A P2P Service Discovery Strategy Based on Content Catalogues

Dr. **Lican Huang, Director**
Institute of Network & Distributed Computing
Zhejiang Sci-Tech University
Hangzhou e-Brain Information Company, LTD

Outline

- Introduction
- Overview of VIRGO
- P2P Service Discovery Based on VIRGO
- Implementation
- Conclusion

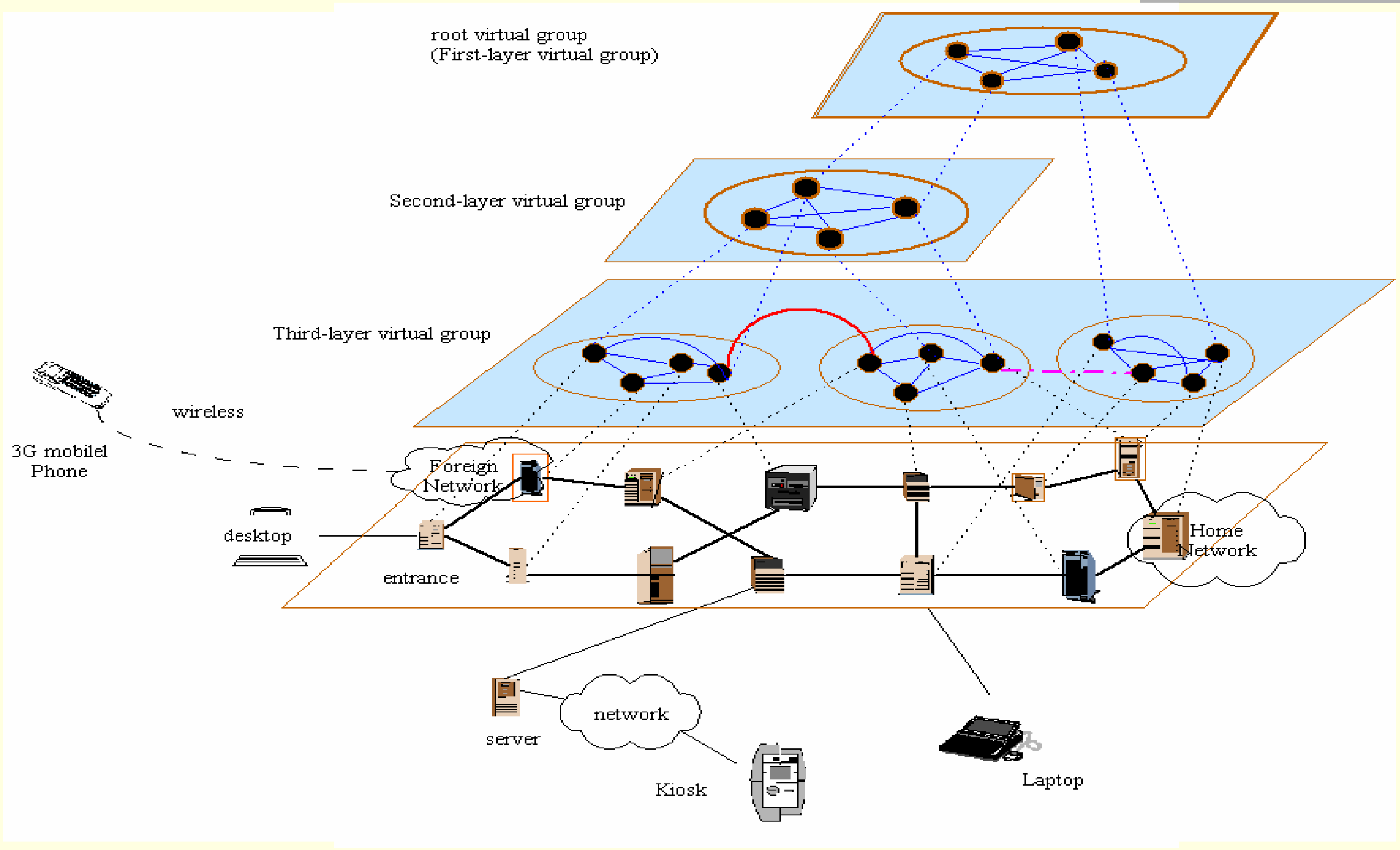
Introduction

- The current strategy for service discovery is based on centralized registers such as UDDI.
- More distributed service discoveries based on P2P technologies such as Chord lose locality.
- Distributed service discovery based on VIRGO may solve the above problems.

