Lezione n.12

JXTA: L'applicazione RestoNet

Materiale didattico distribuito a lezione

Laura Ricci

MATERIALE DIDATTICO

- Codice dell'esempio sulla pagina web del corso
- API JXTA Java 2.3.7: http://platform.jxta.org/nonav/java/api/
- Tutorial JXTA sulla pagina web del corso

JXTA: UN ESEMPIO

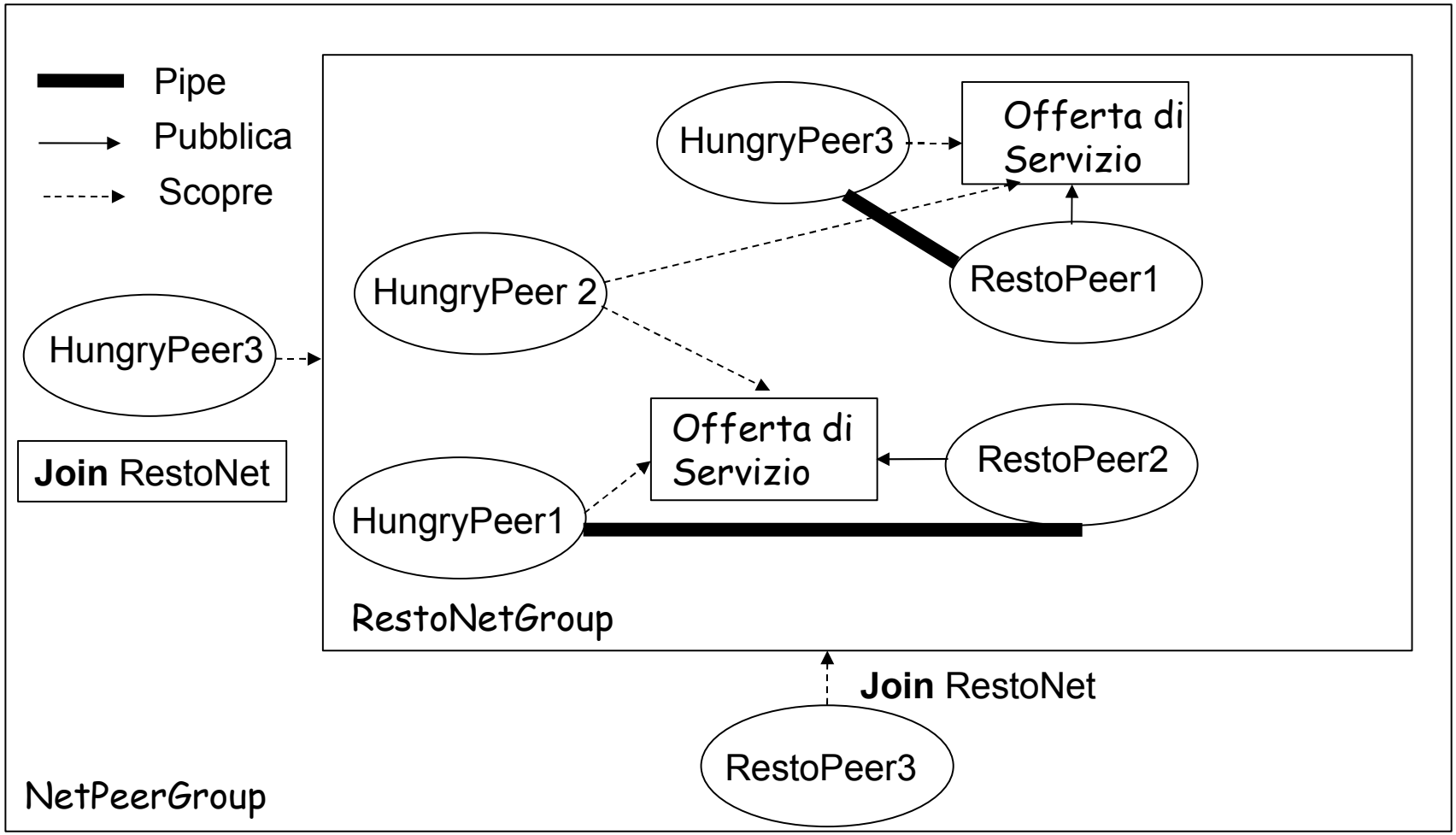
Si vuole progettare un'applicazione P2P per la ricerca distribuita di servizi di ristorazione. Il sistema prevede:

- un peergroup RestoNet a cui si uniscono i ristoratori che presentano alla comunità le proprie proposte di servizio ed i clienti che ricercano le offerte più vantaggiose per loro

- RestoPeer: forniscono servizi di ristorazione. Pubblicano le proprie offerte e rispondono alle richieste di servizio che possono soddisfare.

