Tecnologias de Middleware

RPC – Remote Procedure Calls

Middleware (MW)

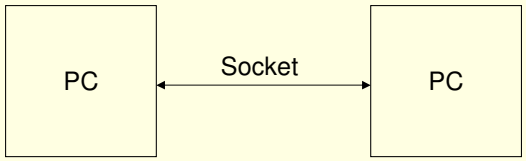
- Disponibiliza e gere a interacção entre aplicações através de plataformas de computação heterogéneas
- Solução de arquitectura para o problema da integração de servidores e aplicações numa interface comum
- Oferece abstracções de programação que escondem alguma da complexidade de construir uma aplicação distribuída

•

• Integrar duas BDs numa mesma rede não é o mesmo que integrar 3 sistemas diferentes de uma mesma companhia residentes em várias sucursais através de uma linha alugada, nem se os sistemas são de companhias diferentes e necessitam de comunicar via Internet.

• Devido às abstracções disponibilizadas pelo MW, o programador não precisa de implementar diversas funcionalidades relacionadas com as aplicações distribuídas.

Aplicação Distribuida Sem MW



• Aplicação em que parte do código deve ser executado num PC e outra parte noutro PC.

• Sockets para abrir um canal de comunicações entre a parte da aplicação num PC e a parte da aplicação no outro.

• Usar o socket para trocar informação.

• Escrever código para criar o canal e para lidar com os erros e/ou falhas que possam ocorrer com o canal.

• Implementar um protocolo de forma a que as duas partes da aplicação possam trocar informação de uma forma ordeira.

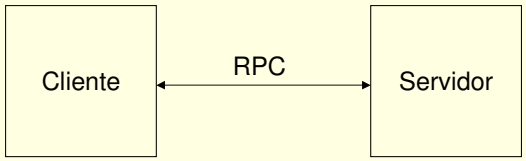
• O protocolo especifica quem envia o quê, quando e qual a resposta esperada.

• Definir o formato da informação trocada.

• Implementar a aplicação que usa o canal de comunicações.

• Código para lidar com mensagens erradas, falhas no outro extremo do canal, procedimentos de recuperação para operações falhadas, etc...

Aplicação Distribuida com RPCs



• Abstracção que esconde o canal de comunicações atrás de uma interface com a aparência exacta de uma chamada a um procedimento.

• Com RPCs só é necessário reformular a comunicação entre as duas partes da aplicação, de modo a usar as chamadas a procedimentos (RPCs).

• O resto é feito pela abstracção RPC implementada pelo MW.

Camadas escondidas pela abstracção RPC

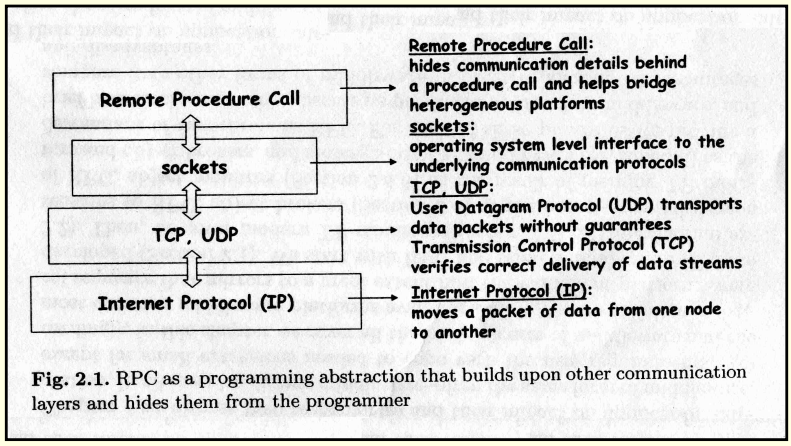


Fig. 2.1. RPC as a programming abstraction that builds upon other communication layers and hides them from the programmer

• Os RPCs agem como a interface de programação contruída em cima da interface de comunicações disponibilizada pelo SO (sockets), que por sua vez é construída em cima de uma pilha de protocolos de comunicação.

• Se se desejar que o MW também faça tratamento de erros, podemos usar RPCs transaccionais.

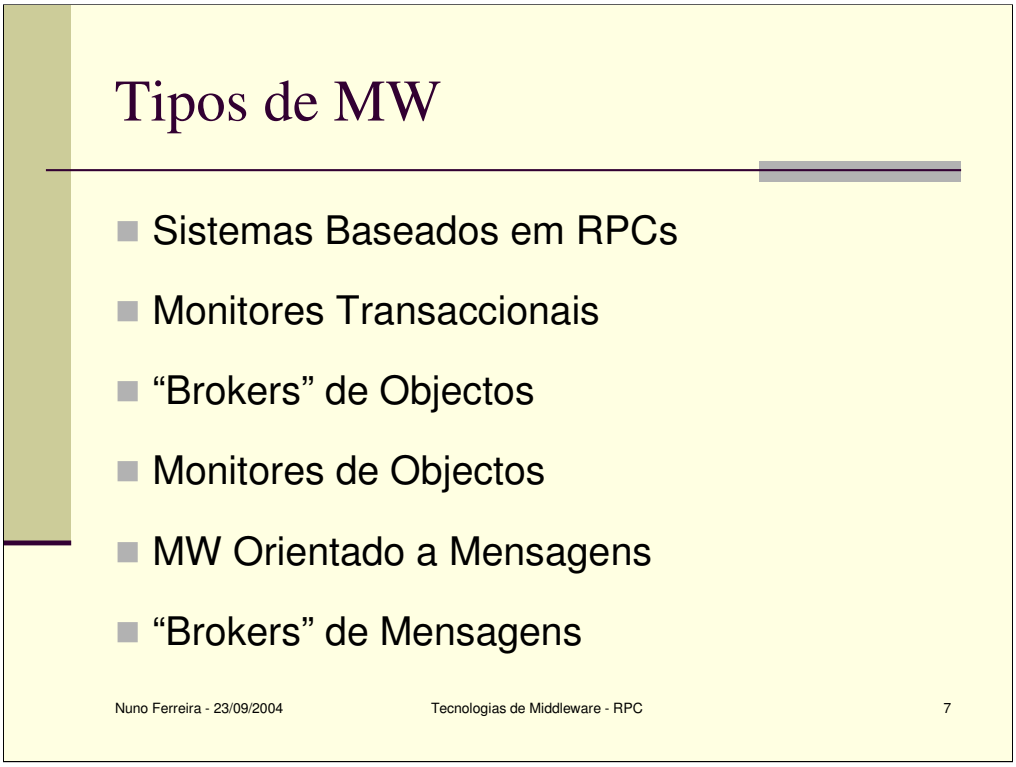
• Com RPCs transaccionais não é necessário preocuparmo-nos com o estado da invocação ou interacção caso aconteça algum erro.