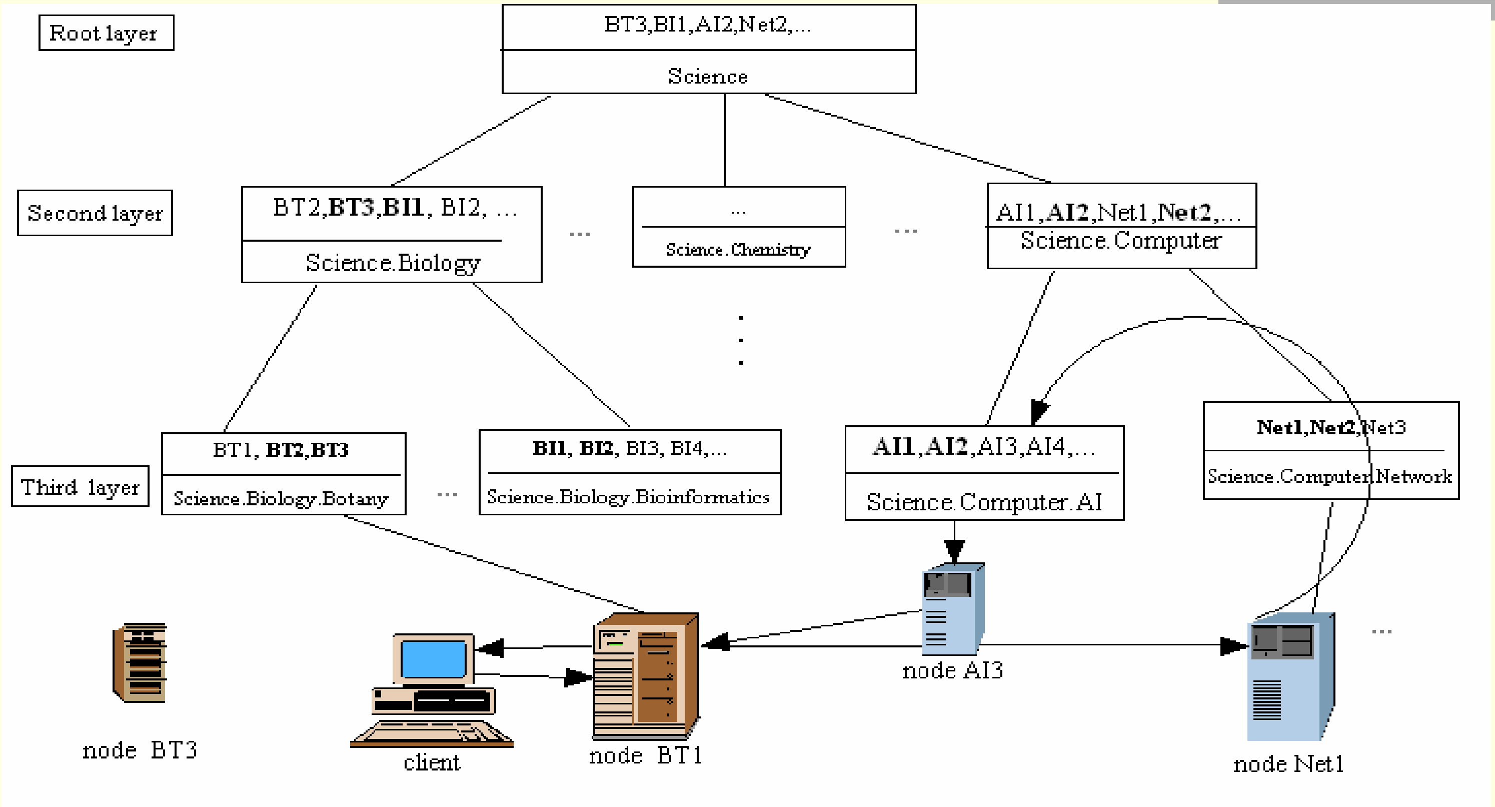
Overview of VIRGO

- **Decentralization**: VIRGO is fully distributed, robust, easy-managed.
- **Load balance**: Cached LRU and MinD nodes in route table help to solve the problem of the load balance in the tree structure.
- **Scalability**: Time complexity, space complexity and message-cost of lookup protocol of VIRGO is O(logN), where N is the total number of nodes in the network.
- **Availability**: There is at least one path between every two nodes.

VIRGO - two_tuple Virtual Hierarchical Overlay Network-1



VIRGO - two_tuple Virtual Hierarchical Overlay Network-2



NE : Science.Biology.Botany.BT3 (1)

type	route node
TREE	BT1(3),BT2(2),BI1(1),BI2(2) ,AI2(1),Net2(1)...
LRU	AI3(3),BI3(3),Net2(1)
MinD	AI1(2) ...

NE : Science.Biology.Botany.BT1 (3)

type	route node
TREE	BT2(2),BT3(1)
LRU	Net1(2)
MinD	Net3(3), ...

NE: Science.Computer.Network.Net1(2)

type	route node
TREE	Net2(1),Net3(3), AI1(2),AI2(1)
LRU	BI3(3)
MinD	BT2(2), ...