- HungryPeers: sono alla ricerca di un locale in cui ristorarsi. Ricercano offerte di ristorazione. Possono selezionare il ristorante in base a diversi criteri (prezzo, tipo di cibo, distanza,….)

L'APPLICAZIONE RESTONET



L'APPLICAZIONE RESTONET

Il RestoPeer

- effettua una ricerca per verificare se RestoNetGroup è già stato creato
- se la ricerca ha esito positivo si unisce al gruppo, altrimenti crea un nuovo gruppo, RestoNetGroup
- crea e pubblica una pipe RP da cui riceve le richieste di servizio da parte degli utenti
- Si mette in attesa di richieste e risponde ad esse

L'HungryPeer

- effettua una ricerca per verificare se RestoNetGroup è già stato creato
- se la ricerca ha esito positivo si unisce al gruppo, altrimenti termina
- ricerca annunci di ristorazione all'interno di RestoNetGroup (ricerca le pipes a cui connettersi per comunicare con i ristoranti)
- crea una pipe HP per ricevere le risposte dai ristoranti
- si connette ad un ristorante: collega l'input pipe ad un suo endpoint
- invia la richiesta ad uno dei ristoranti individuati ed attende la risposta su HP
- se la risposta non è soddisfacente si connette ad un altro ristorante

L'APPLICAZIONE RESTONET: L'IMPLEMENTAZIONE

```
  try
  { PeerGroup netpg = PeerGroupFactory.newNetPeerGroup ( );}
  catch (PeerGroupException e) {
  System.exit(1); }
  DiscoveryService hdisco = netpg.getDiscoveryService ( );
Enumeration ae = null;
int count = 3;
while (count-- >0)
{try
{
ae = hdisco.getLocalAdvertisements(DiscoveryService.GROUP,
                                   "Name","RestoNet");
if ((ae !=null) ) break;
hdisco.getRemoteAdvertisements(null, DiscoveryService.GROUP,
                                   "Name","RestoNet",1,null)
```

L'APPLICAZIONE RESTONET:L'IMPLEMENTAZIONE

```
        try {
                Thread.sleep(timeout)
        } catch (InterrruptedException ie) {   }
        if (ae == null)
                <creazione del nuovo gruppo>
        else   <richiesta di partecipazione al gruppo>}
catch (Exception e)
        {System.exit(1);}
 }
```

CREAZIONE DI UN NUOVO GRUPPO

```
String groupURL"jxta:uuid-4d617267657……..f202002"
PeerGroupID groupID= (PeerGroupID)
                IDFactory.fromURI(new URI("urn"," ",groupURL));

PeerGroup restoNet = null;

ModuleImplAdvertisement implAdv =
          netpg.getAllPurposePeerGroupImplAdvertisement( );

restoNet = netpg.newGroup(groupID,implAdv,"RestoNet","Gruppo Ristoranti");
```

La creazione del nuovo gruppo richiede:

- un identificatore unico per il gruppo
- la definizione dei servizi offerti dal gruppo
- il nome e la descrizione del gruppo

CREAZIONE DI UN NUOVO GRUPPO

Creazione dell'identificatore unico del gruppo:

- Quando un nuovo peer P ricerca un un advertisement per RestoNet e non riesce a reperirlo

  È possibile che esistano comunque dei peer che appartenenti a RestoNet

  in questo caso, la ricerca può aver dato esito negativo perché P non ha atteso la risposta per un intervallo di tempo sufficiente, oppure problemi di routing non hanno consentito di raggiungere quel peer

- in questo caso è possibile che P generi un advertisement per un gruppo che è già stato creato in precedenza

- P deve utilizzare lo stesso identificatore generato per il gruppo al momento della sua creazione

CREAZIONE DI UN NUOVO GRUPPO

- È necessario garantire che tutti i RestoPeer attribuiscano al gruppo lo stesso identificatore

