Luca Cabibbo

Architettura dei Sistemi Software

chiede nel 40% degli esami

Invocazione remota

dispensa asw430

ottobre 2024

The core idea of RPC is to hide the complexity of a remote call. Many implementations of RPC, though, hide too much.

Sam Newman

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 23, Invocazione remota
- Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. **Distributed Systems**: Concepts and Design, fifth edition. Pearson, 2012.
- Tanenbaum, A.S. and Van Steen, M. **Distributed Systems**: Principles and Paradigms, second edition. Pearson, 2007.
- Richardson, C. **Microservices Patterns**: With examples in Java. Manning, 2019.

- Obiettivi e argomenti

- Obiettivi
  - introdurre l'invocazione remota e discutere la sua implementazione
  - discutere la semantica dell'invocazione remota – e come differisce dalla semantica dell'invocazione locale
  - discutere alcune varianti dell'invocazione remota
- Argomenti
  - invocazione remota
  - implementazione dell'invocazione remota
  - semantica dell'invocazione remota
  - ulteriori aspetti e varianti dell'invocazione remota
  - discussione

Dei sistemi distribuiti abbiamo detto che ci sono diverse modalità di comunicazioni tra componenti di un sistema distribuito, le principali sono l'invocazione remota e la comunicazione asincrona.

* Invocazione remota

- L'*invocazione remota* è una delle principali astrazioni di programmazione distribuita implementate dal middleware
  - è una versione "remota" (distribuita) dell'invocazione "locale" di operazioni
  - consente interazioni di tipo client/server (di tipo richiesta/risposta e sincrone) tra componenti
  - è implementata da molte tecnologie di middleware
    - ad es., RPC, RMI, nelle tecnologie a componenti e nelle tecnologie a servizi
  - discutiamo gli aspetti fondamentali dell'invocazione remota – ma senza far riferimento a tecnologie specifiche

Un componente ha il ruolo di client e invoca un'operazione di un altro componente nel ruolo di server. Fa una richiesta e si aspetta una risposta in modalità sincrona, esattamente come Nell'invocazione locale: il chiamante viene bloccato fino a quando il chiamato non termina la sua computazione.

È una versione remota dell'invocazione locale di operazioni presente in tutti i linguaggi di programmazione. Ad esempio in C posso definire funzione, in Java posso definire metodi, e posso chiamare queste funzioni o metodi all'interno di altre funzioni o metodi. Chiamare e invocare sono sostanzialmente sinonimi. Normalmente nei linguaggi di programmazione questa invocazione è locale, quindi tutte le funzioni vivono all'interno dello stesso processo e quindi la chiamata viene gestita localmente all'interno del processo. Invece in un sistema distribuito ci sono molti processi che sono fra loro remoti/distribuiti. Quindi questa è una astrazione per invocare una funzione di un processo da una funzione di un altro processo. Astrazione nel senso che il linguaggio di programmazione di per sè non offre questa opzione, astrazione rispetto a quello che succede normalmente.

Chiamata di procedure remote (RPC)

Remote procedure call (procedure = funzione/metodo/operazione genericamente)

- La *chiamata di procedure remote* (*Remote Procedure Call* o *RPC*) è un'astrazione di programmazione distribuita fondamentale
  - è una delle prime soluzioni di middleware realizzate
  - consente a un componente (*client*) di chiamare (invocare) una "procedura" (*operazione*) di un componente remoto (*server*), affinché questi esegua l'operazione richiesta

Componente nel ruolo di …può voler dire QUESTA interazione
hanno questo ruolo, non è detto che non possa essere altrimenti

Remote Procedure Call

- *RPC* (*Remote Procedure Call*)
  - In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.
    That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. […]
    In the object-oriented programming paradigm, RPC calls are represented by remote method invocation (RMI).
    [Wikipedia, 2020]

In un diverso processo (perché ogni processo ha una zona di memoria separate, non c'è condivisione di memoria)

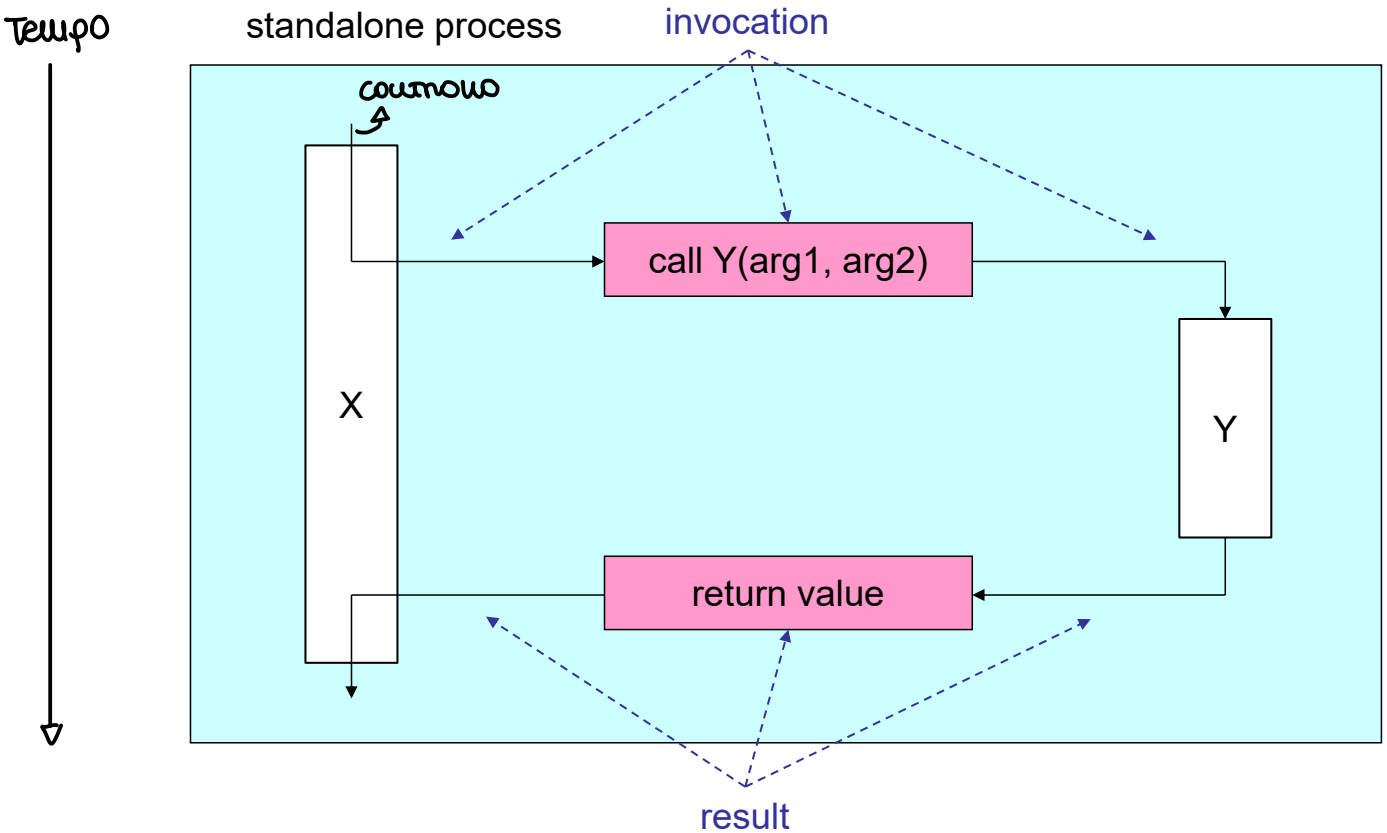
RMI è una RPC in un contesto di programmazione orientato agli oggetti, noi usiamo
RMI sotto intendendo anche RPC, sorvolando sulle sottili differenze

Invocazione di metodi remoti (RMI)

