Data Grids: A Collaborative Semantic Model with Hybrid Namespace

Prof. Ahmed Sameh

Department of Computer Science, The American University in Cairo

Dalia El-Mansy

The Southern Methodist University

Texas, USA

Presentation at the Gracow Grid Workshop 2006-
Poland, October 15-18, 2006

Outline

- Introduction
- Grid Computing Survey
- Problem Definition
- Proposed Model
- Merits of the Proposed Model
- Design and Implementation
- Experimental Setup and Results
- Conclusion and Future Horizons

Introduction

- The Computational Grid
  - Origin
  - Virtual Organizations (VO)
  - Challenges
  - Middleware
- The Data Grid
  - Definition
  - Architectural Aspects
  - Requirements for Data Grid
  - Functional Design
    - *Core Data Grid Services*
    - *Higher-Level Data Grid Components*
- The Semantic Grid

Origin of the Computational Grid

- Huge Data
- Heavy Processing
- Wide Geographic Distribution
- Resource Uniform Access
- Resource Transparent Access

Virtual Organizations (VO)

- Aggregation on for resource sharing
- Collaborative resource ownership/access
- Goals for resource access :
  - Ease
  - Transparency
  - Coordination
  - Security

The challenges of the Grid

- Information services
- Resource Brokering
- Uniform access to resources
- Security
- Job scheduling
- Data Access
- Data Replication

Grid middleware

- The mid-level software that provides services to users and to the applications.
- E.g. Globus

Data Grid Definition

- The data grid is an integrating infrastructure for distributed computation that allows us to identify requirements and components common to different systems and hence apply different technologies in a coordinated fashion to a range of data-intensive petabyte-scale application domains.

Architectural Aspects

- No specific architecture or topology that characterizes data grid
- Hierarchical architecture is adopted merely to make search easier and faster