- L'identificatore del gruppo è costruito a partire da una stringa predefinita "cablata" nel codice. Poiché tutti i RestoPeers eseguono lo stesso codice, tutti attribuiscono lo stesso identificatore al gruppo

- Un identificatore oppotuno può essere ottenuto mediante il comando mkgrp della shell JXTA

- Se l'applicazione assicura che un solo peer P possa creare il gruppo,

  P può impostare groupID a null. JXTA genera automaticamente un nuovo identificatore per il gruppo

  P può invocare la funzione newPeerGroupID() che genera un identificatore unico per il gruppo

CREAZIONE DI UN NUOVO GRUPPO

- JXTA permette di associare al nuovo gruppo un insieme di servizi, descritti mediante un insieme di module advertisements
- i servizi fanno riferimento a implementazioni (codice) disponibili a tutti i membri del gruppo
- i membri del gruppo possono scaricare l'implementazione di un servizio ed utilizzarlo (invocarne le funzioni)
- È possibile associare al nuovo gruppo solo i servizi base offerti da JXTA (peer discovery service, pipe binding service….)
- La funzione getAllPurposePeerGroupImplAdvertisement( ) crea un advertisement standard che contiene solo i riferimenti ai servizi base

RICHIESTA DI PARTECIPAZIONE AD UN GRUPPO

Se il gruppo RestoNet esisteva ed il peer è riuscito a reperire un advertisement che lo definisce

- se il gruppo implementa politiche di sicurezza, il peer che intende partecipare al gruppo deve essere autenticato

- altrimenti è sufficiente istanziare un nuovo gruppo a partire da

```
(PeerGroupAdvertisement) restoNetAdv =
            (PeerGroupAdvertisement) ae.nextElement( );
restoNet = netpg.newGroup(restoNetAdv)
```

JXTA CORE SERVICES E RISPETTIVI PROTOCOLLI

Discovery Service: pubblicazione e la ricerca (Discovery) di servizi e risorse all'inteno della rete	**Peer Discovery Protocol (PDB)**
Pipe Service: creazione e pubblicazione di pipes per lo scambio diretto di informazioni tra i peers	**Peer Binding Protocol (PBP)**
Peer Information Service: permette di monitorare lo stato degli altri peers partecipanti al gruppo	**Peer Information Protocol (PIP)**
Rendez Vouz Service: permette di contattare un rendez-vouz peer per la propagazione delle queries	**Rendez Vouz Protocol (RVP)**
End Point Service: gestisce l'invio dei messaggia basso livello	**END Routing Protocol (ERP)**
Peer Resolver Service: permette di inviare queries generiche a peers singoli a o tutti i peers di un gruppo	**Peer Resolve Protocol (PRP)**
MemberShip e Access Control: implementa meccanismi per il controllo dell'accesso di un peer ai peergroups	