- L'*invocazione di metodi remoti* (*Remote Method Invocation* o *RMI*) è un'estensione OO di RPC
  - un servizio di comunicazione tra *oggetti distribuiti* o *oggetti remoti* – componenti distribuiti realizzati con tecnologie a oggetti e in esecuzione in processi separati
  - a parte l'adozione di un modello a oggetti, un'invocazione di metodo remoto (in RMI) corrisponde essenzialmente a una chiamata di procedura remota (in RPC)
  - nel seguito parleremo semplicemente di "operazioni" e di "invocazioni" "remote"

Un'invocazione "locale"

Due aspetti fondamentali dell'invocazione: la chiamata e la restituzione di un valore che è chiamato risultato, insieme alla cessione del controllo



Nell'invocazione remota io vorrei il seguente comportamento: ci sono due processi diversi, client e server. Durante l'esecuzione di X nel processo client vorrei che venisse chiamata una funzione Y nel processo server, tramite uno scambio: l'invio di una invocazione e la restituzione di una risposta. Questo normalmente non è possibile perché il linguaggio di programmazione non lo consente di default, e sarà responsabilità del middleware.
Uno dei problemi è che le chiamate non possono attraversare i confini tra processi, e mi serve quindi un meccanismo di comunicazione tra processi. Uno dei metodi è lo scambio di messaggi in rete. Non è l'unico metodo, ci sono per esempio anche le pipe che però sono molto più vincolanti. Supponiamo quindi di voler sfruttare lo scambio di messaggi in rete. Quello che possiamo fare per attraversare questi confini tra processi è utilizzare un messaggio di richiesta che codifichi l'invocazione e un messaggio di risposta che codifichi la risposta appunto. Il processo client quindi quando vuole invocare l'operazione crea un messaggio di richiesta che codifica l'invocazione, lo comunica in rete al processo server mediante una comunicazione interprocesso, e questo interpreta il messaggio di richiesta, esegue l'operazione, produce un risultato, lo codifica in un messaggio di risposta e lo trasmette in rete. Questo viene ricevuto dal processo client che estrae il risultato e lo utilizza, per poi proseguire.

Nota la terminologia:
Richiesta
Risposta
Invocazione
Risultato

Invocazione e risultato sono termini dell'invocazione. Parliamo di richiesta e risposta invece quando usiamo dei messaggi per codificare invocazione e risultato. Sono la loro codifica. Quindi richiesta e invocazione NON sono sinonimi, risposta e risultato NON sono sinonimi (quindi non usarli come tali), sono uno la codifica dell'altro.

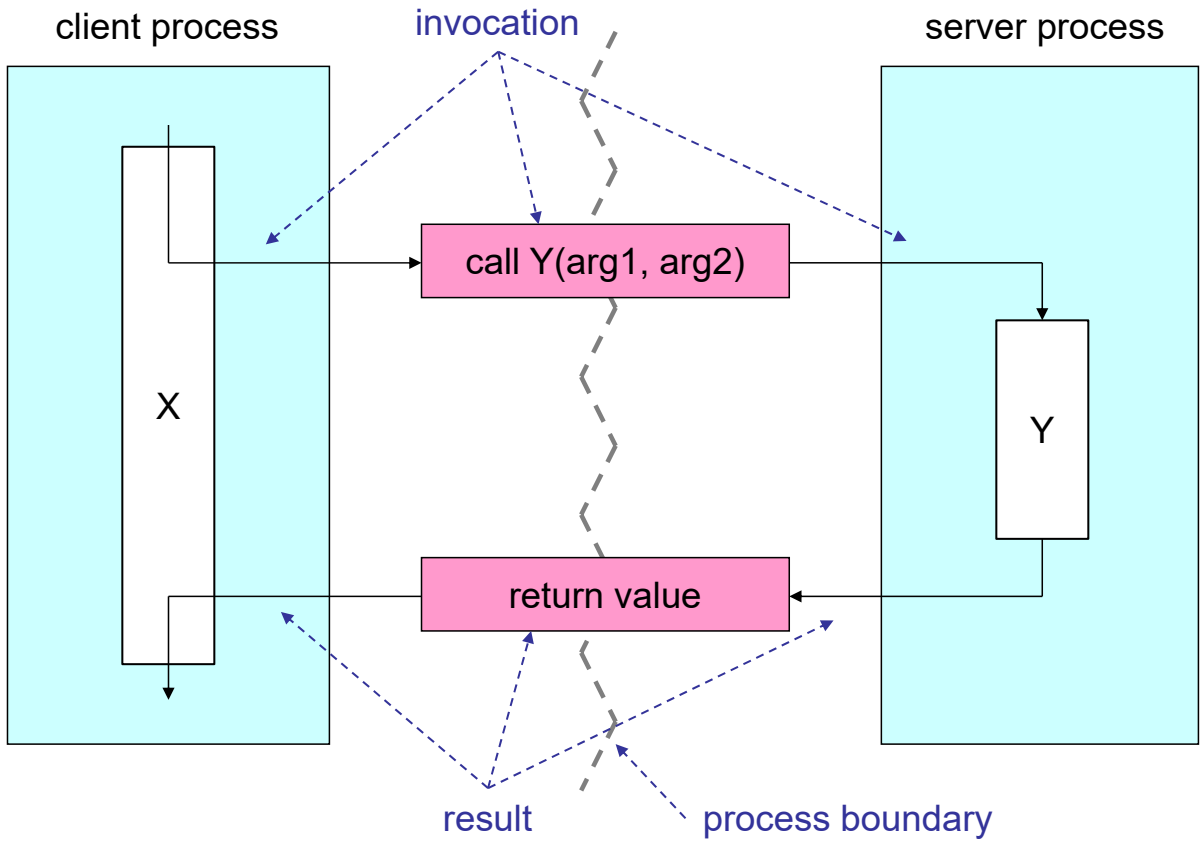
Richiesta e risposta sono MESSAGGI che vengono utilizzati per implementare il protocollo di invocazione remota, e codificano l'informazione dell'invocazione e del risultato.

Nota inoltre che nell'esempio subito sopra ci sono due processi che fungono da client e da server. In realtà ogni processo è formato da componente e connettore. Componenti fanno invocazioni (e rimangono all'oscuro dello scambio di messaggi), connettori scambiano messaggi.