One_tuple Virtual Hierarchical Overlay Network—Music Example

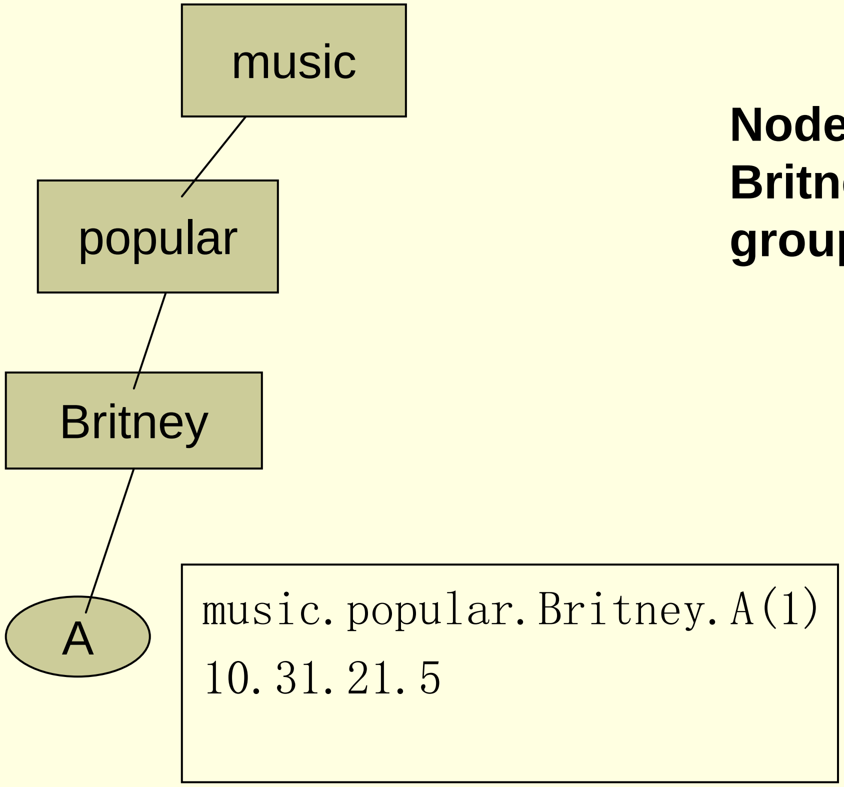
- Music Catalogue

  music.popular

  music.classic
- There are 3 nodes--Node A, B, C
- A node can join more than one group

New VIRGO Network Creation

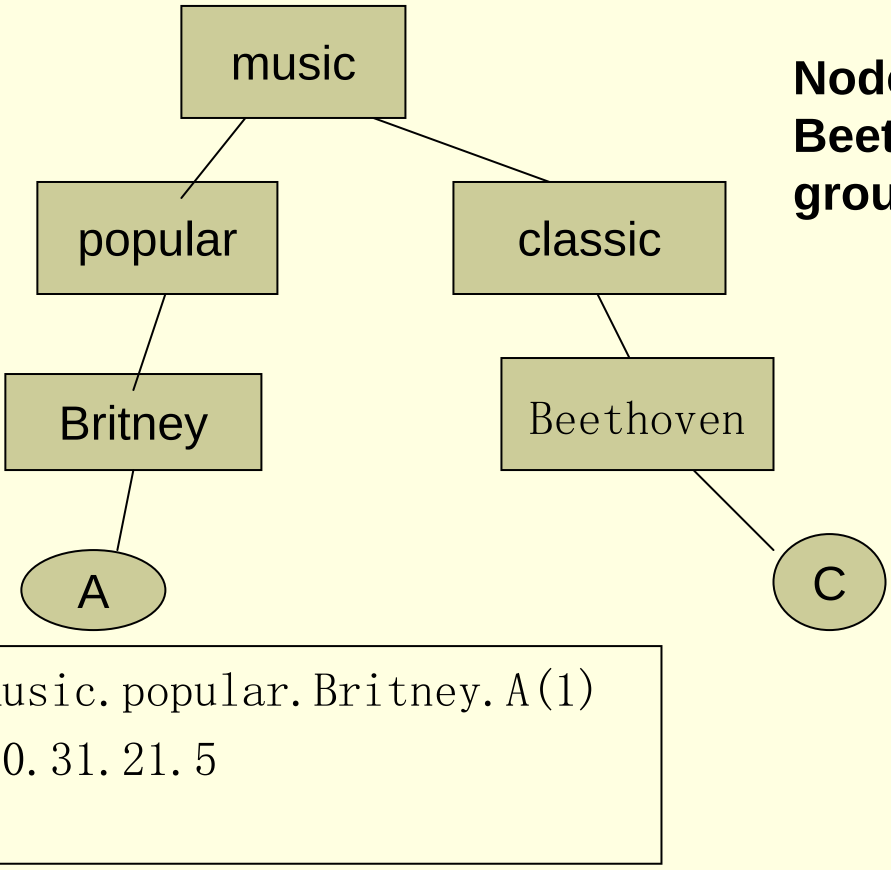
(1) Node A(IP address 10.31.21.5) **Set up new VIRGO network**



Node A is the provider of Song of Britney, so it is classified as the group of music.popular.Britney

