Proactive

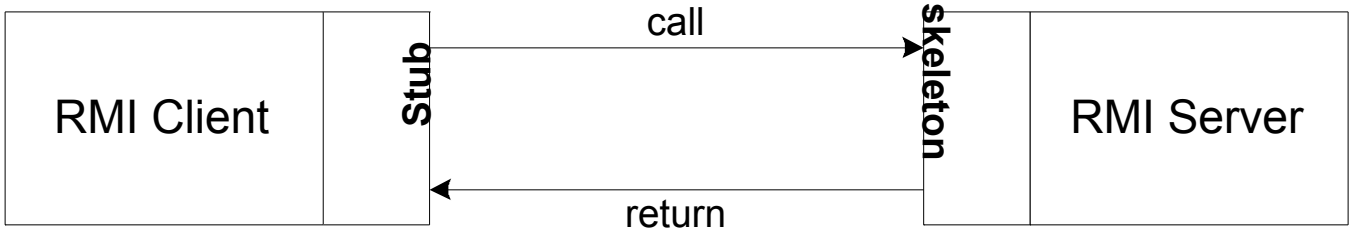
projeto ProActive

http://proactive.activeeon.com

motivação

- integracão de programacão paralela e distribuida com POO
- chamadas de métodos podem criar uma camada de abstração
  - chamada de método
  - operações coletivas
    - » barreira

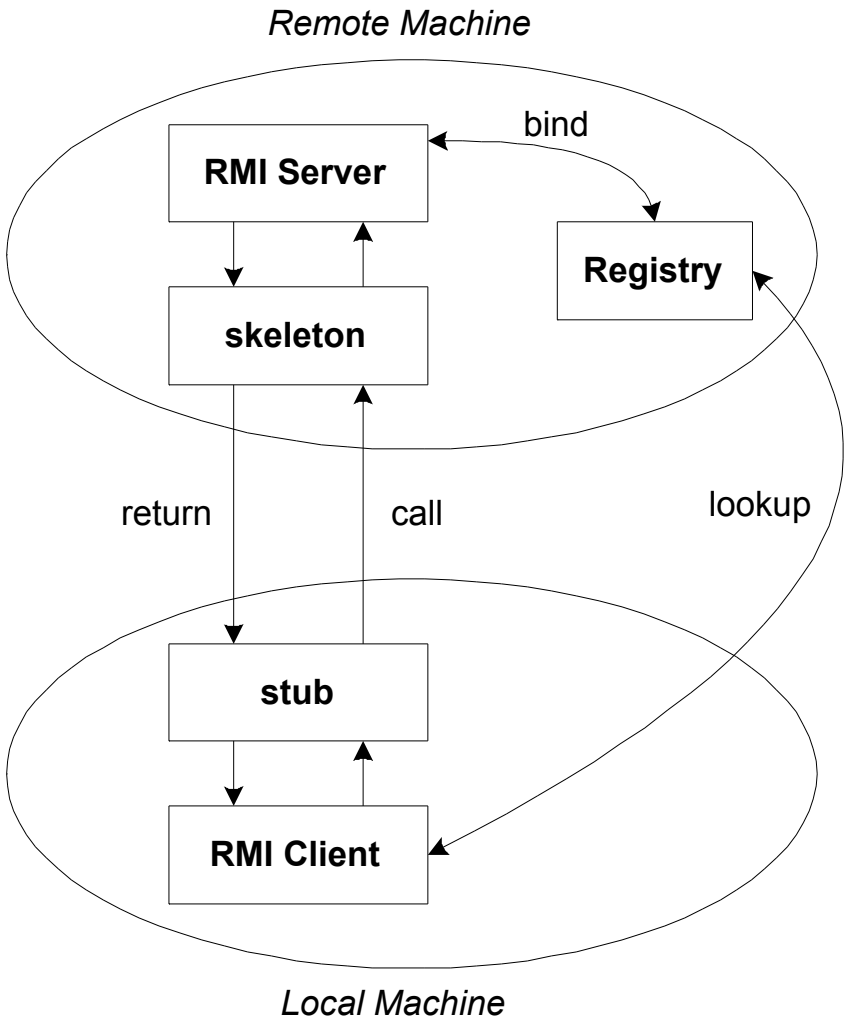
Java RMI



- stub implementa “falsa” chamada a método remoto
  - empacota argumentos, realiza chamada, e retorna resultados

arquitetura RMI

- servidor se registra em registry
- cliente contacta registry com lookup
- cliente pode baixar código de stub dinamicamente