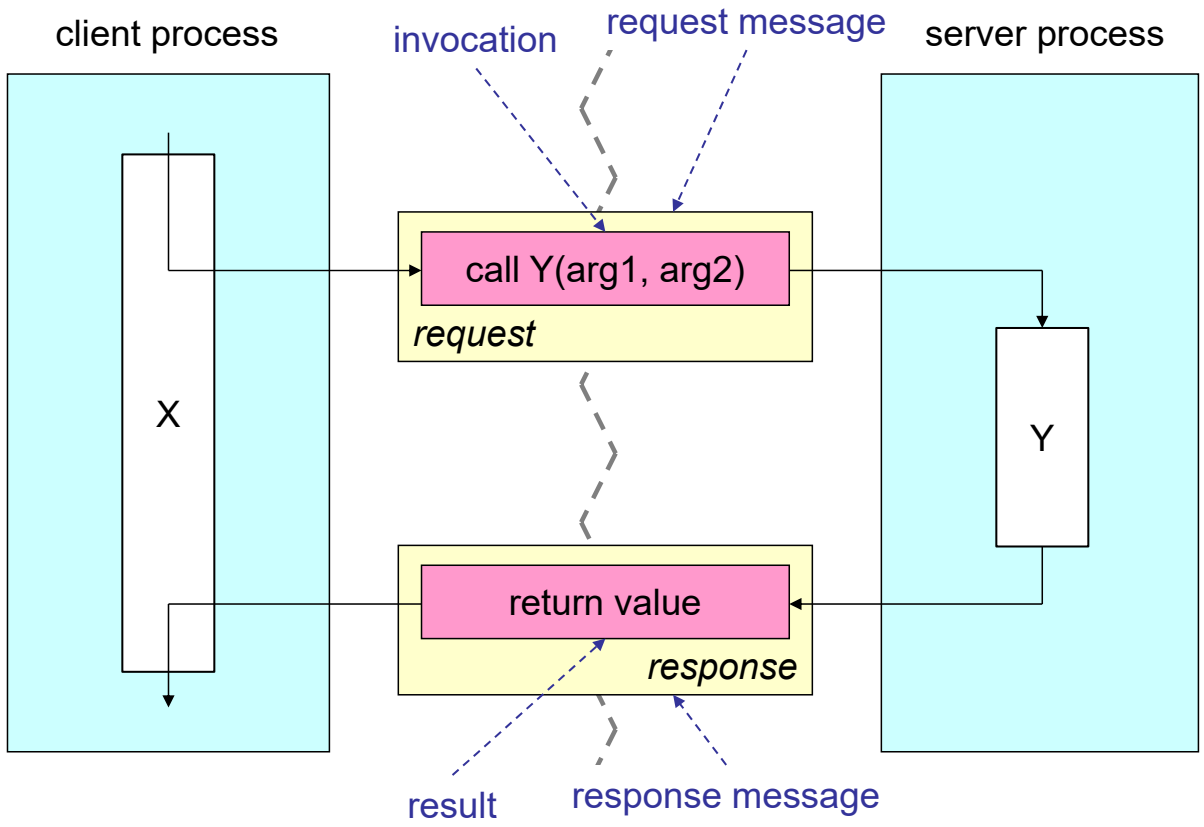
Vedremo che in pratica le chiamate RPC vengono scritte sintatticamente come se fossero invocazioni locali, e in realtà sono invocazioni locali dirette però a dei proxy.

I componenti, client e server, non sanno nulla dei messaggi, che sono gestiti, formati, decifrati dai proxy.

Un'invocazione remota



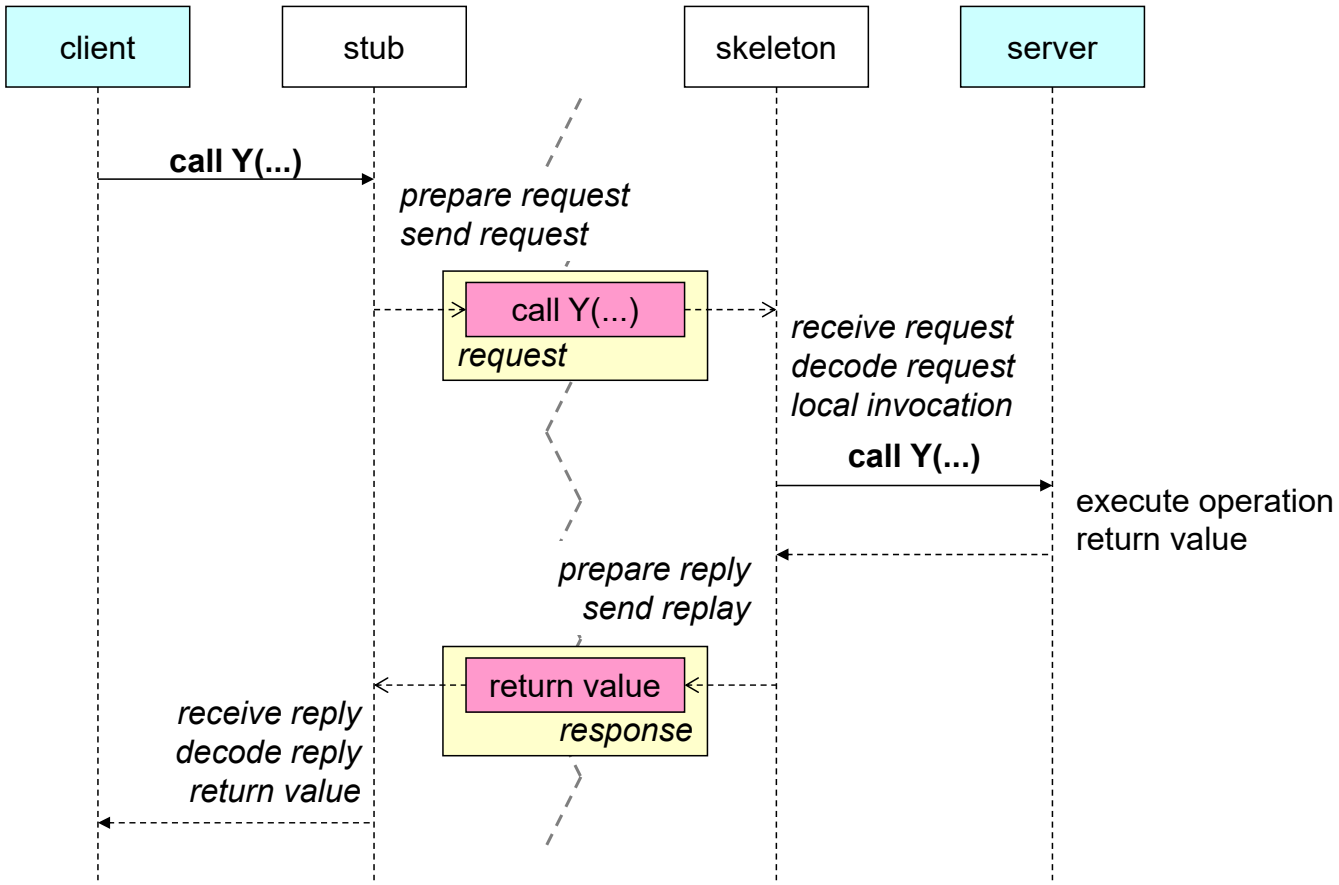
Un'invocazione remota

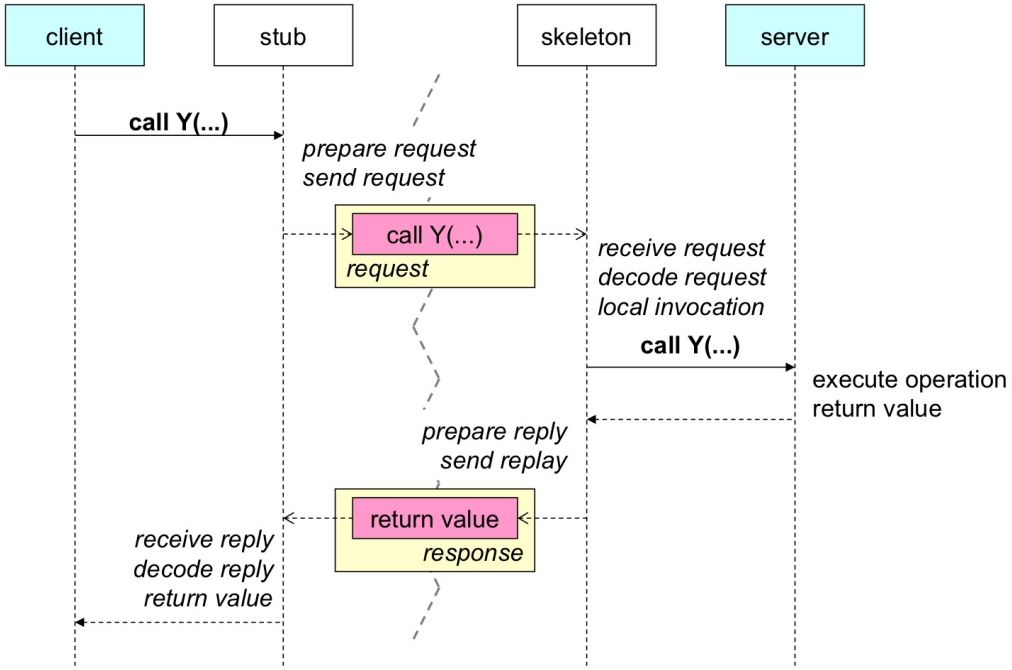


Paradigma dell'invocazione remota

- Un'invocazione remota
  - sintatticamente, l'operazione remota **Y** viene chiamata da **X** come se fosse un'invocazione locale
  - in realtà, l'operazione **Y** vive nel processo server – che è separato dal processo client in cui vive **X**
  - l'esecuzione di un'invocazione remota comprende certamente le attività consuete – legame dei parametri, esecuzione dell'operazione e restituzione dei risultati
  - l'implementazione è basata su un protocollo richiesta-risposta – che prevede lo scambio di un messaggio di richiesta e di uno di risposta
  - il client e il server interagiscono in modo sincrono