Node Join-1

Node C (IP address 78.2.127.45) joins VIRGO network



music.popular.Britney.A(1)
10.31.21.5

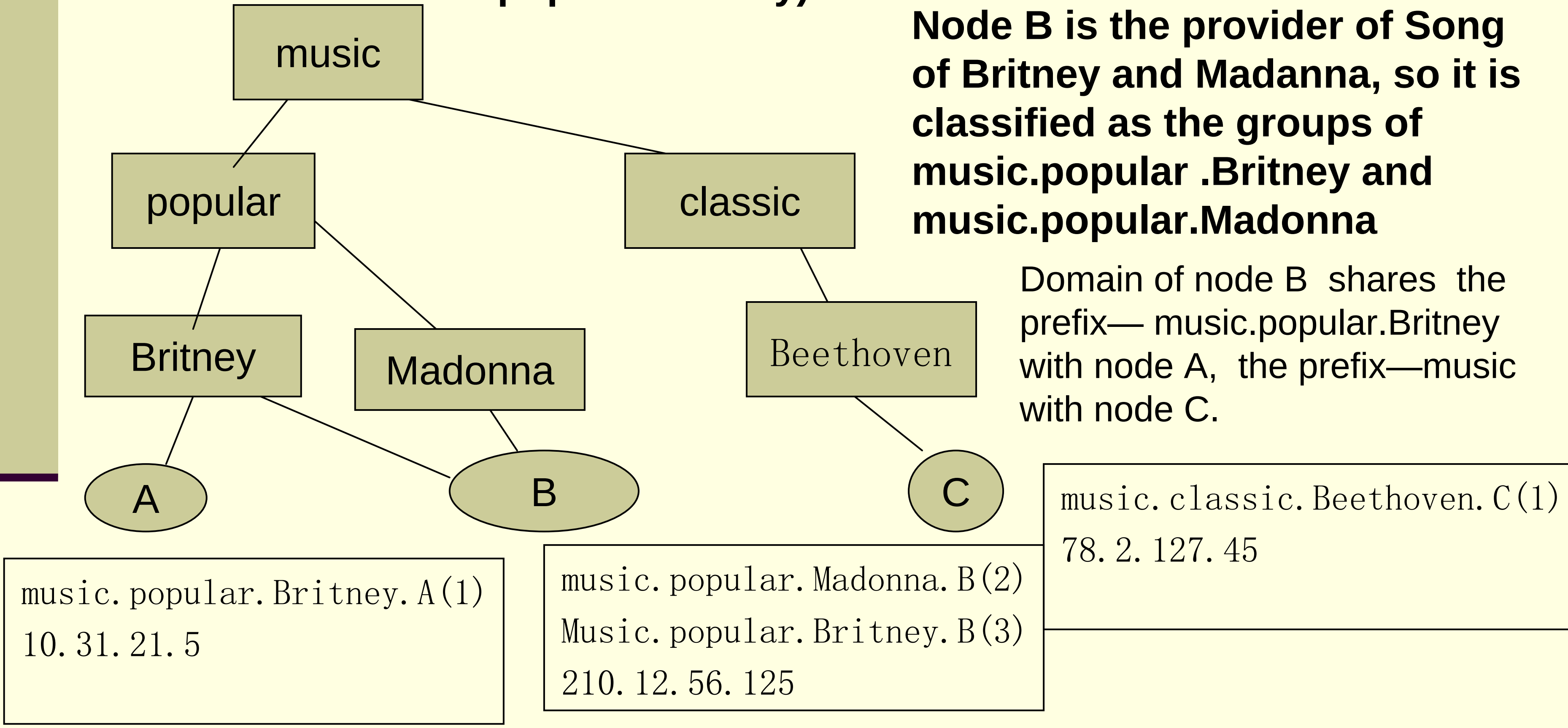
Node C is the provider of Song of Beethoven, so it is classified as the group of music.classic.Beethoven

Domains of Node A and C share the prefix--music

music.classic.Beethoven.C(1)
78.2.127.45

Node Join-2

Node B （IP address 210.12.56.125 ） **joins goup (music.popular. Madonna and music.popular.Britney)**



music.popular.Britney.A(1)
10.31.21.5

music.popular.Madonna.B(2)
Music.popular.Britney.B(3)
210.12.56.125

music.classic.Beethoven.C(1)
78.2.127.45

Node B is the provider of Song of Britney and Madanna, so it is classified as the groups of music.popular .Britney and music.popular.Madonna

Domain of node B shares the prefix— music.popular.Britney with node A, the prefix—music with node C.

Node's Join Protocol

- 1. P_{join}. send(JOINMESSAGE, $P_{groupToJoin}$)
- 2. $P_{groupToJoin}$. send(JOINMESSAGE, $\forall p_i \{p_i \in joinGroup\}$)
- 3. $\forall p_i \{p_i \in joinGroup\}$. send($p_i$. APPROVEMESSAGE, P_{join}) ;
- $\forall p_i \{p_i \in joinGroup\}$. RouteTableAdd($P_{join}$. NE, TREE)
- 4. P_{join}. RouteTableAdd($\forall p_i \{p_i \in joinGroup\}$. NE, TREE)
- 5. Repeat step 2 to 4 in upper layer groups until
- replicated nodes no less than n-tuple or root
- group.