• O MW garante a recuperação de possíveis efeitos secundários caso uma transacção necessite de abortar, através de procedimentos de recuperação.

• Algumas plataformas de MW modernas implementam persistência automática, de forma que as aplicações, em caso de falha, possam manter o estado de coerência e recuperar rapidamente.

Prós e contras do MW

- Quanto mais fizer o MW, melhor será para a programação de uma aplicação
- Mais funcionalidades=>maior complexidade e custos
- O segredo é usar apenas as funcionalidades necessárias à aplicação
- As funcionalidades de MW normalmente são disponibilizadas em plataformas completas
- Usar plataformas menos poderosas e extendê-las conforme as necessidades
- As vantagens da abstracção disponibilizada pelo MW são obscurecidas pelo custo e complexidade das infraestruturas a suportá-las



• **RPCs** são a forma mais básica de MW, são a base de quase todas as outras formas de MW, incluindo o MW de Serviços Web.

• **Monitores Transaccionais** são a forma de MW mais antiga e melhor conhecida, é também a tecnologia mais fiável, melhor testada e mais estável na área da integração aplicacional de empresas. Podem ser vistos como RPCs Transaccionais.

• **Brokers de Objectos** são a evolução das plataformas de MW para suportar a invocação de objectos remotos, sendo em termos de implementação muito semelhantes aos RPCs.

• Extensão aos Monitores Transaccionais para suportar as linguagens orientadas a objectos. A convergência entre os Monitores TP e os Brokers de Objectos resultou em sistemas híbridos chamados **Monitores de Objectos.**

• **MW Orientado a Mensagens** são a extensão dos Monitores Transaccionais com sistemas de filas persistentes de mensagens, disponibilizando o acesso às filas e a um conjunto de primitivas para leitura/escrita de filas locais/remotas.

• **Brokers de Mensagens** são uma forma distinta de MW orientado a mensagens, com a capacidade de transformar e filtrar as mensagens nas filas. Também são capazes de determinar dinâmicamente o destinatário das mensagens, baseados no seu conteúdo. Lógica aplicacional pode ser aplicada às filas, permitindo a implementação de interacções assincronas muito mais sofisticadas.

Convergência do MW

- Demasiado MW com funcionalidades semelhantes e concorrentes
- Problemas ao usar diferentes plataformas MW para poder usar diferentes abstracções de implementação num mesmo sistema
- Duas tendências na evolução do MW, a consolidação de plataformas complementares e o aparecimento de pacotes massivos de produtos que disponibilizam, num ambiente integrado, várias formas de MW

•

• Como cada plataforma de MW disponibiliza a sua infraestrutura fixa, usar várias plataformas implica ter de lidar com várias infraestruturas. Uma percentagem significativa da infraestrutura base é idêntica em todas as plataformas e muito provavelmente baseada em RPCs, mas tornada incompatível devido à customização da infraestrutura a cada produto.

• Monitores de Objectos => Consolidação entre os Monitores Transaccionais e os Brokers de Objectos. As grandes suites de plataformas de MW são disponibilizadas por algumas das maiores empresas do ramo, apesar de ainda não ser perfeita a integração, o caminho melhora rapidamente com as versões implementadas dessas suites.

RPC – O Início

- Surge no início dos anos 80 por A. Birrell e B. Nelson
- O artigo original apresenta os RPCs como uma forma de invocar procedimentos noutras máquinas de uma forma transparente
- Estabelece as noções de cliente, servidor, "Interface Definition Languages" (IDL), serviços de nomes e directórios, ligação dinâmica, interface de serviço, etc.

•

•

• As noções estabelecidas surgem em todas as formas de MW, e também são grande parte dos Serviços Web.

RPC – O Início

- Forma “limpa” de lidar com a distribuição
- Noção de procedimento muito familiar na altura
- Inicialmente era uma colecção de bibliotecas
- A questão sobre se os RPCs deveriam ser ou não ser transparentes ao programador foi muito debatida
- A maioria dos sistemas modernos baseados em RPCs usa a aproximação da tranparência

• Os RPCs tornaram possível começar a construir aplicações distribuídas sem mudar as linguagens nem os paradigmas de programação. Mais tarde, com o aumento de funcionalidades, tornou-se numa plataforma de MW.

•

• Para um programa se tornar parte de um sistema distribuído, bastava ser compilado e ligado ao conjunto correcto de bibliotecas RPC.

• Vozes a favor da transparência indicavam a simplicidade do conceito e o facto de não ser necessário aos programadores lidar com as questões da distribuição directamente.

As vozes contra a trasparência defendiam que ao incluir uma chamada remota num programa, mudava a natureza desse programa. Obrigar os programadores a usar contruções especiais para os RPCs, seria uma forma de os alertar para as implicações da distribuição, daí reduzindo a oportunidade de surgirem erros.

Hoje os RPCs são o coração da maioria dos sistemas de informação distribuídos.

O “Remote Method Invocation” é idêntico aos RPCs mas aplicado a métodos de objectos.

As “Stored Procedures” são uma instância de RPCs aplicadas a BDs.