* Implementazione dell'invocazione remota





L'implementazione di un'invocazione remota prevede che fra il componente client e il componente server ci siano una coppia di proxy remoti chiamati proxy lato client STUB e proxy lato server SKELETON. Nel processo client vivono sia il componente client che il proxy lato client, nel processo server vivono sia il componente server che il proxy lato server. Il client fa una chiamata locale al proprio proxy. Il proxy riceve questa chiamata locale e in uno scenario di successo prepara il messaggio di richiesta e lo trasmette in rete verso il processo server. Quindi la prima cosa che fa lo stub è preparare la richiesta e poi inviare la richiesta. Poi farà la receive e si metterà in attesa del risultato, probabilmente con un timeout. Peró finché non arriva questa risposta lui sta fermo. Lo skeleton riceve questo messaggio di richiesta, lo decodifica quindi capisce quale è l'operazione che si vuole che sia invocata e quali sono i parametri ed effettua la chiamata al server, che è una chiamata locale. Il componente server quindi esegue l'operazione, calcola il risultato, e restituisce questo risultato a chi ha fatto questa chiamata locale, ovvero lo skeleton. A questo punto lo skeleton prepara il messaggio di risposta che codifica il risultato e lo trasmette indietro verso il processo client (verso il suo proxy). Lui (sempre il proxy lato client) è in attesa del messaggio di risposta, quindi sempre sotto l'ipotesi che tutto vada bene riceve il messaggio di risposta, estrae il risultato dal messaggio e lo restituisce al componente client (che lo riceve come se fosse il risultato di una chiamata locale).

Quindi fondamentale è l'uso dei proxy che si occupano della comunicazione in rete e della codifica e decodifica sia di richieste che di risposte.

Invocazione remota e proxy remoti

- Nell'invocazione remota, la comunicazione tra i componenti *client* e *server* avviene tramite un connettore realizzato come una coppia di proxy remoti
  - *stub* – o *proxy lato client*
  - *skeleton* – o *proxy lato server*
  - un *proxy* [GoF] fornisce un surrogato o un segnaposto per un altro oggetto – per controllarne l'accesso
  - i proxy remoti nascondono al programmatore il fatto che la comunicazione sia distribuita

Un proxy è un intermediario nella comunicazione tra una coppia di componenti, ne esistono diversi tipi. In particolare i proxy remoti nascondono al cliente il fatto che l'esecuzione avvenga in modo distribuito e non locale

Invocazione remota, client e server

- Prima di andare avanti, è importante comprendere la terminologia utilizzata
  - va notata in particolare la distinzione tra i seguenti termini – e le relazioni tra di essi
    - componente client e componente server
    - proxy lato client e proxy lato server
    - processo client e processo server
  - quando è scritto semplicemente "client" e "server", senza qualificazioni, bisogna intendere "componente client" e "componente server"

Componente client e processo client sono cose diverse. Il processo client CONTIENE il componente client e il proxy lato client.

Nel testo quando c'è scritto solo client e server si fa riferimento al componente client e al componente server.

Invocazione remota e proxy remoti

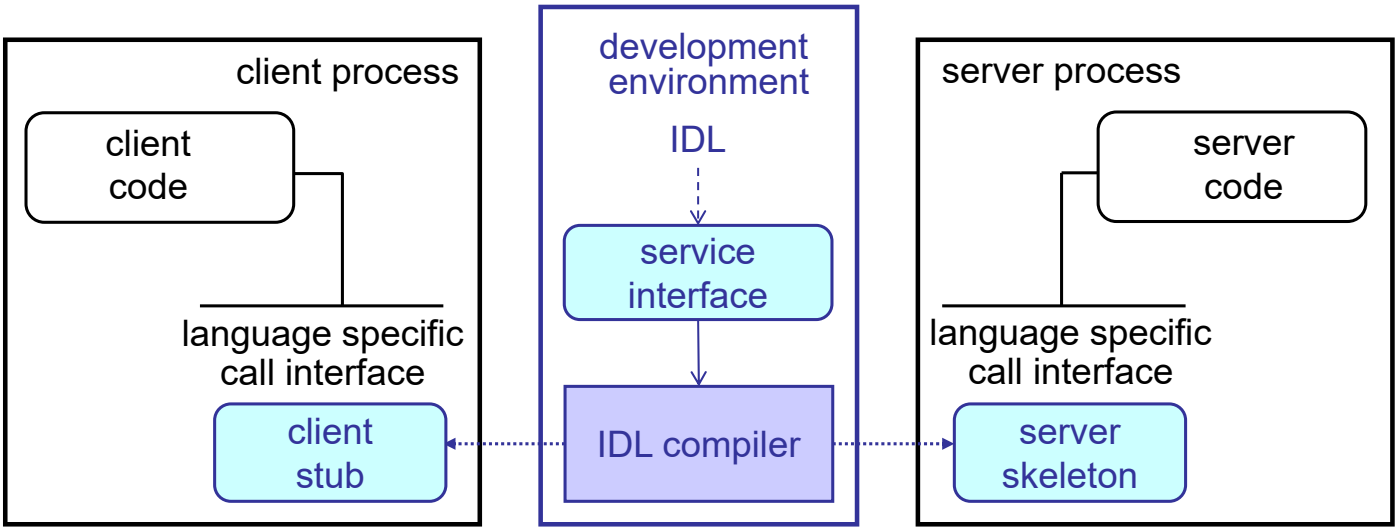
- I due proxy remoti implementano un protocollo richiesta-risposta per l'invocazione remota
  - le responsabilità principali dei proxy remoti
    - inviare e ricevere i messaggi di richiesta e risposta
    - preparare e decodificare i messaggi di richiesta e di risposta
  - con RPC e RMI, i proxy vengono generati automaticamente
  - l'implementazione di questi proxy fa in genere riferimento a dei moduli di comunicazione generali

I proxy sono generati automaticamente dal middleware, e la loro implementazione fa riferimento a delle librerie di alto livello del middleware. Infatti i proxy sono abbastanza piccoli perché delegano molto a queste librerie specifiche che si chiamano moduli di comunicazione

Generazione automatica dei proxy

- La generazione automatica dei proxy remoti è basata su
  - un linguaggio per la definizione di interfacce – *Interface Definition Language* (*IDL*)
  - un *compilatore d'interfacce*

In particolare la generazione automatica di proxy è basata sull'uso di un linguaggio per la definizione delle interfacce e di un compilatore di interfacce. Questo vuol dire che io devo invocare un servizio, il servizio deve avere una interfaccia definita in modo esplicito, definisco questa interfaccia del servizio utilizzando un linguaggio specializzato per le interfacce e poi genero i proxy dando questo linguaggio al compilatore che mi genera i due proxy.



In alcuni casi l'IDL è un linguaggio di programmazione. Ad esempio usando Java RMI il linguaggio per le interfacce è Java. Usando RPC invece il linguaggio per le interfacce è tendenzialmente C. Vedremo nelle esercitazione come sostenere l'interoperabilità attraverso IDL neutri.