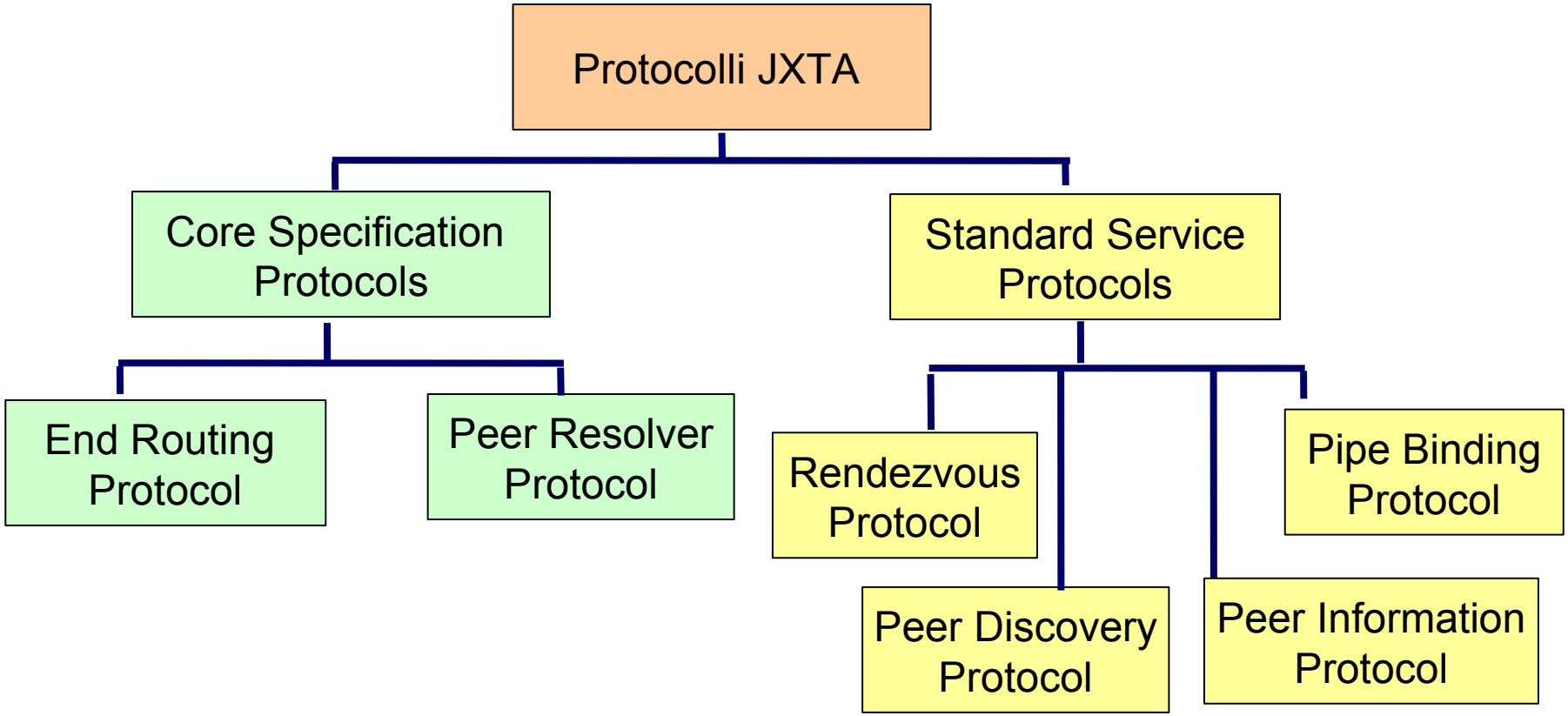
JXTA CORE SERVICES

- Alcuni servizi sono automaticamente associati al gruppo creato

- I riferimenti a questi servizi possono essere reperiti mediante invocazioni a metodi della classe PeerGroup

```
DiscoveryService     discos= restoNet.getDiscoveryService ( );
PipeService          pipes  = restoNet.getPipeService ( );
RendezVouzService    rends  = restoNet.getRendezVouzService( );
PeerInfoService      pinfos = restoNet.getPeerInfoService( );
MembershipService    membs= restoNet.getMembershipService ( );
ResolverService      resvs  = restiNet.getResolverService( );
EndPointService      endps  = restoNet.getEndPointService ( );
```

I PROTOCOLLI JXTA



JXTA: I PROTOCOLLI

Collegamento degli estremi delle Pipe agli endpoint fisici di un peer

Peer Discovery Protocol

Pipe Binding Protocol

Peer Information Protocol

Ottenere informazione sullo stato degli altri peers

Pubblicazione e Reperimento di Advertisements

Peer Resolver Protocol (PRP)

Invio di una query generica e ricezione della risposta

Definire il routing per Inviare un messaggio ad un altro peer

End Routing Protocol (ERP)

Rendezvous Protocol (RVP)

Collegamento ai Rendez vouz Peer

JXTA: IL MECCANISMO DELLE PIPES

- Le pipes consentono agli HungryPeers ed ai RestoPeers di comunicare all'interno del gruppo RestoNet

- Pipe Unidirezionali: occorre creare
  - una pipe tramite cui l'HungryPeer invia messaggi al RestoPeer
  - una pipe tramite cui il RestoPeer invia le risposte all'HungryPeer

- Ogni RestoPeer
  - crea e pubblica un pipe advertisement
  - collega l'inputpipe ad un suo endpoint

- Gli HungryPeers
  - ricercano in RestoNet le pipes create dai RestoPeers
  - utilizzano le pipes reperite per comunicare le loro richieste ai RestoPeers

PIPE ADVERTISEMENTS

- Per evitare di pubblicare una pipe con un nome già esistente, si effettua una ricerca di quella pipe, in base al suo nome

- La assenza di conflitti non può essere comunque garantita con certezza

- In caso di conflitto due peers distinti possono scegliere lo stesso nome per due pipes diverse: in questo caso un restopeer può ricevere messaggi destinati ad un altro restopeer.