RPC – Funcionamento

- Passos para desenvolver um servidor que implementa um procedimento que irá ser usado remotamente por um cliente:
  1. Definir a interface do procedimento e gerar a descrição IDL
  2. Compilar a descrição IDL para produzir os "stubs" cliente/servidor, os "templates" e referências de código
  3. Compilar e ligar o código cliente/servidor com os "stubs" e "templates" e referências de código

• Com recurso a uma linguagem de definição de interfaces (IDL), é disponibilizada uma representação abstracta do procedimento com os parâmetros de "input" e os valores retornados.

• Com a descrição IDL obtemos a especificação dos serviços disponibilizados pelo servidor e podemos desenvolver o cliente e o servidor.

• Qualquer implementação RPC ou qualquer MW que use RPCs, disponibiliza esse compilador de interfaces.

RPC – Funcionamento

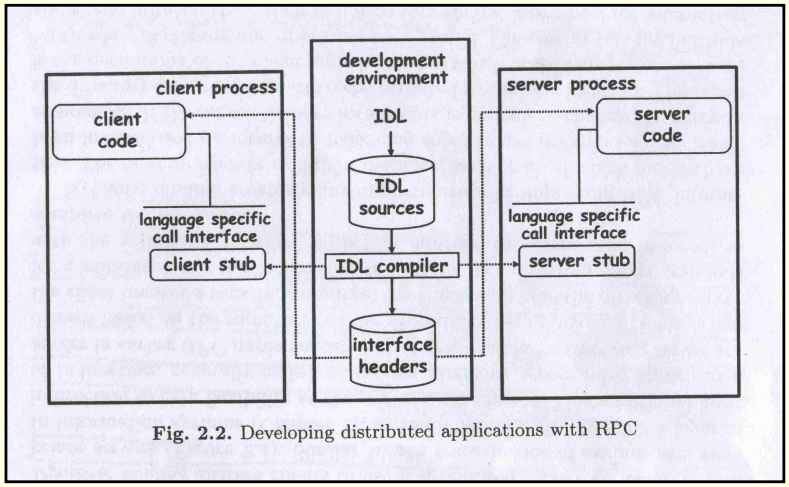


Fig. 2.2. Developing distributed applications with RPC

• Cada assinatura de procedimento no ficheiro IDL resulta num "stub" cliente ou num "stub" servidor.

• O "stub" é um pedaço de código para ser compilado e ligado com o código do cliente ou servidor.

• Quando o cliente efectua uma chamada a um procedimento remoto, a chamada é realmente efectuada localmente ao procedimento disponibilizado pelo "stub" cliente. O "stub" localiza o servidor, formata os dados, comunica com o servidor, obtém a resposta e devolve-a ao cliente como valor de retorno do procedimento invocado pelo cliente.

• O "stub" cliente não implementa o procedimento, mas todos os mecanismos necessários para interagir remotamente com o servidor, de forma a ser lá executado esse procedimento.

• O "stub" servidor contém o código para receber a invocação do "stub" cliente, formatar os dados conforme as suas necessidades, invocar o procedimento real implementado no servidor e devolver os valores retornados pelo procedimento ao "stub" cliente.

• O compilador IDL produz todos os ficheiros auxiliares necessários ao desenvolvimento da aplicação. As 1as versões dos RPC em C, além dos "stubs", geravam os ficheiros *.h necessários na fase de compilação. Hoje, também geram "templates" com código base para o servidor, de forma a ser adicionado código para implementar o procedimento no cliente e servidor.

RPC – Funcionamento

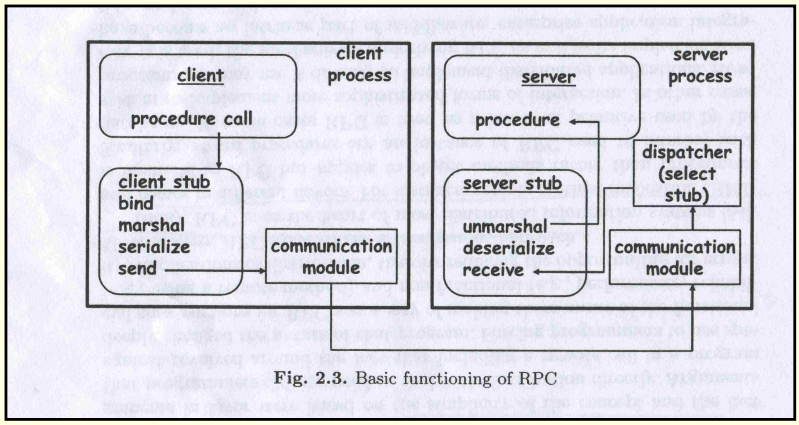


Fig. 2.3. Basic functioning of RPC

• Os “stubs“ RPC lidam com todos os detalhes de programação das comunicações em rede, incluindo “timeouts” e retransmissões se for usado um protocolo não confiável (UDP).

• Se mais funcionalidades forem necessárias, mais interfaces RPC são disponibilizadas pelas infraestruturas.

•Marshaling – Formatar dados numa mensagem antes da sua transmissão.

• Serializing – Transformar a mensagem numa sequência de bits antes de ser enviada num canal de comunicações.

RPC – Ligação

- Ligação é o processo pelo qual o cliente cria uma associação local a um dado servidor de forma a invocar um procedimento remoto
- Ligação estática
- Ligação dinâmica

• Na ligação estática o “stub” cliente é “hardcoded” de forma a conter o “handle” do servidor onde está o procedimento. O tipo deste handle depende do ambiente, pode ser um IP e um porto, pode ser um endereço X.500, etc. Ao invocar o procedimento, o “stub” cliente simplesmente reencaminha a chamada para o servidor.

• A ligação estática é simples e eficiente e não é necessária nenhuma infraestrutura adicional aos “stubs” cliente/servidor. Por outro lado, o cliente e o servidor ficam fortemente associados. Se o servidor falhar, o cliente não funcionará. Se o servidor mudar de local, o cliente tem de ser re-compilado com o novo “stub” que incluirá a nova localização. Redundância de servidores impossível.

• Na ligação dinâmica existe um serviço específico, usado pelos clientes, que se encarrega de localizar os servidores adequados. Chama-se Serviço de Nomes e Directórios e adiciona uma nova camada de indirecção para ganhar flexibilidade, à custa da performance.