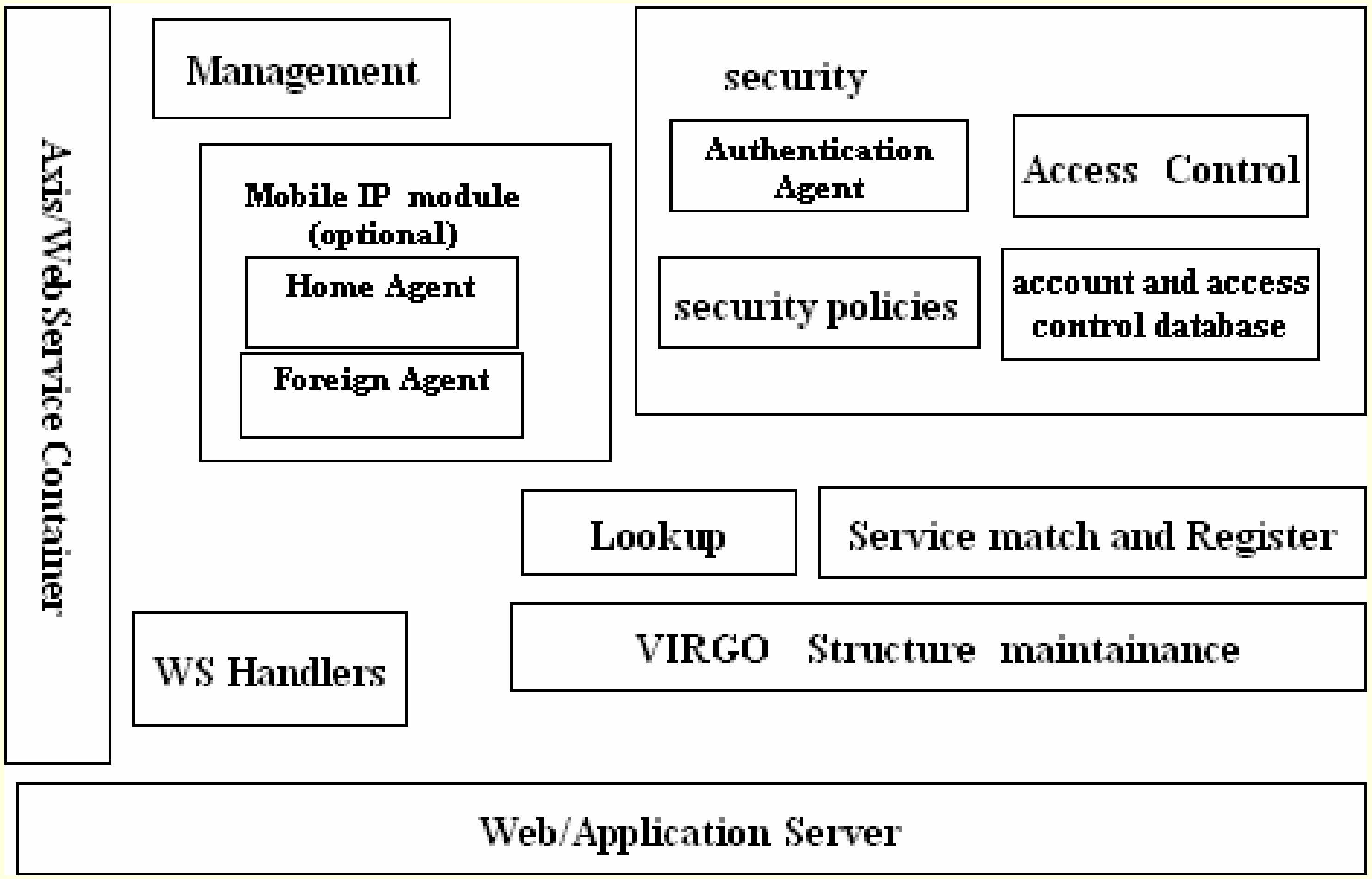
Node's Leave Protocol

- 1. P_{dpt}.send(LEFTMESSAGE, $\forall p_i \{p_i \in leftgroup\}$)
- 2. $\forall p_i \{p_i \in leftgroup\}$.RouteTableDelete($P_{dpt}$.NE, TREE)
- 3. Choose $p_{replacenode}$ to replace the left node's role
- 4. Repeat step 1 and 3 in lower layer groups until to the bottom layer group of P_{dpt}.

Node's failure Protocol

- 1. P_{notice}. send(FAILUREMESSAGE, $\forall p_i \{p_i \in failgroup\}$)
- 2. $\forall p_i \{p_i \in failgroup\}$.RouteTableDelete($P_{fail}$.NE, TREE)
- 3. Choose $p_{replacenode}$ to replace the left node's role
- 4. Repeat step 1 and 3 in lower layer groups
  until to the bottom layer group of left node P_{fail}.