- Per effettuare la ricerca del pipe advertisement

  si utilizza il DiscoveryService associato a RestoNet.

  la propagazione della query è limitata ai peers appartenenti a RestoNet

  meccanismo di scoping fornito dai gruppi

PIPE ADVERTISEMENTS

Il RestoPeer controlla se questa pipe è già stata pubblicata

```
DiscoveryService      disco= restoNet.getDiscoveryService ( );
Name = "Il Gambero Rosso";
Enumeration ae = null;
 int count = 3;
 while (count -- > 0) {
    try
     { ae = disco.getLocalAdvertisements( DiscoveryService.ADV, "name",
                    "RestoNet:RestoPipe:"+ Name);
     if ((ae != null) && ae.hasMoreElements( )) break;
     disco.getRemoteAdvertisements( null, DiscoveryService.ADV,"name"
                    "RestoNet:RestoPipe:"+ Name,1,null);
     try { Thread.sleep (time-out);} catch (InterruptedExeception e) { }
catch(IOException e) { }
}
```

CREAZIONE DEI PIPE ADVERTISEMENTS

- Se il pipe advertisement non viene individuato, ne viene creato uno nuovo e viene pubblicato nella cache locale e propagato nella rete

```
PipeAdvertisement myAdv
    = (PipeAdvertisement)
     AdvertisementFactory.newAdvertisement
                (PipeAdvertisment.getAdvertisementType( ));
myAdv.setPipeID(IDFactory.newPipeID(restoNet.getPeerGroupID( )));
myAdv.setName("REstoNet:RestoPipe:"+Name);
myAdv.setType(PipeService.UnicastType)
disco.publish(myAdv, DiscoveryService.ADV, PeerGroup.DEFAULT_LIFETIME,
                PeerGroup.DEFAULT_EXPIRATION)
disco.remotePublish(myAdv, DiscoveryService.ADV,
                PeerGroup.DEFAULT_EXPIRATION)
```

CREAZIONE DEI PIPE ADVERTISEMENTS

- **public static** Advertisement newAdvertisement(**String** advertisementType)

  costruisce un'istanza di un advertisement che possiede il tipo specificato come parametro (nel nostro caso un PipeAdvertisement)

- setPipeID,setName,setType sono metodi della classe PipeAdvertisement che attribuiscono valori ai diversi campi dell'advertisement

- IDFactory.newPipeID genera un identificatore unico per la pipe.

- l'identificatore è una combinazione dell'identificatore del gruppo e dell'identificatore unico della pipe.

CREAZIONE DELLA PIPE

- La creazione della input pipe prende in input l'advertisement della pipe myAdv
- myAdv è stato creato precedentemente oppure è stato ritrovato mediante il processo di Discovery
- Creazione della input pipe

```
PipeService pipes  = restoNet.getPipeService ( );
PipeIn = pipes.createInputPipe (myAdv)
```

- Gli hungry Peers ricercano advertisement di pipes creati dai RestoPeer
  - la chiave di ricerca è il nome della pipe
  - gli hungry peers creano l'output pipe e la collegano ad un proprio endpoint
  - il meccanismo delle pipe consente agli hungryPeers di invocare un servizio offerto dai RestoPeers

RICEZIONE DI MESSAGGI DALLA PIPE

- I messaggi inviato dall'HungryPeer al RestoPeer deve contenere

  la richiesta, ad esempio il tipo servizio richiesto (esempio: l'hungry peer richiede patatine fritte, invia il suo nome e la quantità di patatine richieste)

  l'advertisement della pipe che dovrà essere utilizzata dal RestoPeer per inviare il messaggio di risposta all'HungryPeer

- Messaggio = lista ordinata di uno o più message elements

- Ogni message element può essere a sua volta un documento strutturato

RICEZIONE DI MESSAGGI DALLA PIPE

```
Message msg = pipeIn.waitForMessage( );

PipeAdvertisement hungryPipe = (PipeAdvertisement)
      AdvertisementFactory.newAdvertisement
                (msg.getMessageElement("HungryPeerPipe").toString());
```

getMessageElement restituisce l'elemento di nome HungryPeerPipe contenuto nel messaggio

```
StructuredDocument request  =
StructuredDocumentFactory.newStructuredDocument
                (msg.getMessageElement("Request"));
```