• O Serviço de Nomes e Directórios devolve o endereço de um servidor ao “stub” cliente de forma a poder completar a sua invocação.

• Com a ligação dinâmica pode ser efectuado o balanceamento de carga e é fácil lidar com a relocalização de servidores. Por outro lado temos o aumento de complexidade na infraestrutura, a necessidade de um protocolo para aceder ao Serviço de Nomes, bem como as primitivas para o registo de procedimentos com o servidor.

RPC – Ligação Dinâmica

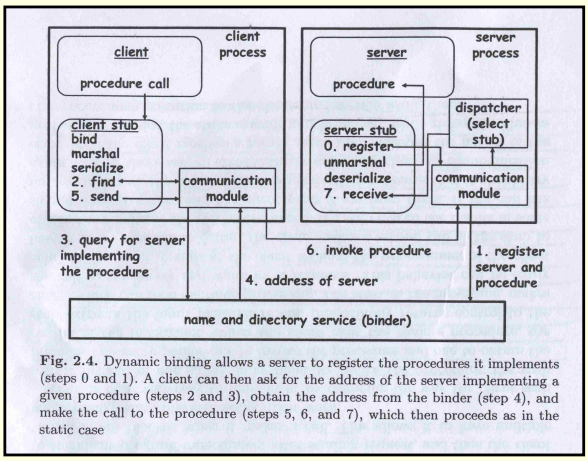


Fig. 2.4. Dynamic binding allows a server to register the procedures it implements (steps 0 and 1). A client can then ask for the address of the server implementing a given procedure (steps 2 and 3), obtain the address from the binder (step 4), and make the call to the procedure (steps 5, 6, and 7), which then proceeds as in the static case

• Todas estas operações estão escondidas ao programador. São executadas pelos "stubs".

• O Serviço de Nomes usa a especificação IDL dos procedimentos para efectuar a ligação dinâmica. Usa as assinaturas dos procedimentos, embora a ligação possa ser efectuada com base noutros critérios.

• O programador não está necessáriamente ao corrente sobre se a ligação é estática ou dinâmica. A decisão é tomada na altura do desenho do sistema e na altura da criação dos "stubs".

RPC – Heterogeneidade

- Através de "stubs" mais complexos, é possível o desenvolvimento de clientes em plataformas diferentes das dos servidores
- Existem muitas plataformas diferentes onde os clientes/servidores podem ser executados
- Existem muitas linguagens diferentes para implementar os clientes/servidores

• Os "stubs" podem esconder não só a distribuição, mas também a heterogeneidade.

• Os PCs da altura não tinham poder de processamento para correr os servidores, só os clientes. Através deste mecanismo dos RPCs foi possível o afastamento dos mainframes, mantendo os servidores a funcionar, por exemplo, em ambientes UNIX, e os clientes em DOS.

RPC – Heterogeneidade

- Conjuntos de “stubs” cliente/servidor para cada combinação possível de linguagem/plataforma (2*N*M “stubs”)
- Usar uma forma de representação intermédia apenas para a qual os clientes e os servidores necessitam de saber traduzir (N+M “stubs”)
- IDL para definir as interfaces e o mapeamento entre as linguagens de programação e as representações intermédias

• N plataformas cliente e M plataformas servidor.

• A segunda forma é muito mais eficiente, uma vez apenas é necessário desenvolver “stubs” cliente/servidor para as plataformas correspondentes.

• Usando IDL permitia aos clientes/servidores ignorar as especificidades da arquitectura da máquina, das linguagens de programação e dos tipos de dados usados. O IDL define as interfaces de especificação e o formato dos dados trocados entre os clientes e os servidores através da rede.

RPC – Extensões

- Os RPCs convencionais têm algumas limitações
- Várias extensões aos mecanismos RPC básicos foram implementadas para responder às exigências dos sistemas de informação distribuídos
- Várias extensões aos RPCs resultaram em plataformas de MW

• Os RPCs convencionais implicam que o cliente fique bloqueado à espera de uma resposta, e, apesar de os servidores terem várias “threads” a correr ao mesmo tempo, cada uma está alocada a apenas um cliente.

•

• As extensões aos RPCs resultaram em plataformas de MW tais como Monitores Transaccionais, Monitores de Objectos, Sistemas de Filas ou “Brokers” de Mensagens.

RPC – RPCs Assíncronos

- Uma das primeiras extensões aos RPCs a suportar chamadas não bloqueantes
- O “handler” de comunicações retorna o controlo de execução ao programa cliente imediatamente após ter enviado o pedido
- O “stub” disponibiliza um ponto para efectuar a chamada ao procedimento e um ponto para obter a resposta à chamada
- Para serem realmente úteis, os RPCs Assíncronos necessitam de uma infraestrutura muito mais sofisticada do que os “stubs” disponibilizados

• Permite que um cliente efectue envie uma mensagem de pedido de um serviço sem se bloquear à espera da resposta.

• Permite que um cliente tenha vários pedidos pendentes enquanto executa outras tarefas.

• Quando o cliente efectua a chamada ao procedimento remoto, o “stub” extrai os parâmetros e retorna o controlo imediatamente ao cliente. O “stub” efectua a chamada ao servidor e espera pela resposta. Isto é conseguido com ”threads” do lado do cliente, das quais o programador do cliente não se apercebe. Mais tarde, o cliente efectua uma segunda chamada ao “stub” para obter os resultados. Se a chamada já tiver retornado, o “stub” coloca os resultados numa estrutura de dados partilhada com o cliente, caso contrário, é retornada uma indicação para o cliente tentar novamente mais tarde. Em caso de erro, um código de erro é retornado ao cliente.

• Como menos informação acerca da ligação é mantida e as respostas estão dissociadas dos pedidos, pode ser mais difícil recuperar de uma falha nos RPCs Assíncronos.