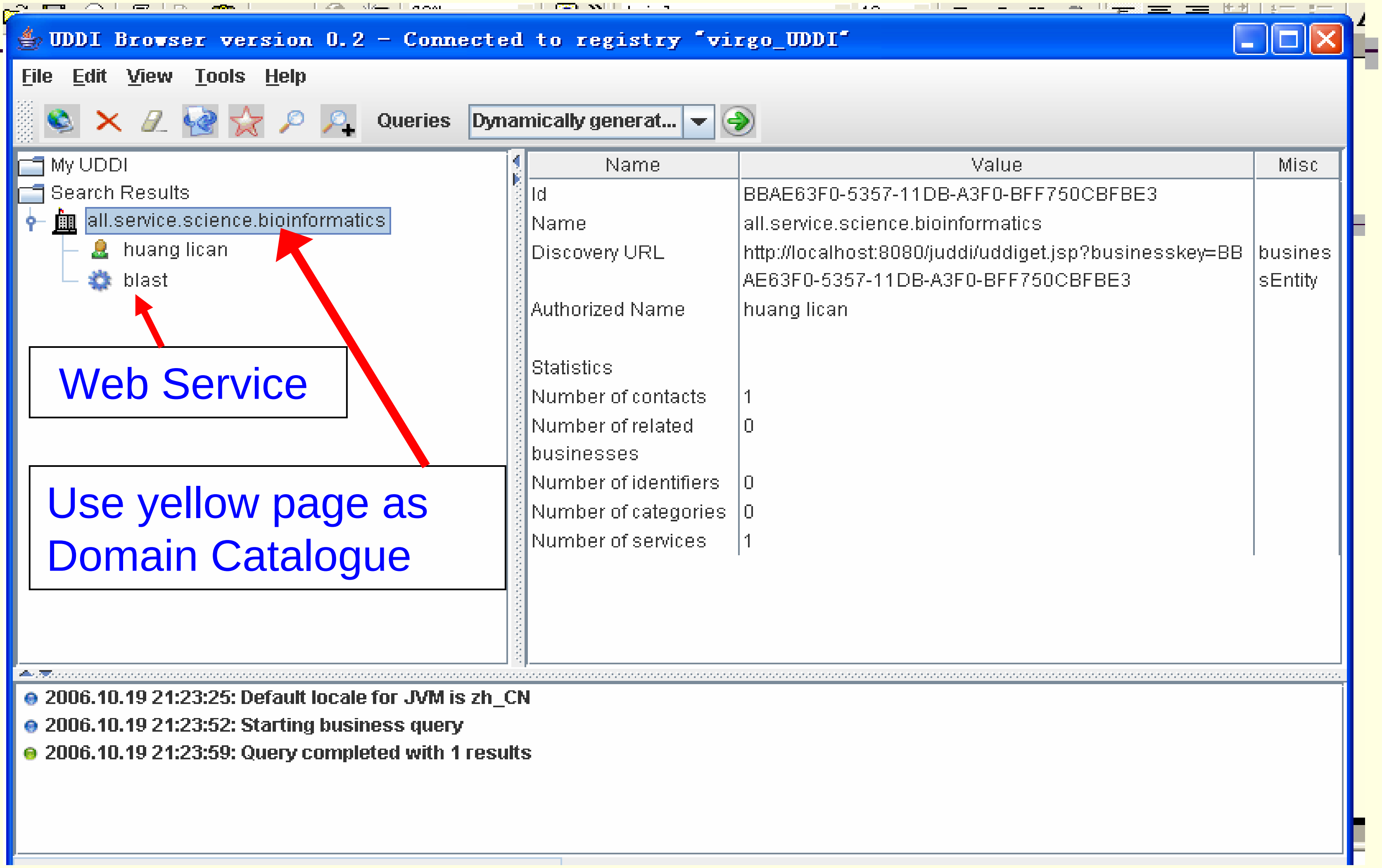
Software Architecture of VIRGO


Axis/Web Service Container
Management
Mobile IP module (optional)
Home Agent
Foreign Agent
security
Authentication Agent
Access Control
security policies
account and access control database
Lookup
Service match and Register
WS Handlers
VIRGO Structure maintainance
Web/Application Server

VIRGO based Distributed Service Discovery

- **Web Services are classified into catalogues according to their functions or disciplines such as (all.service.science.bioinformatics).**
- **Service providers registry their services into their own Registers such as UDDIs.**
- **Service providers join VIRGO network according to their domains which are the same as catalogues of web services they provided.**