RICEZIONE DI MESSAGGI DALLA PIPE

```
Enumeration enumeration = request.getChildren( );
while (enumeration.hasMoreElements( ))
        {
        Element el = (Element) enumeration.nextElement( );
        String attr = (String) el.getKey( );
        String value =(String) el.getValue( );
        if (attr.equals("Name"))
            { name = value}
        else if (attr.equals("Patatine"))
            { size = value}
        }
```

RICEZIONI DI MESSAGGI DA UNA INPUT PIPE

Per ricevere un messaggio da una input pipe

- Ricezione sincrona dei messaggi

  Message waitForMessage( ) **throws** InterruptedException

  blocca l'esecuzione del peer fino al momento in cui viene ricevuto un messaggio. Restituisce il messaggio ricevuto

  l'attesa bloccante può essere interrotta da un altro thread

- Ricezione asincrona dei messaggi

  si associa un PipeMsgListener alla pipe, al momento della sua creazione

  si implementa il metodo pipeMsgEvent del listener, che definisce il comportamento del peer al momento della ricezione dei messaggi

INVIO DELLA RISPOSTA

Il RestoPeer, per inviare la risposta all'HungryPeer

- Utilizza l'advertisement della pipe utilizzata da HungryPeer per ricevere le risposte per creare un' outputpipe

```
pipeOut = pipes.createOutputPipe(hungryPipe, timeout)
```

- Crea il messaggio di risposta msg (può utilizzare la classe Structure Document)

- Invia il messaggio di risposta (prezzo delle patatine, offerte speciali,…)sulla Output Pipe

```
pipeOut.send(msg)
```

HUNGRYPEER: IMPLEMENTAZIONE

- ricerca un advertisement per il gruppo RestoNet
- se non lo trova, termina, altrimenti ricerca un certo servizio presso un qualsisi ristorante del gruppo RestoNet
- l'accesso ai servizi di ristorazione avviene per mezzo del meccanismo delle pipe
- ricercare un servizio equivale a ricercare una pipe che permetta di comunicare con quel servizio.
- individuata in RestoNet l'advertisement di una pipe pubblicato da un RestoPeer, collega l'output pipe ad un proprio endpoint ed invia la richiesta
- attende la proposta di servizio e, nel caso risulti insoddisfacente, contatta un altro RestoPeer

HUNGRYPEER: IMPLEMENTAZIONE

- Ricerca del servizio di ristorazione:

```
Enumeration ae=
  disco.getLocalAdvertisements
            (DiscoveryService.ADV, "name", "RestoNet:RestoPipe:*")

disco.getRemoteAdvertisements
        (null, DiscoveryService.ADV, "name", "RestoNet:RestoPipe:*", 5, null)
```

- si utilizza una wild-card per ricercare una qualsiasi pipe di tipo RestoPipe, pubblicata all'interno del gruppo RestoNet

PDP: PEER DISCOVERY SERVICE

- Permette ad un peer di
  - specificare e pubblicare i tipi di servizio che può offrire
  - ricercare (Discovery) servizi all'interno della rete JXTA
- Esempio: pubblicazione di un servizio di piping mediante il quale scambiare i dati con gli altri peers
- Ogni servizio viene descritto mediante un advertisement
- Le query vengono automaticamente propagate sulla rete dal supporto
- L'insieme di risposte ad ogni query vengono salvate nella cache locale di un peer, per consentire una successiva ricerca più efficiente dello stesso servizio

PDP: PBBLICAZIONE DEGLI ADVERTISEMENTS

- Peer Discovery Service: gestisce due diversi tipi di query:

  local query: il servizio di discovery cerca gli advertisements all'interno della cache locale del client

  remote query: si cerca all'interno di tutta la rete JXTA (utilizza il peer resolver protocol).