Moduli di comunicazione

- L'implementazione dei proxy remoti è basata su degli opportuni *moduli di comunicazione*
  - questi moduli implementano il protocollo richiesta-risposta sottostante all'invocazione remota
  - si occupano soprattutto dello scambio dei messaggi di richiesta e di risposta
  - hanno un ruolo fondamentale nella definizione della *semantica dell'invocazione remota*

Sintatticamente io scrivo un'invocazione remota come se fosse una chiamata locale, la semantica riguarda quello che succede o può succedere

* Semantica dell'invocazione remota

- L'invocazione di un'operazione remota viene scritta, sintatticamente, allo stesso modo di un'invocazione "locale"
  - ma la semantica di un'invocazione remota è la stessa di un'invocazione "locale"?
  - no!!! ecco alcune importanti differenze
    - le operazioni sono eseguite in processi e spazi degli indirizzi separati Quindi questi processi vanno fatti comunicare, normalmente in rete
    - si possono verificare dei problemi di comunicazione
    - il legame dei parametri e dei risultati è gestito diversamente
  - discutiamo ora questi aspetti

- Problemi di affidabilità

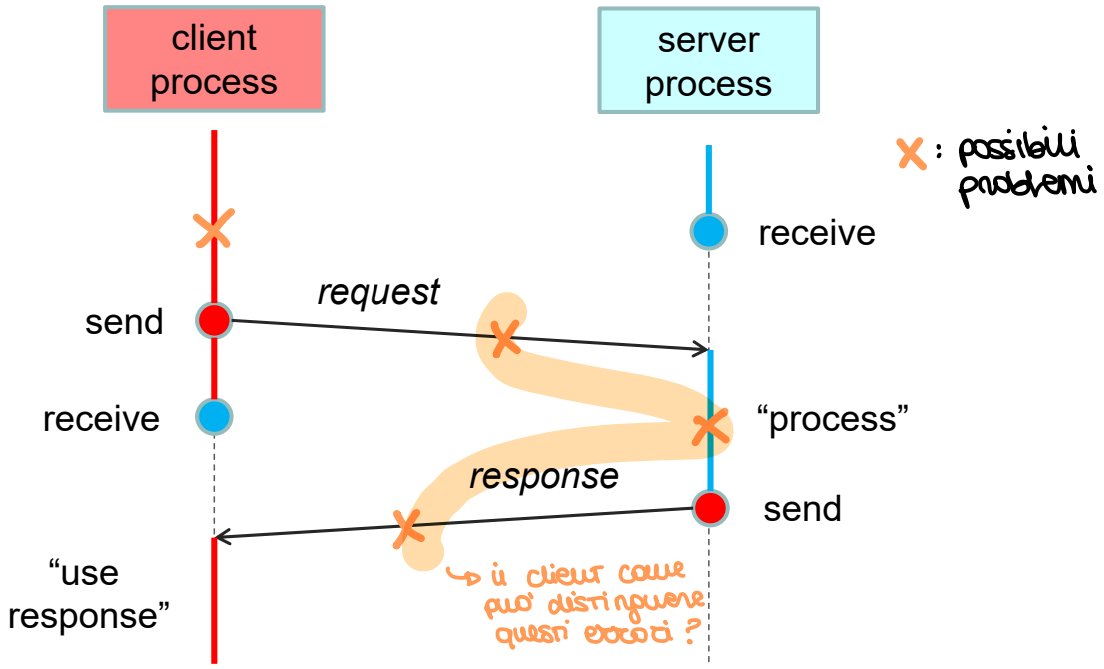
- Nei sistemi distribuiti si possono verificare diversi problemi di affidabilità Nei sistemi distribuiti si possono verificare problemi di affidabilità
  - fallimenti nella comunicazione I messaggi possono perdersi
  - fallimenti nei processi coinvolti Si può rompere uno dei nodi coinvolti
- A causa di questi (o di altri) problemi, un'invocazione remota potrebbe terminare, per un client, con la ricezione di un'*eccezione remota*
  - che cosa vuol dire?
  - chi può ricevere un'eccezione remota? e chi la può sollevare?

Il componente client che ha fatto un invocazione remota quindi può ottenere, invece di un risultato, un'eccezione remota.

Che cosa può essere successo? Cosa può capire?

Problemi di affidabilità

- Questo è uno scenario di successo nell'invocazione remota



  - in quali casi si può verificare un'eccezione remota?

Problemi di affidabilità

- Per semplicità, consideriamo solo la possibilità di perdere messaggi (richieste o risposte) Cioè ignoriamo che i processi possano andare in crash
  - l'uso di un protocollo di rete a livello di trasporto "affidabile" (come TCP) non garantisce l'affidabilità della comunicazione remota
  - i moduli di comunicazione devono considerare esplicitamente la possibilità di perdere messaggi nella gestione di un'invocazione remota
  - i moduli di comunicazione possono essere realizzati per tollerare la perdita di alcuni messaggi – la loro implementazione ha impatto sulla semantica dell'invocazione remota

- Semantica dell'invocazione remota

- Si consideri un client C che effettua un'invocazione remota di un'operazione O di un servizio (server) S
  - a tal fine, verranno scambiati alcuni messaggi in rete e verranno svolte alcune attività – tra cui, forse, l'esecuzione di O
  - alla fine, per C ci sono due possibilità
    - C riceve un *risultato*
    - C riceve la segnalazione di un'*eccezione remota*
  - a fronte di questi possibili esiti, che cosa può capire C di quanto è effettivamente successo nel sistema?
  - la semantica dell'invocazione remota (*call semantics*) riguarda appunto ciò che può succedere durante un'invocazione remota

Semantica dell'invocazione remota

- I proxy remoti – e in particolare i loro moduli di comunicazione (*mdc*) – possono gestire la perdita dei messaggi di richiesta e di risposta sulla base di tre opzioni (ciascuna opzione richiede anche le precedenti)
  - il proxy lato client ripete il messaggio di richiesta
  - il proxy lato server filtra richieste duplicate
  - il proxy lato server ritrasmette risposte

I proxy possono provare a sopperire alla perdita di messaggi in vari modi:
Se il proxy lato client non riceve una risposta alla sua richiesta può anche non fare niente. Però abbiamo visto che in caso di guasti transienti una richiesta può anche essere ripetuta (NOTA che queste decisioni spettano al proxy lato client, non al componente client, che invoca una volta sola).

Lato server, questo potrebbe cercare di individuare l'arrivo di richieste duplicate (alle richieste viene associato un identificatore) in modo da inviare la stessa risposta a due richieste identiche

Semantica dell'invocazione remota

- Queste opzioni danno luogo a un ventaglio di semantiche differenti per l'invocazione di operazioni remote
  - il proxy lato client non ripete richieste (*maybe*) (maybe cosa?)
  - il proxy lato client ripete richieste, il proxy lato server non filtra duplicati e semmai chiede al server di rieseguere l'operazione (*at least once*) (cosa?)
  - il proxy lato client ripete richieste, il proxy lato server filtra duplicati e ritrasmette le risposte (senza far rieseguire l'operazione al server) (*at most once*) (cosa?)

Iniziamo dal primo caso (il proxy lato client non ripete richieste). Nel caso in cui il proxy lato client fa una singola richiesta e si mette in attesa della risposta e se non c'è risposta solleva immediatamente un'eccezione remota possono succedere queste cose: non si perde nessun messaggio (allora vuol dire che il client alla fine riceverà il risultato, e l'operazione è stata eseguita una volta) oppure si perde il messaggio di richiesta (l'operazione quindi non viene mai eseguita, e non c'è risposta persa perché non c'è proprio risposta. Il proxy lato client riceve un timeout e solleva un'eccezione remota. In questo caso il client riceve l'eccezione e l'operazione non è stata eseguita) oppure si perde il messaggio di risposta (il server esegue l'operazione, invia la risposta che si perde, il proxy lato client non riceve la risposta e solleva l'eccezione. In questo caso il client riceve l'eccezione ma l'istruzione è stata eseguita una volta). Quindi possono succedere tre cose:
1. Il client riceve il risultato. L'operazione è stata eseguita una volta
2. Il client riceve l'eccezione remota. L'operazione è stata eseguita una volta
3. Il client riceve l'eccezione remota. L'operazione non è stata eseguita

Quindi il "maybe" si riferisce al fatto che se il client riceve un'eccezione remota, comunque FORSE l'operazione è stata eseguita. Il client non è in grado di capirlo.