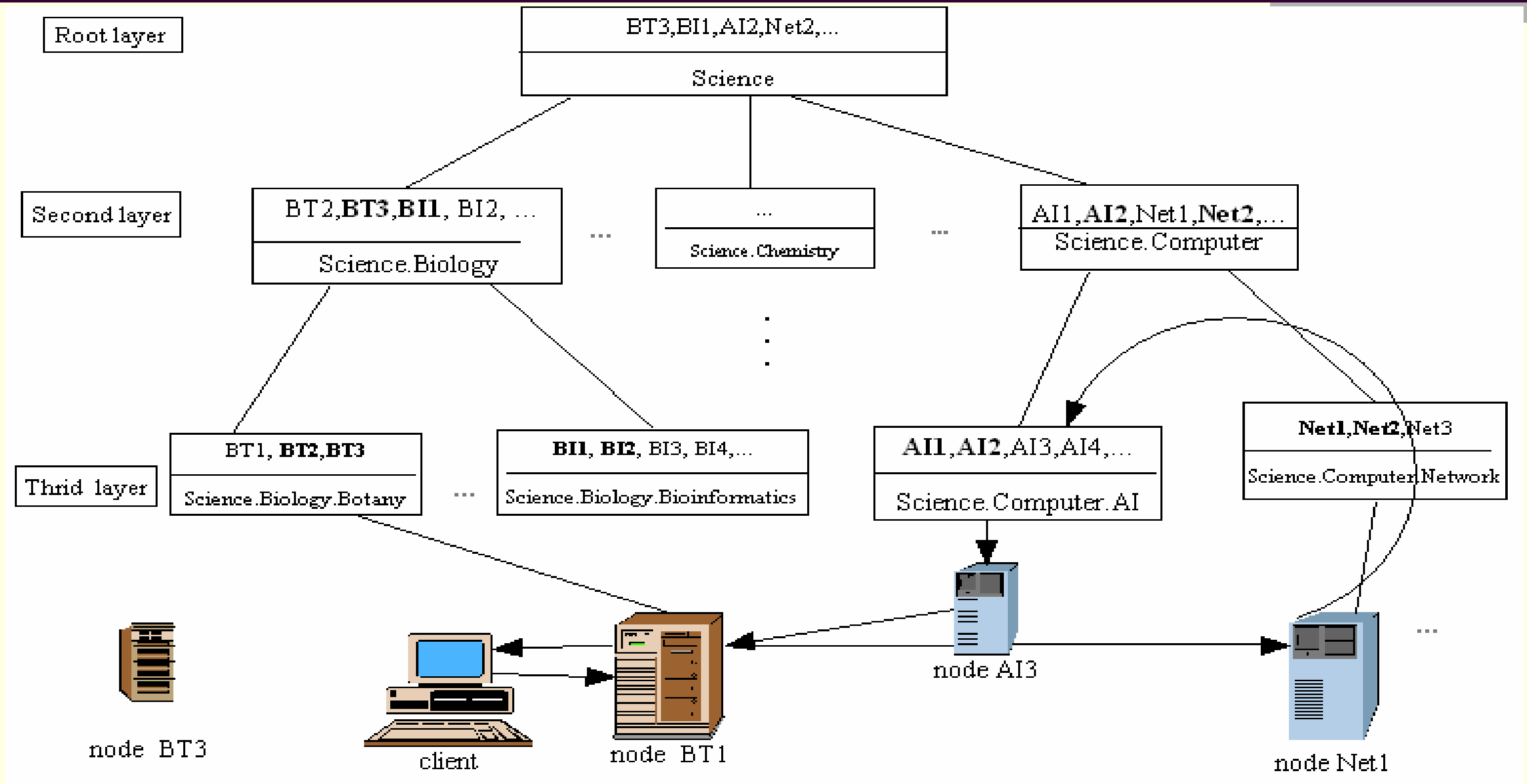
UDDI-classification


UDDI Browser version 0.2 - Connected to registry "virgo_UDDI"
File
Edit
View
Tools
Help
Queries
Dynamically generat...
My UDDI
Search Results
all.service.science.bioinformatics
huang lican
blast
Web Service
Use yellow page as Domain Catalogue

Name	Value	Misc
Id	BBAE63F0-5357-11DB-A3F0-BFF750CBFBE3	
Name	all.service.science.bioinformatics	
Discovery URL	http://localhost:8080/juddi/uddiget.jsp?businesskey=BBAE63F0-5357-11DB-A3F0-BFF750CBFBE3	businessEntity
Authorized Name	huang lican	
Statistics		
Number of contacts	1	
Number of related businesses	0	
Number of identifiers	0	
Number of categories	0	
Number of services	1	

```
2006.10.19 21:23:25: Default locale for JVM is zh_CN
2006.10.19 21:23:52: Starting business query
2006.10.19 21:23:59: Query completed with 1 results
```

Service Lookup



NE : Science.Biology.Botany.BT3 (1)

type	route node
TREE	BT1(3),BT2(2),BI1(1),BI2(2) ,AI2(1),Net2(1)...
LRU	AI3(3),BI3(3),Net2(1)
MinD	AI1(2) ...

NE : Science.Biology.Botany.BT1 (3)

type	route node
TREE	BT2(2),BT3(1)
LRU	Net1(2)
MinD	Net3(3), ...

NE: Science.Computer.Network.Net1(2)

type	route node
TREE	Net2(1),Net3(3), AI1(2),AI2(1)
LRU	BI3(3)
MinD	BT2(2), ...

Lookup Protocol

- Step 1 user uses client to send QUERY MESSAGE to entrance node.
- Step 2 entrance node forwards QUERY MESSAGE to user's owner node.
- Step 3 user's owner node checks the user's authentication.
- Step 4 user's owner node routes to the node which is closer to the
- destination group.
- Step 5 the route node routes to the closer node to the destination
- group. Repeat process step 5 until the destination group has
- been found.
- Step 6 the node found by step 5 broadcasts QUERY MESSAGE to all
- nodes belongs to the destination Group and gets the responses
- from all the nodes.
- Step 7 the node sends the RESULT MESSAGE to the user's owner node.
- Step 8 owner node sends RESULT MESSAGE to entrance node. The latter
- forwards the message to the client.

Service Lookup-QueryMessage

XML Format for QueryMessage:

```
<querymessage>
<UserID>…</UserID>
<ClientID>…</ClientID>
<entranceNode>…</entranceNode>
<ownerNode>…</ownerNode>
<ObjectDomain>…</ObejectDomain>
<serviceMeta> …</serviceMeta>
<AuthenticationTicket> …< /AuthenticationTicket>
</querymessage>
```

Service Lookup-ResultMessage

XML Format for ResultMessage:

```
<resultmessage>
<UserID>…</UserID>
<ClientID>…</ClientID>
<entranceNode>…</entranceNode>
<ownerNode>…</ownerNode>
<ObjectDomain>…</ObejectDomain>
<serviceMeta> …</serviceMeta>
<AuthenticationTicket> …< /AuthenticationTicket>
<serviceLocation>…<serviceLocation>
</resultmessage>
```