RPC – Distributed Computing Environment (DCE)

- Disponibilizado pela Open Software Foundation (OSF), ainda é usado por várias plataformas MW e produtos de integração empresariais
- DCE resulta de uma tentativa falhada de criar um “standard” RPC
- A apresentação do CORBA como “standard” beneficiou do erros do DCE, só apresenta uma especificação, não uma implementação

• DCE disponibiliza não só uma especificação completa de como os RPCs deveriam funcionar, mas uma implementação standard. Direccionado aos vendedores, para que pudessem usar e extender conforme necessário aos seus produtos.

• A esperança era que todos os produtos que usassem o DCE fossem compatíveis entre si.

• A OSF poderia ter razão, uma vez que várias implementações “CORBA-compliant” não são compatíveis entre elas.

RPC – DCE

- A plataforma DCE disponibiliza RPCs e outros serviços úteis ao desenvolvimento de sistemas de informação distribuídos
- Estes serviços incluem:
  1. Serviço de directórios de células
  2. Serviço de tempo
  3. Serviço de "threads"
  4. Serviço de ficheiros distribuídos
  5. Serviço de segurança

• O Serviço de directórios de células é um serviço de nomes e directórios sofisticado, usado para criar e gerir domínios RPC que podem coexistir pacificamente na mesma rede.

• O Serviço de tempo disponibiliza sincronismo temporal a todos os nós.

• O Serviço de "threads" disponibiliza "threads" e suporta múltiplos processadores.

• O Serviço de ficheiros distribuídos permite que vários programas partilhem dados num ambiente DCE.

• O Serviço de segurança disponibiliza autenticação e comunicações seguras entre clientes e servidores.

RPC – Distributed Computing Environment (DCE)

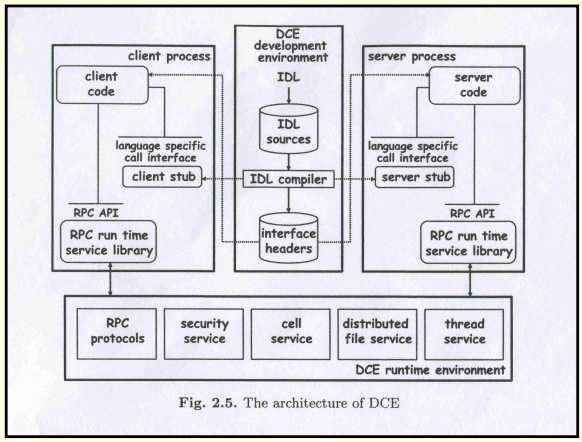
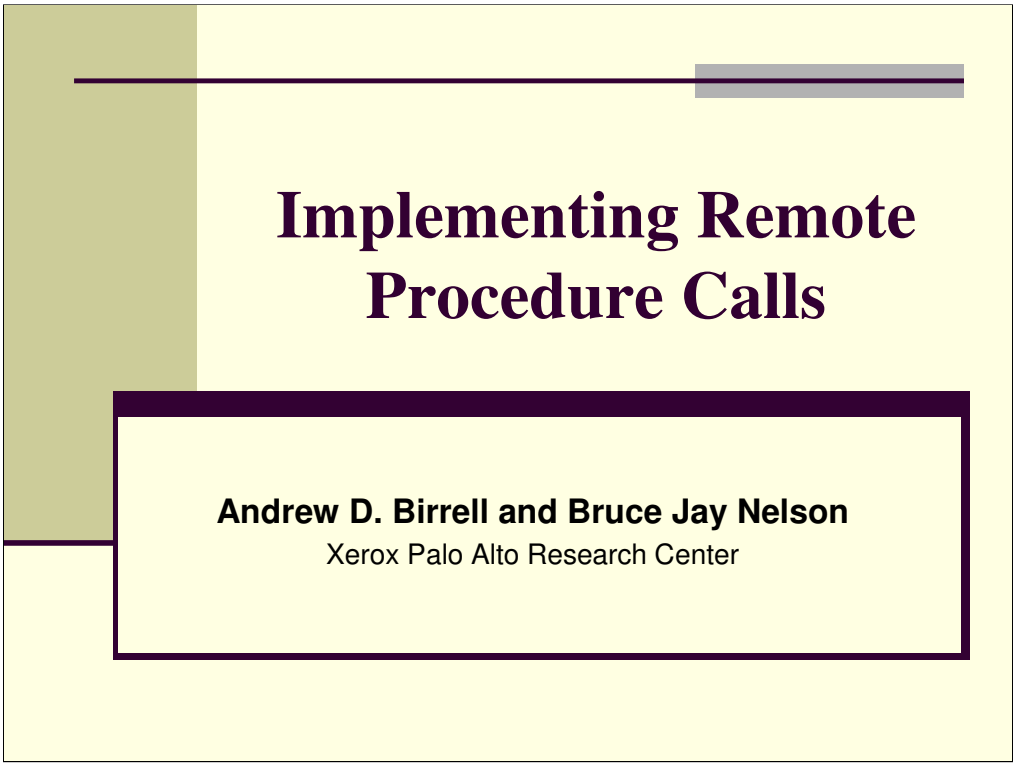


Fig. 2.5. The architecture of DCE



- Publicado em 1984.
- Paper original que deu origem aos RPCs.
- Descreve um pacote que disponibiliza uma infraestrutura para RPCs no âmbito do projecto Cedar.
- Aborda as decisões tomadas pelos autores.
- Descreve a estrutura global do mecanismo RPC.
- Descreve a infraestrutura para efectuar a ligação aos clientes RPC.
- Descreve o protocolo de comunicações ao nível de transporte.
- Descreve alguma análise de desempenho.
- Inclui a descrição de optimizações usadas para maximizar a performance e reduzir a carga nos servidores com muitos clientes.

Ambiente envolvendo o projecto

- O pacote RPC construído foi desenvolvido para ser usado no ambiente de programação Cedar
- A maioria dos computadores eram Dorados, com um espaço de endereçamento virtual de 24 bits e um disco de 80 MB
- As comunicações eram estabelecidas sobre a rede interna de investigação da Xerox, maioritariamente Ethernet a 3 Mbps e certos troços a 10 Mbps
- As redes ligavam-se entre elas via linhas dedicadas e/ou ligações por satélite com ritmos de transferência entre 4800 e 56000 bps
- Programação de alto-nível essencialmente em Mesa mas também algum Smalltalk e InterLisp

• O Cedar era um grande projecto que se concentrava em desenvolver um ambiente de programação poderoso e adequado à construção de sistemas e programas experimentais. Desenhado para “workstations” mono-utilizador, embora também também tenha sido usado em servidores.

