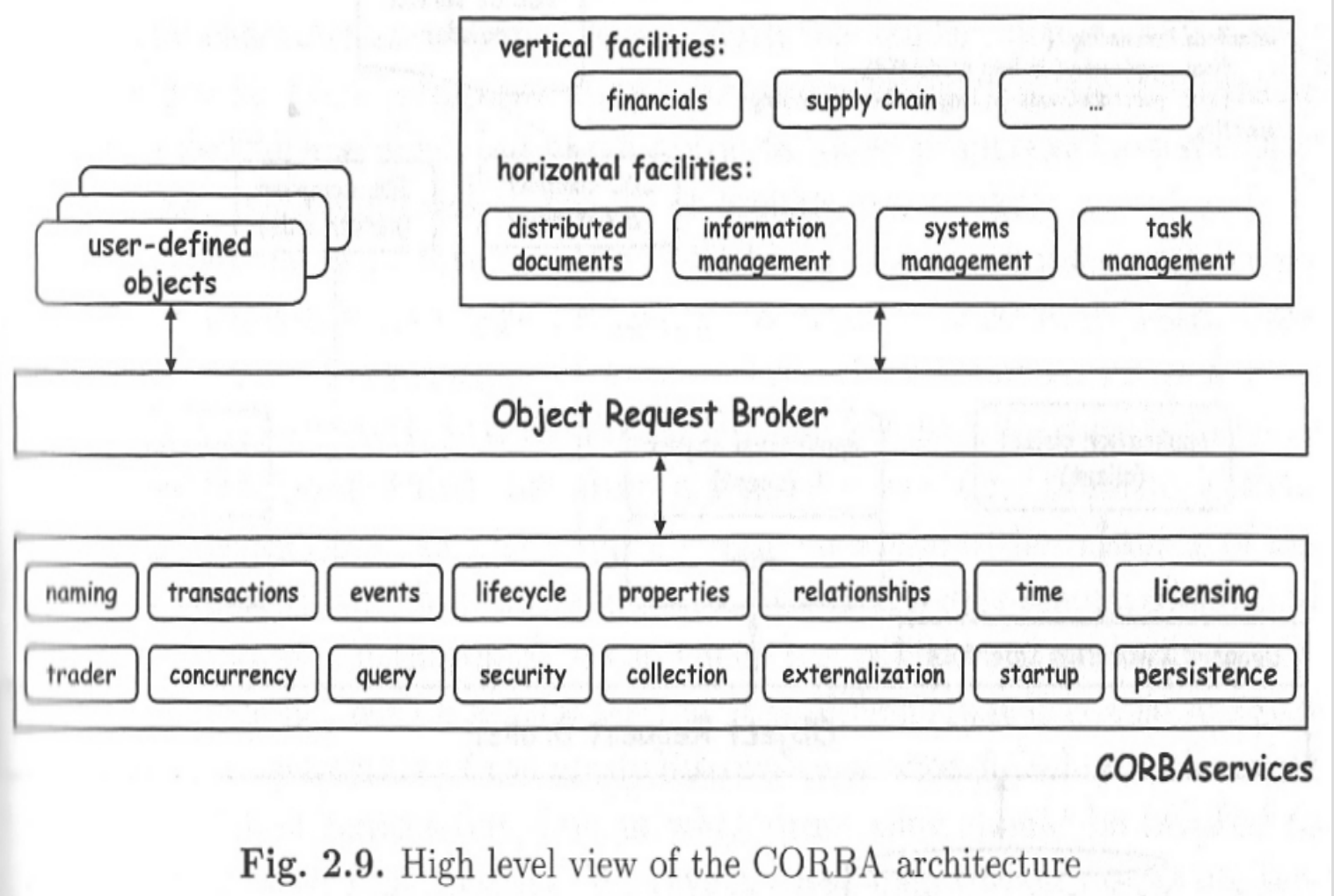

Fig. 2.9. High level view of the CORBA architecture

Extensões

- General Inter-ORB Protocol (GIOP), integração de vários ORBs de forma a ligar vários sistemas.
- Internet Inter-ORB Protocol, especialização do GIOP sobre a internet.
- Novos desafios:
  - Rapidez de comunicação
  - Segurança
  - Novos requisitos, como servidor de directórios.

Conclusão

- As vantagens da abstracção disponibilizada em *middleware* aumenta o custo e a complexidade das infraestruturas que a suportam.

- RPC foi o percursor do *middleware*.
- CORBA é a evolução natural de RPC.
- As tecnologias partilham:
  - Objectivo;
  - Filosofia;
  - Arquitectura base;

Conclusão

- Mecanismos de invocação semelhantes, mas:
  - RPC: invocação de uma função especifica;
  - ORB: invocação de métodos de objectos, no entanto, métodos de classes diferentes podem responder de forma diferente, dado o polimorfismo;
  - RPC: todas as funções com o mesmo nome possuem a mesma implementação;
  - ORB: precisão cirúrgica, a chamada dá-se num objecto especifico que controla dados específicos que por sua vez implementa a função tal como especificada para a classe em causa;

Bibliografia

- Birrel, Nelso. “Implemening Remote Procedure Calls”, ACM Transactions on Computer Systems, Vol 2, N.1, Feb. 1984, pp 39-59.
- Alonso, Casati, Kuno, Machiraju. “Web Services: Concepts, Architectures and Applications”, Springer, 2004
- OMG. “Common Object Request Broker Architecture: Core Specification”, 2004.
- Orfali, Robert; Harkey, Dan; Edwards, Jeri. “Instant Corba”, Wiley Computer Publishing, 1997.

Outras Fontes:

- RPC: http://www.wikipedia.com/
- Code Project: http://www.codeproject.com/internet/rpcintro1.asp e http://www.codeproject.com/internet/rpcintro2.asp
- OMG: http://www.omg.org/
- Introduction to CORBA: http://java.sun.com/developer/onlineTraining/corba/
- CORBA: http://www.wikipedia.com/
- CORBA Services: http://www.javaolympus.com/J2SE/NETWORKING/CORBA/CORBAServices.jsp