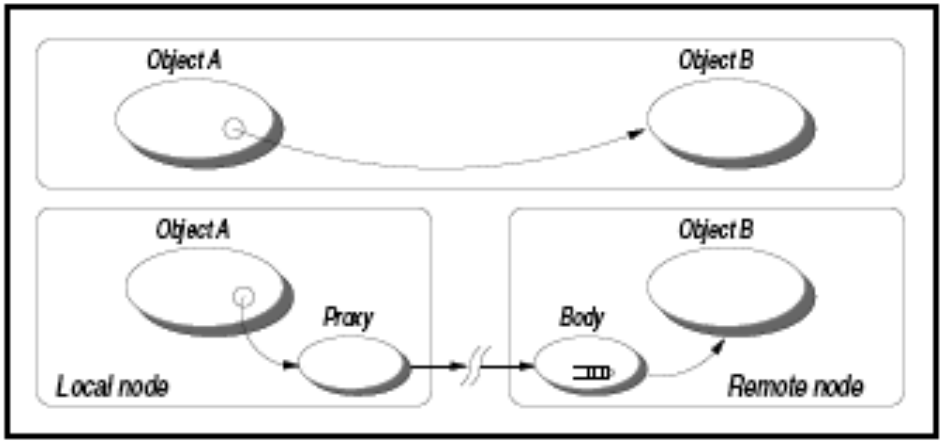


PUC RIO

ProActive

- exploração de modelo de programação assíncrono
  - transparência de localização de objetos
  - futuros e *wait on necesssity*
- comunicação coletiva
  - extensão de chamada de método
- alocação de objetos em nós remotos
- ênfase atual em grades e nuvens

Java//



- biblioteca para programação multithread e distribuída
- uso de reflexão computacional para interceptar chamadas

PUC RIO

Java// - atendimento a reqs

```
live (Body myBody)
{
  while (true)
  {
    myBody.serveOldest ();
    myBody.waitARequest ();
  }
}
```

- suporte a programação do “skeleton” com métodos da biblioteca
  - programação de políticas de atendimento a requisições

PUC RIO

sincronização intra objetos

```
class BoundedBuffer extends FixedBuffer implements Active
{
 live (Body myBody)
 {
   while (true)
   {
     if      (this.isFull())   myBody.serveOldest ("get");
     else if (this.isEmpty())  myBody.serveOldest ("put");
     else myBody.serveOldest();

     myBody.waitARequest ();
    }
  }
 }
```

sinc. inter objetos

- objetos ativos modelam concorrência e distribuição

```
class pA extends A implements Active {}
Object[] params = {"foo", new Integer (7)};
A a = (A) Javall.newActive ("pA", params, myNode);
```

- chamadas a objetos ativos são sempre assíncronas
- chamada retorna imediatamente um valor *futuro*
  - instância de subclasse do retorno declarado
  - tem todos os métodos do retorno esperado
- quando algum método é chamado sobre esse valor retornado é que ocorre a sincronização
  - wait-by-necessity

futuros

```
m1 = m0.getBlock (0, 0, m, n-1);
m2 = m0.getBlock (m+1, 0, n-1, n-1);

m1=(Matrix) Javall.turnActive(m1, remoteNode);
m2=(Matrix) Javall.turnActive(m2, localNode);

// Computes both right products
v1 = m1.rightProduct (v0);
v2 = m2.rightProduct (v0);

// Creates result vector
v3 = v1.concat (v2);
```

PUC RIO

futuros como obj de 1a classe

- objetos retornados por operações assíncronas podem ser passados como argumentos em novas operações
- otimização da transferência de dados

```
v1 = a.foo (...); // chamada assíncrona
v2 = a.bar(...); // chamada assíncrona
...
v1.f(v2); // espera apenas v1 (lazy)
```

PUC RIO

programação SPMD

- suporte a operações coletivas
  - grupos tipados: coleções de objetos de determinada classe (ou subclasses)

```
A ag = (A) ProActiveGroup.newGroup ("A",
                params, {node1, node2, node3});
ag.foo() // operação coletiva
...
V vg = ag.bar()
   // vg é um grupo de classe "V"
vg.f(); // outra operação coletiva
// chamadas sobre f's disparadas a medida
// em que resultados se tornam disponíveis
```

PUC RIO

interface Group

```
public interface Group extends Collection {
...
void add (Object o)
      //Add an element into the group.

void addMerge (Object ogroup)
      //Merge a group into the group.

Object getByType ()
      //Return an object representing the group under the typed form.

Class getType ()
      // Return the (upper) class of member.

int indexOf ()
      //Return the index in the group of the first occurence of the
      //specified element. (-1 if the list does not contain this element).

iterator iterator ()
      //Return an Iterator on the members in the group

void remove (int index)
      //Remove the object at the specified index.

int size ()
      //Return the number of members
... }
```

OO-SPMD

```
   // A group of type "A" and its members
   // are created at once by an external
   // active object
Object[][] params = {{...}, {...}};
A ag = (A) ProSPMD.newSPMDGroup("A",
                        params, {Node1,...});
   // The computation on each member may
   // now be started, i.e. invoking a method
   // called e.g compute() defined in class A
ag.compute();
```