Immaginiamo che l'operazione remota sia "dai mille dollari in beneficienza" quindi io provo a darli, se ricevo l'ok so che li ho dati, se non ricevo l'ok non so se li ho dati o meno. Quindi per il componente client decidere se ripetere l'invocazione non è una cosa scontata, perché potrei aver già dato i soldi, ma non lo so.

Seconda semantica (at least once - il proxy lato client se non riceve una risposta ripete la richiesta, quello lato server NON filtra i duplicati): poniamo che se il proxy lato client non riceve risposta ripete la richiesta per al più un'altra volta (due in totale al massimo quindi). La prima possibilità è che non si perda né la richiesta né la risposta, e quindi il client riceve immediatamente il risultato. La seconda possibilità è che si perda la prima richiesta, allora il client non riceve il risultato e il proxy lato client invia una seconda richiesta; se si perde anche questa l'operazione non viene mai eseguita e il client riceve un'eccezione remota. In questa seconda possibilità però potrebbe anche accadere che la seconda volta il messaggio di richiesta non venga perso e che l'operazione venga eseguita, allora il proxy lato client riceve la risposta, il client riceve il risultato, e in questo caso l'operazione è stata eseguita una volta. La terza possibilità invece è che si perdano le risposte: il proxy lato client invia la richiesta, l'operazione viene eseguita, viene persa la risposta che non torna al proxy lato client, che quindi ripete la richiesta (nota che non è il componente client a ripetere la richiesta, nel suo scope lui ha fatto solo un'invocazione locale) e l'operazione viene eseguita una seconda volta; la risposta si perde di nuovo, il client riceve un'eccezione remota ma l'operazione a questo punto è stata eseguita due volte.
Quindi andando a vedere tutte le combinazioni di questa semantica, attraverso le tre possibilità, se il client riceve un risultato vuol dire che ALMENO UN messaggio di richiesta e ALMENO UN messaggio di risposta sono stati recapitati correttamente però i messaggi di richiesta che sono arrivati al server potrebbero essere tanti, più di uno. Se ricevo un risultato quindi l'operazione è stata eseguita almeno una volta, at least once. Inoltre se il client riceve l'eccezione remota l'operazione potrebbe non essere mai stata eseguita o anche eseguita tutte le volte che è stata richiesta (le n volte che il proxy reinvia la richiesta). At least once quindi si riferisce al caso in cui il client riceve un risultato, se invece il client riceve un'eccezione remota non posso determinare quante volte è stata eseguita l'operazione.

Terza semantica (at most once - il proxy lato client se non riceve una risposta ripete la richiesta, quello lato server filtra i duplicati): alle richieste viene associato un identificatore e il proxy lato server filtra i duplicati. Se invio tante richieste e si perdono tutte non ricevo nessuna risposta, quindi nessun risultato e il client riceve un'eccezione remota. Se invece si perdono tutte le risposte la prima volta quello che succede è che la prima volta che arriva la richiesta l'operazione viene eseguita e il risultato messo nella cache ma quando ricevo la richiesta successiva l'operazione NON viene riseguita, quindi se alla fine il client riceve un risultato sa che l'operazione è stata eseguita ESATTAMENTE una volta, se invece riceve un'eccezione remota, dal momento che non sa capire se si sono perse le richieste o le risposte, sa che l'operazione forse è stata eseguita ma se è stata eseguita è stata eseguita una sola volta (quindi 0 o 1 volta). Quindi "at most once" si riferisce al caso in cui viene ricevuta dal client un'eccezione remota.

Nota che "maybe" si riferisce al caso in cui il client riceve un'eccezione remota, "at least once" si riferisce al caso in cui il client riceve un risultato, "at most once" si riferisce al caso in cui il client riceve un'eccezione remota.

Si consideri anche il fatto che se si tratta solo di PERDITA di messaggi, allora l'unico che può sollevare eccezioni remote è il proxy lato client. Se invece c'è CORRUZIONE di messaggi, anche il proxy lato server può sollevare eccezioni remote (ad esempio il proxy lato client fa una richiesta per l'invocazione di un'operazione Pippo() che NON è presente sul server e quindi il proxy lato server non riesce ad estrarre il messaggio di invocazione dalla richiesta - eccezione remota).

Semantica dell'invocazione remota

- Possibili semantiche per l'invocazione remota
  - che cosa può capire il client se riceve un risultato?
  - e se riceve un'eccezione remota?
  - *maybe* – il proxy lato client non ripete richieste
  - *at least once* – il proxy lato client ripete richieste, il proxy lato server non filtra duplicati e semmai chiede al server di rieseguire l'operazione
  - *at most once* – il proxy lato client ripete richieste, il proxy lato server filtra duplicati e ritrasmette le risposte (senza chiedere al server di rieseguire l'operazione)

A seconda dei casi potrebbe capire che l'operazione è stata eseguita esattamente una volta (at mosto once) o almeno una volta (at least once)

L'operazione potrebbe essere stata eseguita oppure no, ma al massimo una volta (maybe, at most once), l'operazione potrebbe essere stata eseguita 0,1 o anche più volte (at least once). Qui maybe e at most once coincidono ma at most once è più tollerante ai guasti

Queste semantiche sono diverse dalla semantica dell'invocazione locale:

- La semantica dell'invocazione "locale" è invece *exactly once*
  - l'operazione viene eseguita esattamente una volta
  - un'invocazione termina sempre con la restituzione di un risultato

Discussione

Come faccio a capire qual è la semantica dell'invocazione remota?
Devo consultare la semantica di tale invocazione remota

Semantica	Opzioni	Se il client riceve un risultato, l'operazione è stata eseguita	Se il client riceve un'eccezione remota, l'operazione è stata eseguita	Note
Maybe	il proxy lato client non ripete richieste	esattamente 1 volta	0 o 1 volta	
At least once	il proxy lato client ripete richieste, il proxy lato server non filtra duplicati e chiede al server di riesegue l'operazione	1 o più volte	0 o più volte	
At most once	il proxy lato client ripete richieste, il proxy lato server filtra duplicati e ritrasmette le risposte (senza chiedere al server di rieseguire l'operazione)	esattamente 1 volta	0 o 1 volta	maggiore tolleranza ai guasti rispetto a maybe
Exactly once	invocazione locale	esattamente 1 volta	non è possibile	invocazione locale

Discussione

- Dunque, tre possibili semantiche per l'invocazione remota
  - nessuna corrisponde alla semantica *exactly once* dell'invocazione "locale"
  - differiscono in quello che può accadere durante un'invocazione remota
    - l'operazione remota può essere eseguita dal server zero, una o anche più volte
  - differiscono in quello che il client può comprendere alla fine dell'invocazione remota
    - quando al client viene restituito il controllo, insieme a un risultato oppure a un'eccezione remota

Discussione

- Quando si usa uno specifico servizi di middleware, è importante capire qual è la semantica dell'invocazione remota che implementa, nonché le sue implicazioni – ad esempio
  - se la semantica è *at-least once*, allora l'invocazione di un'operazione può causare delle esecuzioni multiple
    - in alcuni casi (quali?) è opportuno implementare l'operazione in modo idempotente
  - se invece la semantica è di tipo *at-most once* (o *maybe*)
    - l'idempotenza delle operazioni remote non è in genere necessaria (oppure no?)