•

•

• A maioria do tráfego ocorria nas redes locais daí que o menor ritmo de transferência nas ligações internet não era um problema. As redes raramente se encontravam sobrecarregadas.

• Não havia linguagem assembly para os Dorados.

Decisões fundamentais

- A escolha entre usar a chamada a procedimentos ou a troca de mensagens é indiferente
- A hipótese de emular o uso de um espaço de endereçamento partilhado entre computadores foi descartada
- A semântica das chamadas a procedimentos remotos devem ser o mais semelhante possível às locais
- Inexistência de “timeouts” nas chamadas

• A decisão pela utilização de chamada a procedimentos, deveu-se à predominância deste mecanismo na linguagem MESA, usada na Xerox.

• Não sabiam se seria exequível, mas dois problemas foram previstos, a incerteza sobre se seria possível integrar na MESA a representação de endereços remotos sem efeitos malignos, e a incerteza sobre seria atingida eficiência aceitável com os custos acrescidos.

• Não seguir este princípio levaria às mesmas complexidades de outros pacotes de comunicações e protocolos que os tornaram dificeis de usar.

• Chamadas locais não têm “timeouts” e as suas linguagens continham mecanismos para abortar uma actividade que faziam parte dos seus mecanismos de processamento paralelo. Implementar “timeouts” iria complicar o mecanismo RPC desnecessariamente.

Estrutura

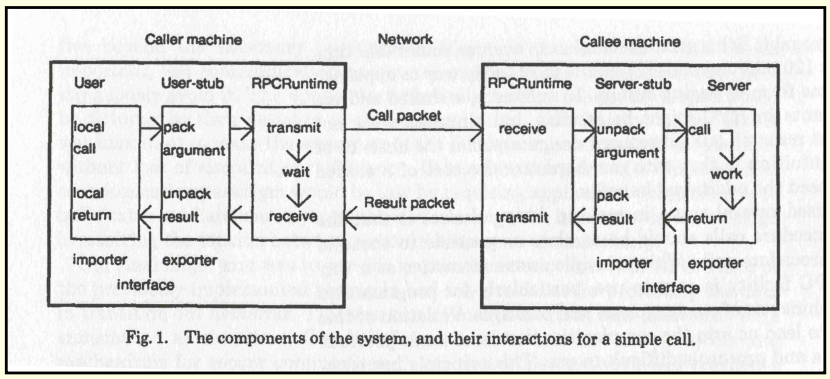


Fig. 1. The components of the system, and their interactions for a simple call.

• O RPCRuntime é responsável pelas retransmissões, “acknowledments”, encaminhamento de pacotes e encriptação.

• O RPCRuntime faz parte do sistema Cedar.

• Aqui o compilador IDL é um programa chamado Lupine, que gera os “stubs”.

• A geração da interface é feita recorrendo aos módulos de interface MESA.

Ligação - Nomes

- Uma interface a exportar é composta por um Tipo e uma Instância
- O Tipo especifica, com algum nível de abstracção, qual o tipo de interface que o cliente espera que o servidor implemente
- A Instância especifica qual é a implementação particular da interface abstracta desejada

•

• Uma abstracção de um servidor de email.

• Um implementação específica de um servidor de email entre vários.

Ligação – Localizar o servidor

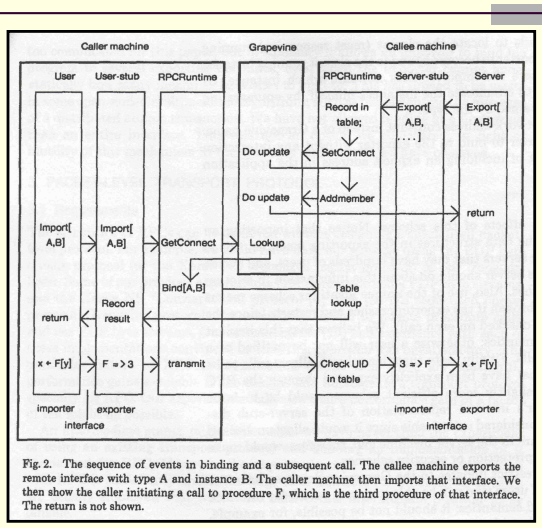


Fig. 2. The sequence of events in binding and a subsequent call. The callee machine exports the remote interface with type A and instance B. The caller machine then imports that interface. We then show the caller initiating a call to procedure F, which is the third procedure of that interface. The return is not shown.

• Para a ligação RPC foi usado o sistema de BDs Grapevine distribuída por vários servidores na rede interna da Xerox.

• Para um servidor dar a conhecer uma interface aos seus clientes remotos, invoca um procedimento no "stub" servidor, que por sua vez invoca um outro procedimento no RPCRuntime, que actualiza o Grapevine com o nome da interface (tipo e instância) juntamente com o procedimento que lhe está associado. O RPCRuntime do servidor mantém uma tabela com todas as interfaces exportadas (nome da interface, procedimento exportado, identificador único de 32 bits).

• Quando um utilizador quer usar um procedimento remoto, invoca um procedimento no seu "stub" utilizador indicando o tipo e instância do interface desejado, que por sua vez invoca um procedimento no RPCRuntime que determina o endereço de rede do servidor ao consultar o Grapevine. O RPCRuntime do utilizador efectua um RPC ao RPCRuntime do servidor para determinar a informação de ligação associada ao tipo e instância do interface desejado. Se o servidor estiver a exportar o interface, então o identificador único na tabela do RPCRuntime é devolvido e, consequentemente, o utilizador já pode executar o seu procedimento remoto. O "stub" do utilizador guarda o endereço de rede, o identificador e index na tabela para usar em chamadas futuras.