PDP: PUBBLICAZIONE DEGLI ADVERTISEMENTS

- ogni advertisement viene indicizzato utilizzando il servizio SRDI (Shared Resource Distributed Indexes) usando un numero predefinito di chiavi (es: identificatore, nome, dell'advertisement)

- ogni peer P può pubblicare un advertisement A
  nella sua cache locale
  mediante invio a tutta la rete

Pubblicazione locale

- **public void** publish (Advertisement adv) **throws** IOException

- **public void** publish (Advertisement adv, **long** lifetime,
  long expiration) **throws** IOException

PDP: PUBBLICAZIONE LOCALE

Il peer P pubblica localmente l'advertisement A:

- A viene memorizzato nella cache locale
- viene creato una chiave che viene memorizzata nell'indice locale e nell'indice del rendezvouz peer
- l'indice degli altri rendez-vouz peer non viene aggiornato immediatamente. L'implementazione attuale aggiorna gli indici ogni 30 secondi.
- **long** lifetime tempo di vita, in millisecondi, dell'advertisement. Trascorso questo intervallo di tempo l'advertisement non risulta più valido.
- **long** expiration

  ogni advertisement scoperto nella rete viene memorizzato in una cache locale. L'advertisement rimane valido per un certo intervallo di tempo T, dopo cui non è considerato più valido. expiration indica, in millisecondi, il valore di T
- se i valori di lifetime e di expiration non vengono indicati, si considerano i valori definiti per default (definiti nell'interfaccia DiscoveryService, esempio 1 anno e due ore)

PDP: PUBBLICAZIONE REMOTA

Pubblicazione Remota

- **public void** remotePublish (Advertisement adv) **throws** IOException

- **public void** remotePublish (Advertisement adv, **long** expiration) **throws** IOException

- il peer P pubblica in remoto l'advertisement A:

  A non viene memorizzato nella cache locale

  A viene propagato sulla rete e memorizzato dai peers del gruppo a cui appartiene P che lo ricevono

  si può specificare un valore per l'expiration time (tempo massimo di validità dell'advertisement nelle caches degli altri peers).

  si può specificare l'identificatore di un peer su cui si vuole pubblicare A

PDP: RICERCA DI SERVIZI REMOTI

int getRemoteAdvertisements (**String** peerid, **int** type, **String** attribute, **String** value, **int** threshold)

ricerca gli advertisement pubblicati dai peer appartenenti al peergroup e li memorizza nella cache locale del peer

peerid: indica l'identificatore del peer su cui si vuole effettuare la ricerca.

- peerid=null, la query viene propagata a tutti i peer del peergroup
- propagazione: mediante multicast sulla sottorete locale, memdiante rendez vouz peer per la propagazione ai peer remoti

type tipo dell'advertisement

attribute + value: consentono di restringere la ricerca.

threshold: indica il numero massimo di advertisements che possono essere restituiti.

PDP: RICERCA DI SERVIZI REMOTI

int getRemoteAdvertisements (**String** peerid, **int** type, **String** attribute, **String** value, **int** threshold, **DiscoveryListener** listener)

DiscoveryListener: permette la ricerca asincrona dei servizi remoti.

Ricerca asincona:

- creazione di un oggetto che implementa l'interfaccia DiscoveryListener
- l'interfaccia definisce al suo interno un metodo, discoveryEvent, che
  - viene automaticamente invocato al momento della ricezione di un advertisement
  - deve essere implementato per definire le operazioni da eseguire al momento della ricezione di un advertisement
  - l'implementazione del metodo può accedere ad un oggetto DiscoveryEvent, che riferisce un DiscoveryResponseMsg che contiene i dati relativi all'advertisement individuato
- Associazione del DiscoveryListener al servizio di Discovery

PDP: RICERCA ASINCRONA DI SERVIZI REMOTI

Esempio: l'HungryPeer effettua una ricerca asincrona dei servizi (pipes) in RestoNet

```
public class HungryPeer implements DiscoveryListener
    ……
    disco.addDiscoveryListener (this);
    disco.getRemoteAdvertisements (…….)
    ……
    public void discoveryEvent(DiscoveryEvent e) {
        DiscoveryResponseMsg msg = e.getResponse();
        Enumeration e = msg.getResponses( );
        while (e.hasMoreElements( ) ){
            <memorizza l'advertisement reperito>}
```

PDP:RICEZIONE ASINCRONA DEI MESSAGGI

```
PipeMsgListener  pipeListener = new PipeListener ( )
            { public void pipeMsgEvent (PipeMsgEvent e)
                    try
                            { Message msg = e.getMessage ( ) ;
                            PipeID pipeID = e.getPipeId( );
                            System.out.println("Message"+
                            msg.getString("DataTag")+"pipe"+ (String)
                                                    pipeID;)
                            }
                    catch (Exception e)
            }

……
createInputPipe(pipeAdv, pipeListener)
……
```