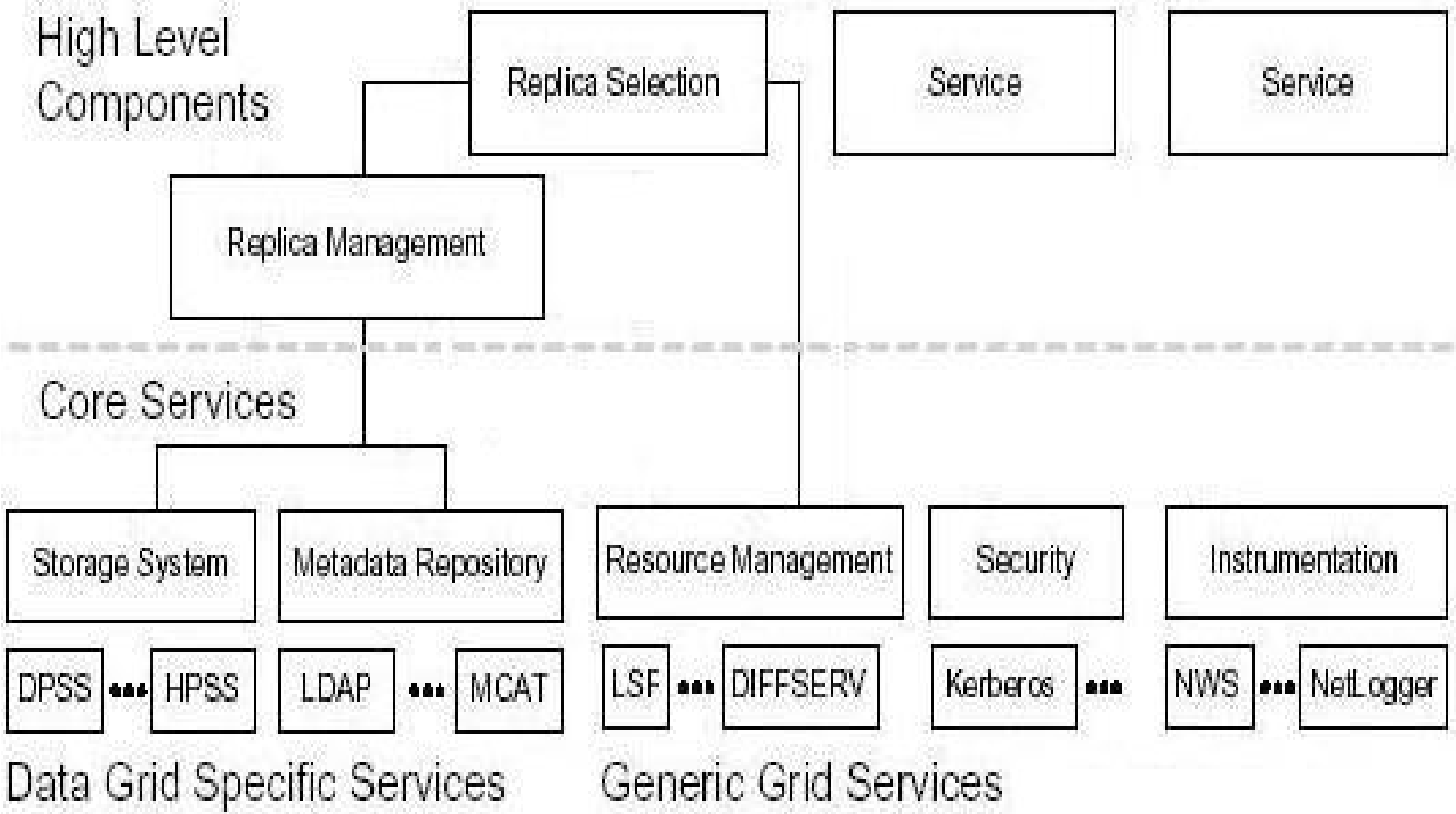
Requirements for Data Grid

- Data files' replication.
- Grouping multiple data resources as single (compound) data entity.
- Defining and describing data by metadata.
- Data identity, ownership and versioning.
- Data publish, retrieve, search & discover.
- Keeping data provenance records.

Functional Design

- **Core Data Grid Services**
  - Authorization and authentication
  - Resource reservation and co-allocation
  - Performance measurements and estimation
  - Instrumentation
  - Directory Services
- **Higher-Level Data Grid Components**
  - Replica management
  - Replica selection and data filtering

Data Grid Architecture



Data Grid Layered Architecture

Grid Computing Survey

- Historical Review
- The Evolution of Grid and P2P Computing
- Sample Grid Projects and Research Works
- Concluding Remarks on Previous Work
- Motivation