• O "stub" utilizador usa o identificador único e o index na tabela do interface ao efectuar a chamada remota. O RPCRuntime usa o index recebido para confirmar a sua existência na tabela e passar o pacote da chamada à implementação do procedimento.

• O controlo de acesso ao Grapevine garante que não pode haver utilizadores mal-intencionados a exportar interfaces.

• Usando um protocolo seguro, é possivel implementar autenticação bi-direccional entre os utilizadores e o Grapevine, de forma a garantir que os serviços anunciados no Grapevine são fidedignos.

• Já era disponibilizada ligação dinâmica e estática.

Protocolo de Transporte ao nível do pacote – Chamadas Simples

- Uma chamada só é interrompida se o servidor "crashar" ou houver uma falha nas comunicações
- Apenas um pacote por cada pedido de invocação e um por cada resultado da invocação

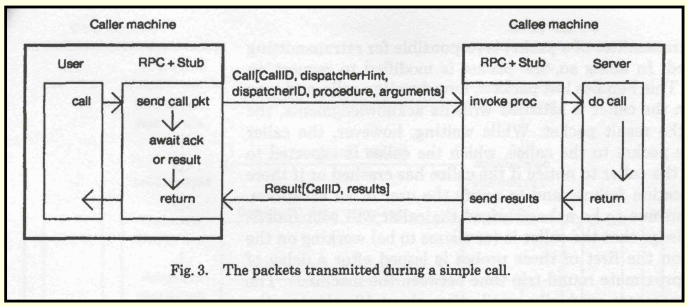


Fig. 3. The packets transmitted during a simple call.

• Após algumas experiências, decidiram que obteriam ganhos de desempenho se fosse desenhado e desenvolvido um protocolo de transporte específico para RPCs.

• O custo de iniciar, terminar e manter informação sobre o estado de uma ligação era considerável, e, se para grandes transferências de dados isso se tornaria irrelevante, no caso dos RPCs, em que o volume de dados é curto mas muitas vezes efectuado, com muitos clientes, isso tornar-se-ia muito dispendioso.

• Para imitar o funcionamento das chamadas locais, se o servidor encalhar num "loop" infinito ou num "deadlock" a chamada nunca termina.

• Tornar eficientes as comunicações. A máquina que transmite o pedido é responsável por retransmiti-lo até obter uma resposta. A resposta a um pedido é "acknowledge" suficiente para o pedido e um segundo pedido é "acknowledge" suficiente para a resposta ao pedido, uma vez que cada pedido tem um identificador único.

• Cada ligação entre utilizador e servidor mantêm-se apenas entre o pacote de pedido e o pacote de resposta, estes dois pacotes são o início e o fim da ligação.

• O identificador único de cada pedido, obtido durante o processo de ligação, é informação suficiente acerca do estado da ligação, evitando os "handshakes" necessários de outros protocolos de comunicação.

Protocolo de Transporte ao nível do pacote – Chamadas Complexas

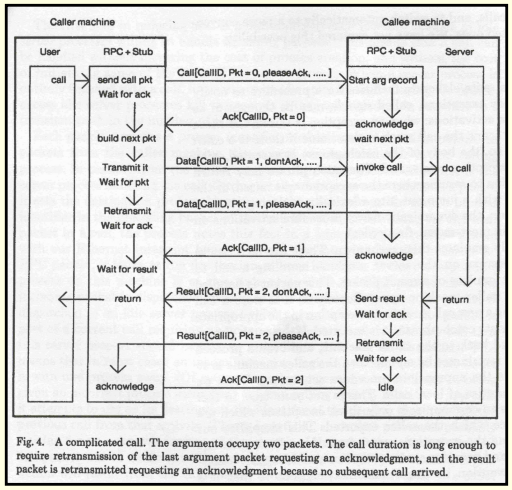


Fig. 4. A complicated call. The arguments occupy two packets. The call duration is long enough to require retransmission of the last argument packet requesting an acknowledgment, and the result packet is retransmitted requesting an acknowledgment because no subsequent call arrived.

• Um pacote sonda é enviado de tempos a tempos (5 em 5 minutos), ao qual o servidor tem de responder, de forma a que o utilizador verifique se há problemas nas comunicações, ou se o servidor "crashou". Não detecta "dead-locks" nem "loops"infinitos.

• Se o pedido é demasiado grande para caber num só pacote, então o pedido é dividido em pacotes, e todos menos o último têm de ser "acknowledged" antes de ser enviado o próximo. Nesta situação, existe um número de sequência associado à chamada para permitir a eliminação de pacotes em duplicado.

Tratamento de Excepções

- Inspirado na linguagem MESA
- O Servidor devolve um pacote-excepção em vez de um pacote-resultado quando "apanha" uma excepção
- Se o RPCRuntime do utilizador tiver um tratamento para a excepção, executa-a, caso contrário, devolve o processamento ao utilizador com indicação da excepção
- Além das excepções, o RPCRuntime do utilizador pode devolver-lhe uma excepção "call-failed" no caso de problemas de comunicação

• Com "signals" e "catchs".

Processos

1. O processo utilizador constrói o 1º pacote, inclui o seu id de processo e um id de processo destino plausível, envia-o e bloqueia-se à espera da resposta
2. No servidor, o "interrupt handler" recebe o pacote e notifica um processo servidor adequado
3. O processo servidor executa a chamada, constrói e devolve o pacote resposta incluindo o id do processo utilizador
4. O "interrupt handler" no utilizador recebe a resposta e devolve-a ao processo utilizador
5. O processo utilizador fica agora a conhecer o id do processo servidor e pode usá-lo nos pacotes subsequentes da chamada ou para iniciar uma chamada posterior

• O processamento em paralelo está "built-in" nas "features" da Mesa e do Cedar.

• Para reduzir os custos de criar e fazer "swaping" de processos, mantêm-se no servidor uma série de processos "idle" dispostos a tratar pacotes recebidos, que regressam ao estado "idle", em vez de morrer, depois de tratar cada pedido.

• Em chamadas simples não há lugar à criação de processos, apenas trocas de processos que se encontram "idle".