E questo può causare danni collaterali, nel caso di operazioni molto costose da eseguire o di operazioni che modificano dati sensibili (esempio contare i voti, spostare soldi ecc)

Dipende da come si comporta l'utente lato client (lo stesso client potrebbe ripetere la richiesta). In alcuni casi quindi potrebbe essere comunque opportuno implementare operazioni in modo idempotente, ad esempio le operazioni di accettazioni di ordini o di pagamenti

Discussione

- La tattica della ripetizione dei messaggi di richiesta in assenza di una risposta dal server – usata nelle semantiche *at-least once* e *at-most once* – è sempre efficace?

  È efficace se il guasto è transiente. Se invece è duraturo, questo può peggiorare le cose.

  - questa tattica consente di gestire l'eventuale perdita dei messaggi di richiesta e di risposta – ma è efficace solo se i problemi di comunicazione sono transienti
  - può consentire di gestire eventuali problemi di indisponibilità del processo server – ma solo se questi problemi sono transienti
  - tuttavia, se questi problemi sono duraturi, questa tattica può addirittura peggiorare la situazione, anziché migliorarla
    - ad es., se il processo server è lento a rispondere per un problema di carico (ci sono molti thread attivi), la ripetizione delle richieste, soprattutto con la semantica *at-least once*, può peggiorare le cose (i thread attivi possono aumentare)
    - potrebbero essere preferibili altre tattiche, per sostenere l'isolamento dei guasti

- Prestazioni

- Invocazioni remote e invocazioni "locali" differiscono anche rispetto alle prestazioni
  - il protocollo richiesta-risposta su cui si basa l'invocazione remota introduce degli overhead (quali?)
  - per minimizzare la penalizzazione sulle prestazioni, è spesso utile minimizzare il numero di invocazioni remote (come?)

C'è un overhead, per lo scambio di messaggi in rete ecc, quindi devo essere cosciente che un'invocazione remota è diversa da una locale. Inoltre devo essere consapevole che un'invocazione remota ha una grana diversa da quella locale. Se faccio 10 operazioni in invocazione remota per settare i dati del cliente, questo non è efficiente. Voglio fare UNA invocazione remota di un'operazione a GRANA GROSSA (setta tutti i dati nella stessa chiamata).

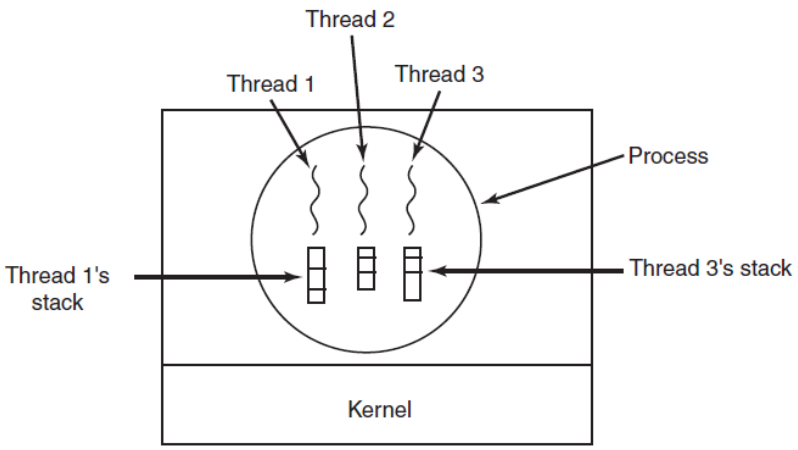
- Concorrenza

- Le operazioni remote possono essere eseguite in modo concorrente tra loro
  - un'operazione remota viene di solito eseguita in un pop-up thread distinto del componente remoto
  - l'esecuzione di un'operazione remota viene gestita in uno stack dedicato alla chiamata remota
  - è possibile che più invocazioni remote vengano eseguite in modo concorrente
  - è possibile che più operazioni remote eseguite in modo concorrente operino su risorse condivise

Parentesi: Processi e thread

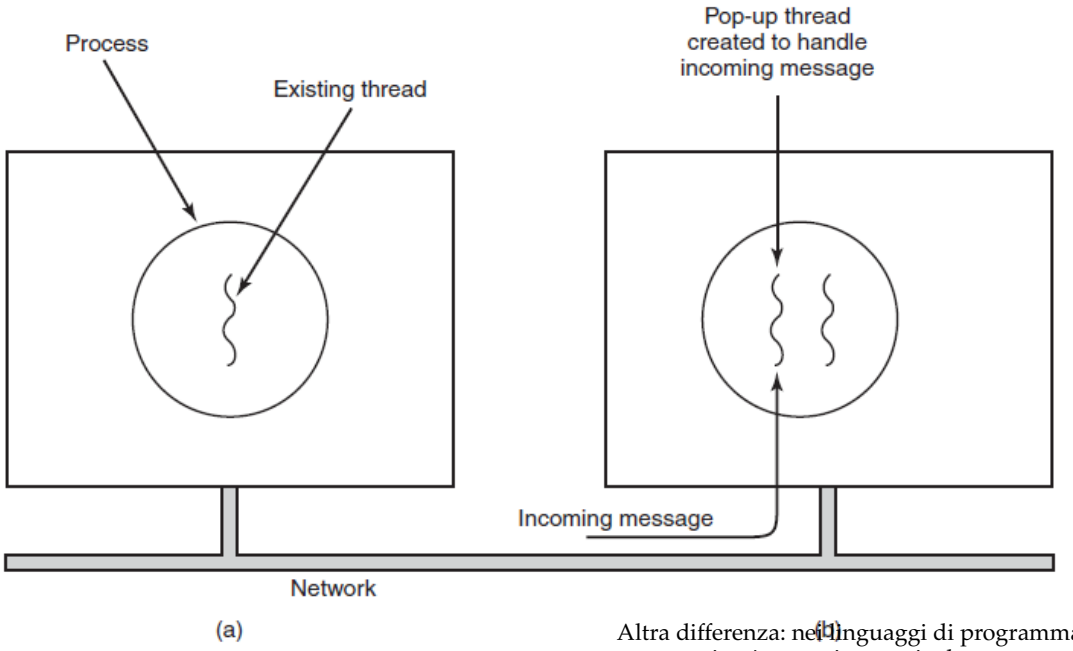
I processi possono essere multithread, un processo ha un unico spazio degli indirizzi quindi un unico heap mentre invece ogni thread ha un proprio stack, quindi quando viene creato un nuovo thread condivide dati con gli altri thread nell'heap ma non condiyide lo stack. Quando il thread termina questo stack viene liberato

- Nei sistemi operativi
  - un *processo* è un'istanza di un programma in esecuzione
  - un *thread* è un'entità che può essere assegnata a un processore ed eseguita dal processore
  - ogni processo ha almeno un thread di esecuzione
  - i thread di un processo sono eseguiti in modo concorrente e condividono con il processo le sue risorse



Parentesi: Pop-up thread

- In un processo server, si parla di *pop-up thread* se, ogni volta che il server riceve una richiesta, il processo server crea un nuovo thread (pop-up thread) per gestire quella richiesta



- Legame dei parametri

Altra differenza: nei linguaggi di programmazione il legame dei parametri può avvenire tramite legame per valore (si ha un'espressione, valuto il valore e lo assegno alla variabile corrispondente) o legame per riferimento (il parametro è un riferimento e l'invocato puo invocare operazioni tramite questo riferimento o cambiare dati in una struttura nello heap e quindi provocare effetti collaterali). Nell'invocazione remota invece il legame dei parametri è per valore-risultato (ogni operazione ha dei parametri di input e dei parametri di output che possono essere più di uno, i parametri di input vengono legati al momento dell'invocazione, i parametri di output vengono legati al contrario quando viene ricevuto il risultato).