OO-SPMD

- modelo de execução assíncrono
  - depois de inicialização, cada processo entra em loop de eventos

```
  // A reference to the typed group I belong to
A a = (A) ProSPMD.getSPMDGroup();
  // An asynchronous reference to myself
A me = (A) ProActive.getStubOnThis();
  // My rank in the group
int rank = ProSPMD.getMyRank();
 // Start the 'iterative' loop by sending
 // myself an asynchronous method call
me.loop();
 // To iterate, loop() again calls me.loop()
```

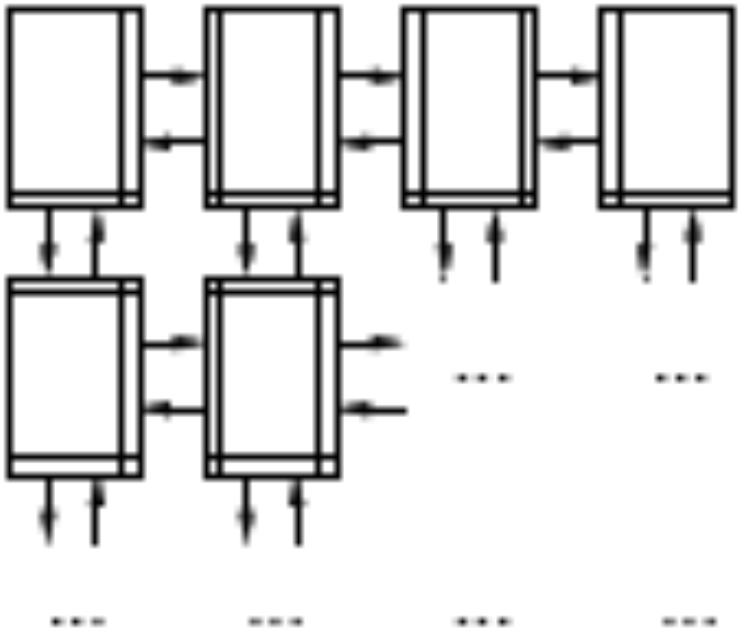
exemplo: Jacobi

```
while (!converged) {
  for (y=1 ; y<MATRIX_HEIGHT-1 ; y++) {
    for (x=1 ; x<MATRIX_WIDTH-1 ; x++) {
      new(x,y) = ( old(x,y-1) + old(x,y+1) +
                   old(x-1,y) + old(x+1,y) )/4;
      if (abs(new(x,y)-old(x,y)) < THRESHOLD) {
        converged = true;
      }
      exchange(new,old);
} } }
```

- cálculo de valor de ponto a cada interação depende de valores vizinhos na iteração anterior

PUC RIO

paralelizaçao



- ditribuição de matriz em blocos
- comunicação com dois, três ou quatro vizinhos

MPI Jacobi

```
while (!converged) {
  internal_compute(&converged);
  MPI_Send(north_border, SUBMATRIX_WIDTH,
           MPI_DOUBLE, north, 1,
           MPI_COMM_WORLD, &status);
  MPI_Recv(border_received_from_north,
           SUBMATRIX_WIDTH, MPI_DOUBLE,
           north, 1, MPI_COMM_WORLD, &status);
    // send and receive for south, east, west
  ...
  boundaries_compute(&converged);
  exchange(new,old);
} } }
```

PUC RIO

OO-SPMD Jacobi

```
me = ProActive.getStubOnThis();
public void jacobiIteration() {
  internal_compute(); //updates converged
➔ neighbors.send(boundariesGroup);
  ProSPMD.barrier({"send", ... ,"send"});
➔ me.boundaries_compute(); //updates converged
➔ me.exchange();
➔ if (!converged) me.jacobiIteration();
}
```

- barreiras forçam sincronização antes da execução de nova chamada assíncrona

➔ chamadas assíncronas

PUC RIO

Barreiras

- totais

  proSPMD.barrier ("minhabarreira");

- parciais

  proSPMD.barrier ("barreiraviz", grupo);

- locais

  proSPMD.barrier({"foo", "bar", "gee"):

PUC RIO

operações de scatter

- parâmetros que são grupos podem ser usados para compor envio por *scatter*

```
    // Broadcast the group gb to all the members
    // of the group ag1:
ag1.foo(gb);

    // Change the distribution mode of the
    // parameter group:
ProActiveGroup.setScatterGroup(gb);

    // Scatter the members of gb onto the
    // members of ag1:
ag1.foo(gb);
```

PUC RIO

modelo de programação

- modelo reativo: computação ocorre em resposta à chegada de eventos
- programa está sempre pronto para receber chamadas
  - no exemplo, entre chamadas a `me.jacobiIteration`
- adequação a distribuição geográfica e grades