Implementation-- Enviroment

Program languages: Java, Jsp

Operation systems: linux, windows

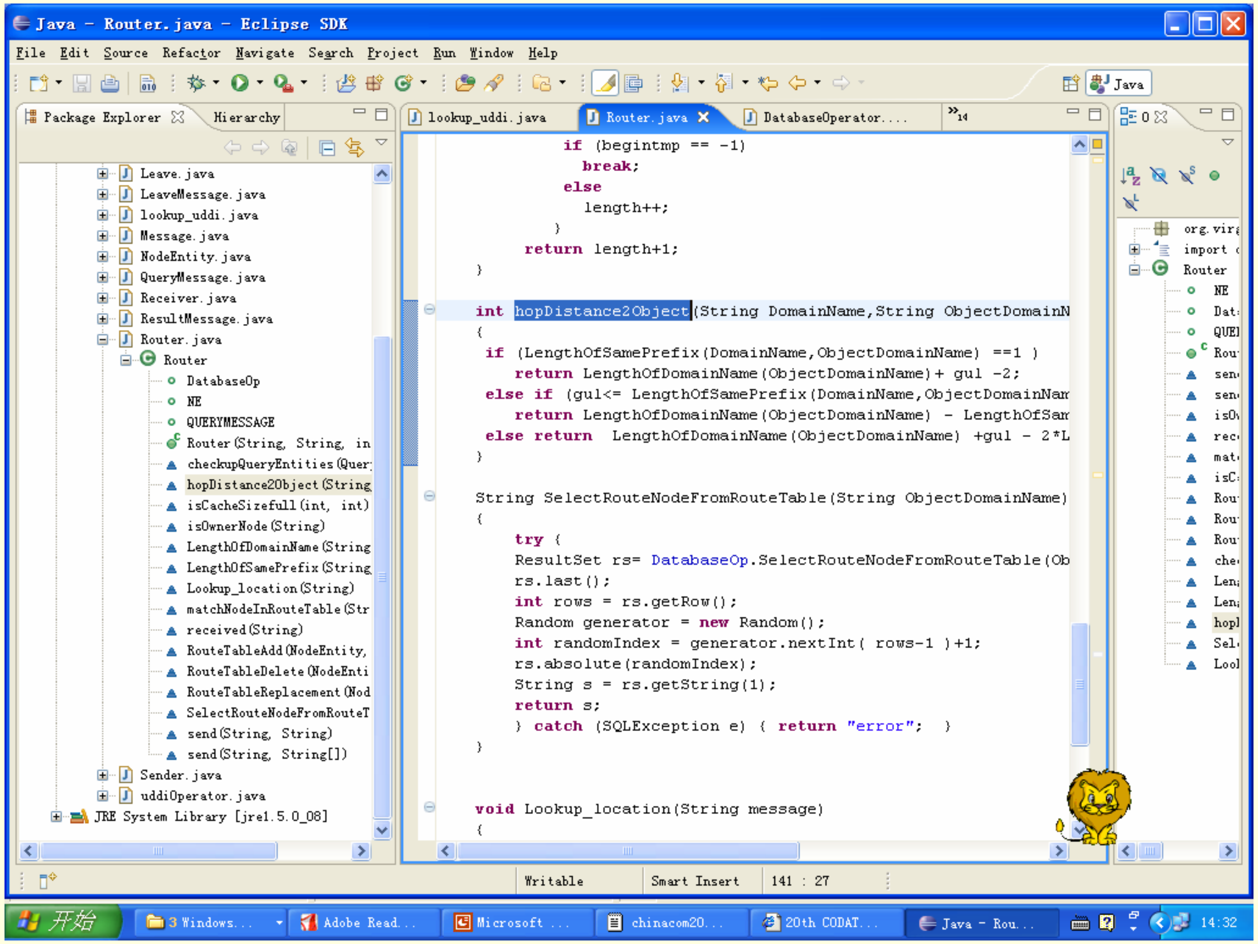
Web service container: Tomcat + AXIS

Database: mySQL

UDDI server: JUDDI

UDDI Client: UDDI Browser version0.2

Implementation-Package



```
Leave.java
LeaveMessage.java
lookup_uddi.java
Message.java
NodeEntity.java
QueryMessage.java
Receiver.java
ResultMessage.java
Router.java
  Router
    DatabaseOp
    NE
    QUERYMESSAGE
    Router(String, String, in
    checkupQueryEntities(Quer
    hopDistance2Object(String
    isCacheSizefull(int, int)
    isOwnerNode(String)
    LengthOfDomainName(String
    LengthOfSamePrefix(String
    Lookup_location(String)
    matchNodeInRouteTable(Str
    received(String)
    RouteTableAdd(NodeEntity,
    RouteTableDelete(NodeEnti
    RouteTableReplacement(Nod
    SelectRouteNodeFromRouteT
    send(String, String)
    send(String, String[])
Sender.java
uddiOperator.java
JRE System Library [jre1.5.0_08]
```

```
                if (begintmp == -1)
                  break;
                else
                  length++;
              }
          return length+1;
      }

      int hopDistance2Object(String DomainName,String ObjectDomainN
      {
       if (LengthOfSamePrefix(DomainName,ObjectDomainName) ==1 )
          return LengthOfDomainName(ObjectDomainName)+ gul -2;
       else if (gul<= LengthOfSamePrefix(DomainName,ObjectDomainNam
          return LengthOfDomainName(ObjectDomainName) - LengthOfSam
       else return  LengthOfDomainName(ObjectDomainName) +gul - 2*L
      }

      String SelectRouteNodeFromRouteTable(String ObjectDomainName)
      {
          try {
          ResultSet rs= DatabaseOp.SelectRouteNodeFromRouteTable(Ob
          rs.last();
          int rows = rs.getRow();
          Random generator = new Random();
          int randomIndex = generator.nextInt( rows-1 )+1;
          rs.absolute(randomIndex);
          String s = rs.getString(1);
          return s;
          } catch (SQLException e) { return "error";  }
      }


      void Lookup_location(String message)
      {
```

```
org.vir
import
Router
  NE
  Dat
  QUEI
  Rou
  sen
  sen
  isO
  rec
  mat
  isC
  Rou
  Rou
  Rou
  che
  Len
  Len
  hopI
  Sel
  Lool
```

Conclusion

1. **VIRGO-based distributed service discovery is fully self-organized.**
2. **VIRGO-based distributed service discovery is self-contained.**
3. **VIRGO-based distributed service discovery is effective.**
4. **All service providers have their own service registers(such as UDDI)**
5. **All messages are XML-formed**

More material

http://virgo.sourceforge.net/

Question?

Thanks