• São disponibilizadas funcionalidades para encriptar as chamadas e os resultados. Estas funcionalidades usam o Grapevine como Serviço de Autenticação ou Centro de Distribuição de Chaves e usam o DES (Data Encryption Standard). A autenticação é bi-direccional.

Java Remote Method Invocation – Distributed Computing for Java

SUN Developer Network White Paper

Visão geral dos RMIs

- Disponibiliza um modelo simples e directo para o desenvolvimento de aplicações distribuídas usando objectos Java
- Ligam-se a sistemas existentes usando o interface Java de métodos nativos JNI
- Ligam-se a BDs relacionais usando a "package" JDBC
- As combinações RMI/JNI e RMI/JDBC permitem usar os RMIs para comunicar com servidores usando outras linguagens

Vantagens dos RMIs

- OO: Objectos como argumentos e como valores de retorno sem complexidade acrescida
- Mobilidade de Comportamento: O cliente pode importar os objectos que implementam comportamentos dos servidores
- Segurança: Usam os mecanismos de segurança embebidos no Java
- Fácil de escrever/usar: Um interface remoto é um interface Java
- Ligação a sistemas existentes: Usando RMI/JNI é possível escrever clientes em Java que se ligam a servidores existentes, e reescrever também os servidores em Java. Usando JDBC é possível a ligação a BDs relacionais existentes

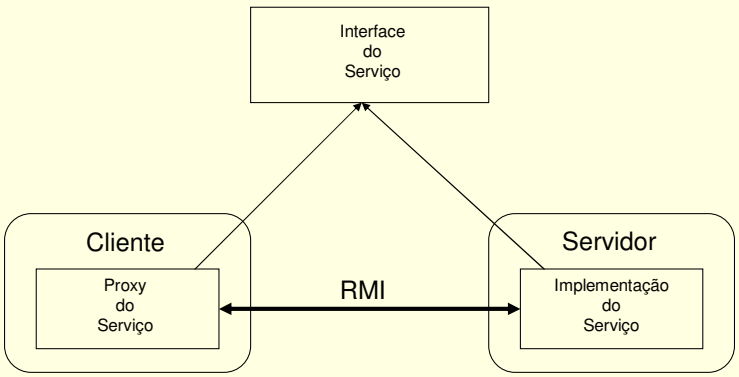
•

• Quando é feita a primeira chamada, o cliente pode importar o objecto que implementa o interface disponibilizado pelo servidor. Se a implementação mudar, o servidor pode passar a disponibilizá-la aos clientes. O método é executado localmente ao cliente e aliviar a carga do servidor sem ser necessário descarregar mais software.

Vantagens dos RMIs

- Portabilidade: Qualquer sistema RMI corre em qualquer máquina virtual Java
- “Garbage Collection” distribuída: Conseguem reaproveitar objectos no servidor que não são usados por mais nenhum cliente remoto
- Computação Paralela: “Multi-threading” suportado através da recorrência às “threads” Java para melhorar o processamento concorrente dos pedidos dos clientes
- Parte da solução “Java Distributed Computing”: Faz parte do núcleo da plataforma Java JDK 1.1, logo, presente em qualquer máquina virtual Java 1.1

Estrutura de Implementação RMI



• O RMI suporta duas classes que implementam o mesmo interface. Uma corre no servidor, sendo a implementação do comportamento. A outra corre no cliente e serve como proxy do serviço remoto.

• Quando um cliente quer invocar um método remoto, importa do servidor o objecto que implementa o método, tal como especificado pela interface em Java. Se for a primeira vez que o RMI runtime do cliente vê a implementação do método, então pede uma cópia ao servidor.

• O RMI usa o mecanismo standard do Java de serialização de objectos para passar objectos.

• Permite a passagem de endereços de objectos remotos como argumentos.

• Outra possibilidade é a de enviar como argumento um objecto Task, que irá ser executado num servidor remoto, típicamente muito mais performante, e obter como resposta o resultado da execução remota (método run() do objecto Task).

Arquitectura

- Ao exportar um servidor, o seu tipo de referência é definido
- Quando um cliente recebe uma referência de um servidor, o RMI descarrega um "stub" que traduz as chamadas a essa referência em chamadas remotas ao servidor
- O "stub" efectua o "marshall" dos argumentos e serializa-o para o servidor
- No servidor, o "skeleton" recebe e "unmarshall" a chamada, e invoca o método com os argumentos da chamada
- O percurso inverso é semelhante, sendo devolvido o resultado da invocação do método ou uma excepção

• Diferentes tipos de servidores têm diferentes semânticas de referência.

• O "stub" efectua o "marshalling" dos argumentos para o método, usando a serialização de objectos, e envia a pedido "marshalled" para o servidor

Segurança

- O RMI disponibiliza a utilização de sockets de diversos tipos, incluíndo sockets encriptados
- O RMI usa o mecanismo standard do Java que usa o object SecurityManager para tratar das questões de segurança, obrigando à instalação de um SecurityManager antes de exportar um objecto do servidor ou invocar um método de um servidor

-
- O objecto RMISecurityManager irá impedir que implementações descarregadas tenham qualquer tipo de acesso aos computadores locais.
- Também é possível a implementação de gestores de segurança customizados.
- Para clientes atrás de “firewalls” poderem comunicar com servidores remotos da maneira mais rápida possível é possível usar o obejcto UnicastRemoteObject, que efectua as seguintes tentativas:
    1. Usar “sockets” para comunicar directamente com o porto do servidor;
    2. Se falhar, construir um URL com o “host” do servidor e com o porto e efectuar um pedido HTTP POST ao URL construído, enviando a informação para a “skeleton” no corpo do POST. Se for bem sucedido, a resposta do “skeleton” ao “stub” do cliente é o resultado do POST;
    3. Se também falhar, construir um URL com o host do servidor e o porto 80, o porto HTTP, usando um script CGI que irá enviar o pedido RMI ao servidor.
- Se alguma técnica funcionar passa a ser usada com o servidor, caso contrário, o RMI falha.