- Nell'invocazione remota si adotta di solito il legame dei parametri per *valore-risultato*
  - in un'operazione, si fa distinzione tra parametri di input (in) e risultati/parametri di output (out)
  - i parametri di input e output vengono legati in momenti distinti
- Esempio: calcola le radici di un'equazione di secondo grado
  - intestazione dell'operazione (omettendo i tipi)
    - calcolaRadici(in a, in b, in c, out x1, out x2)
  - invocazione dell'operazione
    - calcolaRadici(1, 4, 2, radice1, radice2)
- In molti linguaggi, è disponibile solo in una versione limitata
  - double sqrt(double x)
  - r = sqrt(125)

Legame di dati strutturati e oggetti

- Una differenza importante tra la semantica dell'invocazione remota e dell'invocazione "locale" è nel modo in cui avviene il legame di dati strutturati – come oggetti o record
  - in un'invocazione "locale", viene di solito passato un *riferimento* all'oggetto
    - questo rende possibile il verificarsi di effetti collaterali sugli oggetti passati come parametri
  - in un'invocazione remota, viene invece passata una *rappresentazione serializzata* dello stato dell'oggetto
    - questo disabilita la possibilità di avere effetti collaterali sugli oggetti passati come parametri (o restituiti come risultati)

Per esempio in Java l'unico legame dei parametri è per valore però se una variabile è di tipo riferimento al valore, allora l'invocato può invocare operazioni sul riferimento e provocare effetti collaterali su questo oggetto passandolo come parametro

Quando il tipo di un parametro è un valore viene passato il valore (nel messaggio di richiesta o di risposta viene passata la codifica di questo valore) mentre quando il parametro è un oggetto o un record questo oggetto o questo record viene SERIALIZZATO e trasmesso in rete. Da un'altra parte verrà deserializzato

Serializzazione

- *Serialization* Consiste nella codificazione di una struttura di dati
  - In computer science, serialization is the process of translating a data structure or an object state into a format that can be stored (for example, in a file) or transmitted (for example, across a network connection link) and reconstructed later (possibly in the same or in a different computer environment).
    When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.
    [Wikipedia, 2020]

Legame di dati strutturati e oggetti

- Un parametro che è un oggetto viene passato come una rappresentazione serializzata dell'oggetto – che viene trasmessa e poi ricostruita nel processo remoto
  - **alpha(value-object)**

Quello che succede quando la serializzazione è fatta in rete è che, per esempio, io ho un albero nel processo client, ne trasmetto la rappresentazione serializzato in rete, ricostruisco l'albero nel processo server ma questo è un albero diverso (è identico, è un clone ma NON È lo stesso albero). Quindi se io sul server aggiungo dei nodi, l'albero sul lato client rimane lo stesso, e quindi normalmente non ha effetti collaterali a meno che io non ritrasmetta l'albero indietro e poi lato client non cerchi di capire cosa è stato cambiato di quell'albero magari per uniformarlo alla nuova versione

call alpha(

```
{
    "type" : "asw.model.Employee",
    "id" : "410",
    "firstName" : "John",
    "lastName" : "Smith",
    "dept" :
    {
        "type" : "asw.model.Department",
        "id" : "992",
        "name" : "Engineering"
    }
}
```

)

Quando passo un oggetto, quando viene fatta la serializzazione, soprattutto quando questa viene fatta in modo automatico, nella rappresentazione serializzata come nell'esempio in json insieme all'oggetto impiegato ci sono i dati sul dipartimento. Quindi quando serializzo un oggetto in realtà vengono serializzati TANTI oggetti perché i collegamenti vengono seguiti, e in alcuni casi bisogna stare attenti per non passare dati (es se l'impiegato conosce il dipartimento ok, se il dipartimento conosce la città ok, ma se il dipartimento conosce i suoi impiegati quando io passo un impiegato viene passato il dipartimento e con esso tutte le informazioni degli impiegati, e quindi volevo passare le informazioni su un impiegato e invece trasmetto in rete le informazioni su TUTTI gli impiegati)

Legame di riferimenti remoti

- In RMI, è possibile passare anche dei parametri che sono dei "riferimenti remoti" a oggetti remoti
  - **beta(remote-object)**

call beta(*remote reference to remoteObject*)

  - in questo caso, l'operazione remota potrà effettuare a sua volta delle invocazioni remote sull'oggetto remoto passato come parametro

Altra possibilità è che io passi un riferimento remoto all'oggetto: ad alcuni oggetti possono essere assegnati riferimenti remoti; i riferimenti locali sono puntatori ma i puntatori in processi distinti non hanno senso perché i puntatori in processi distinti fanno riferimento a blocchi di indirizzi diversi. Quello che posso fare però è assegnare ad alcuni oggetti degli identificatori che sono una coppia (indirizzo ip, porta associata all'oggetto del processo). In questo caso nelle tecnologie a oggetti distribuiti io posso fare invocazioni remote su oggetti passati mediante riferimento remoto. Attenzione che sono invocazione remote, devo tenere a mente che non sono invocazioni remote altrimenti rischio una forte penalizzazione delle prestazioni.

- Discussione

- Le invocazioni remote vengono scritte sintatticamente, nel codice, in modo simile a quelle “locali” – tuttavia, la loro semantica è differente, per vari motivi
  - per la possibilità di fallimenti nella comunicazione remota – e per il modo in cui vengono gestiti
  - per il tempo richiesto dalla gestione di una chiamata
  - per la possibile concorrenza delle chiamate remote
  - per come viene effettuato il legame dei parametri – e per gli effetti collaterali che si possono verificare (o non verificare)
  - per aspetti legati alla sicurezza e alla disponibilità (che non sono stati discussi)

Altre differenze riguardano ovviamente la sicurezza, dal momento in cui passo informazioni in rete potrei voler usare cifratura per proteggere le informazioni

* Ulteriori aspetti e varianti dell’invocazione remota

- Discutiamo ora alcuni ulteriori aspetti e varianti dell’invocazione remota
  - invocazione remota “asincrona”
  - trasparenza dalla locazione

- Invocazione remota “asincrona”

- Alcune varianti dell’invocazione remota – utili soprattutto per operazioni di lunga durata e dunque con una latenza alta
  - *invocazione (remota) asincrona*
    - la chiamata è non bloccante – e restituisce immediatamente un oggetto (*promise* oppure *future*) per il risultato
  - *invocazione (remota) sincrona differita*
    - il chiamante effettua una prima chiamata (non bloccante) per attivare l’operazione remota – e poi una seconda chiamata per accedere al risultato dell’operazione
  - *callback*
    - il chiamante, al momento dell’invocazione remota (asincrona), specifica un’operazione (*callback*) che l’oggetto remoto può invocare per comunicare il risultato

Il client fa una chiamata che non è bloccante quindi viene iniziata l’esecuzione dell’operazione sul server ma il client va avanti con l’esecuzione del suo comportamento (potrà ricevere il risultato della sua chiamata in un momento diverso)

- Trasparenza dalla locazione

- Nell’invocazione remota, il client deve conoscere la locazione in rete del server per l’operazione richiesta – ad es., indirizzo IP (oppure nome di dominio nel DNS) e porta
  - nei casi più semplici, la locazione dei servizi è definita staticamente – nel client potrebbe essere specificata in un file di configurazione
  - le cose sono spesso più complesse – soprattutto nei sistemi software moderni virtualizzati o sul cloud, perché i servizi possono essere replicati e possono cambiare locazione dinamicamente
  - può allora essere utile un’infrastruttura di comunicazione per rendere i client indipendenti dalla locazione fisica dei server – un broker oppure un servizio di service discovery

* Discussione

- L'invocazione remota è un'astrazione di programmazione distribuita fondamentale, basata sull'invocazione di operazioni
  - consente a un componente (nel ruolo di client) di chiamare/invocare un'operazione di un componente remoto (nel ruolo di server), affinché il componente remoto server esegua l'operazione richiesta dal client
  - questo stile di comunicazione è sostenuto dal pattern architetturale *Broker* [POSA]