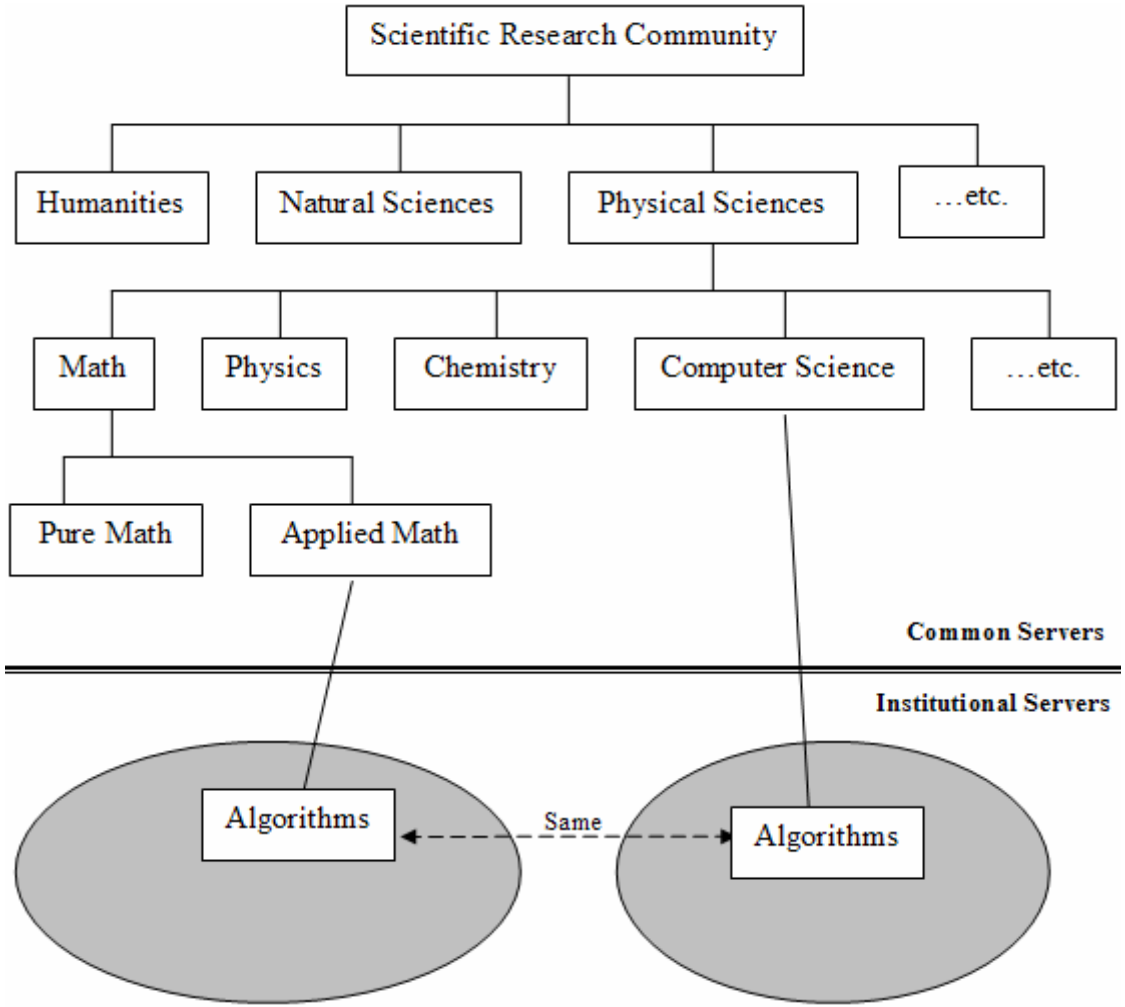
Sample Grid Projects

- AppLeS: A Network Enabled Scheduler
- Condor: Cycle Stealing Technology for High Throughput Computing
- Data Grid
- Globus: A Toolkit for Grid Computing
- Javelin : Java based infrastructure for internet-wide parallel computing.
- Legion: A Grid Operating System
- MOL: Metacomputing Online Kernel
- NetSolve: A Network Enabled Computational Kernel
- Ninf: A Network Enabled Server
- PUNCH: The Purdue University Network Computing Hubs
- Nimrod-G Grid Resource Broker
- Giggle: A Framework for Constructing Scalable RLSs
- myGrid : UK e-science project – A Semantic Grid

Problem Definition

- Data Grid functional design assumes uniformity of information infrastructure intended to sharpen the collaboration operability but narrows the scope of usability by missing possible cooperation opportunities
- Sometimes Data Grid Contributors fail to follow a unified taxonomy or naming system. *Example Scientific Research Community*
- Data Grid projects are always isolated islands
- Contributors do not recognize different degrees of similarity. Therefore, they do not benefit from it
- They could miss cooperation opportunities just because they have different taxonomies
- A wider collaborative model is needed to glue contributors and data grids in a **relaxed aggregation**
- Inter-data grid communications are needed

Example : Scientific Research Community



The field of Algorithms can be classified differently in different institutions

Proposed Model

- Defining the model
  - Overview
  - Introducing data grids to the model
  - Wider scope collaboration
- Usability of the model
- Security of the model
- The design goals of the model

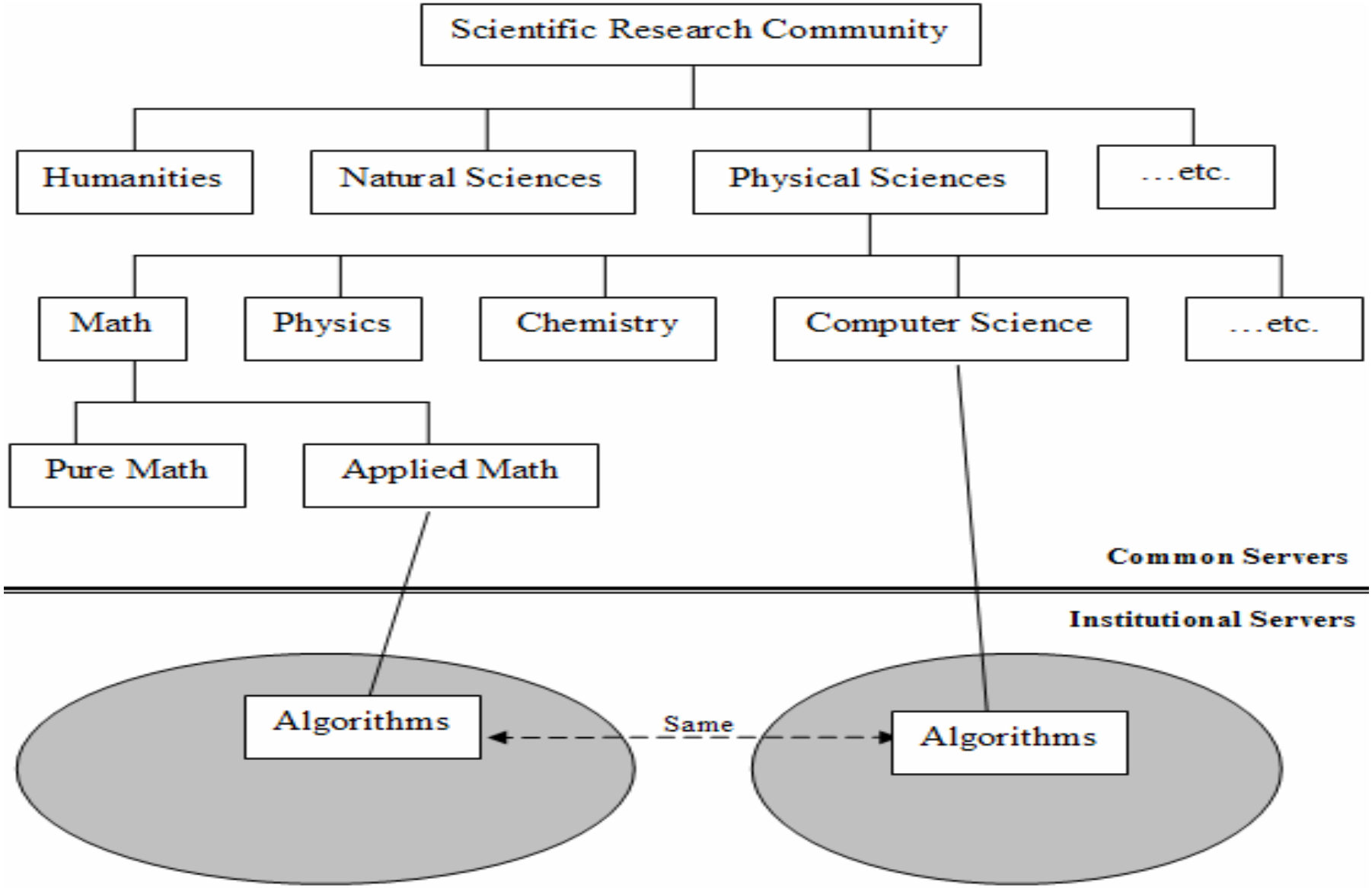
Proposed Model

- A wider collaborative model is designed in an effort to solve the problem
- A middle ground solution that let the research centers in different institutions follow their preferred taxonomies while automating the effort the researchers pay when looking around for knowledge
- Unifying the upper part of the hierarchical namespace that is common worldwide.
- Then let research centers join on that basis and start their different classifications.
- The common part of the namespace is represented by a hierarchy of huge servers.
- Institutions' servers branch from any level of the common namespace server hierarchy
- Transitive non-recursive "Same" relations to link equivalent institutional servers.

Proposed Model (ctd.)

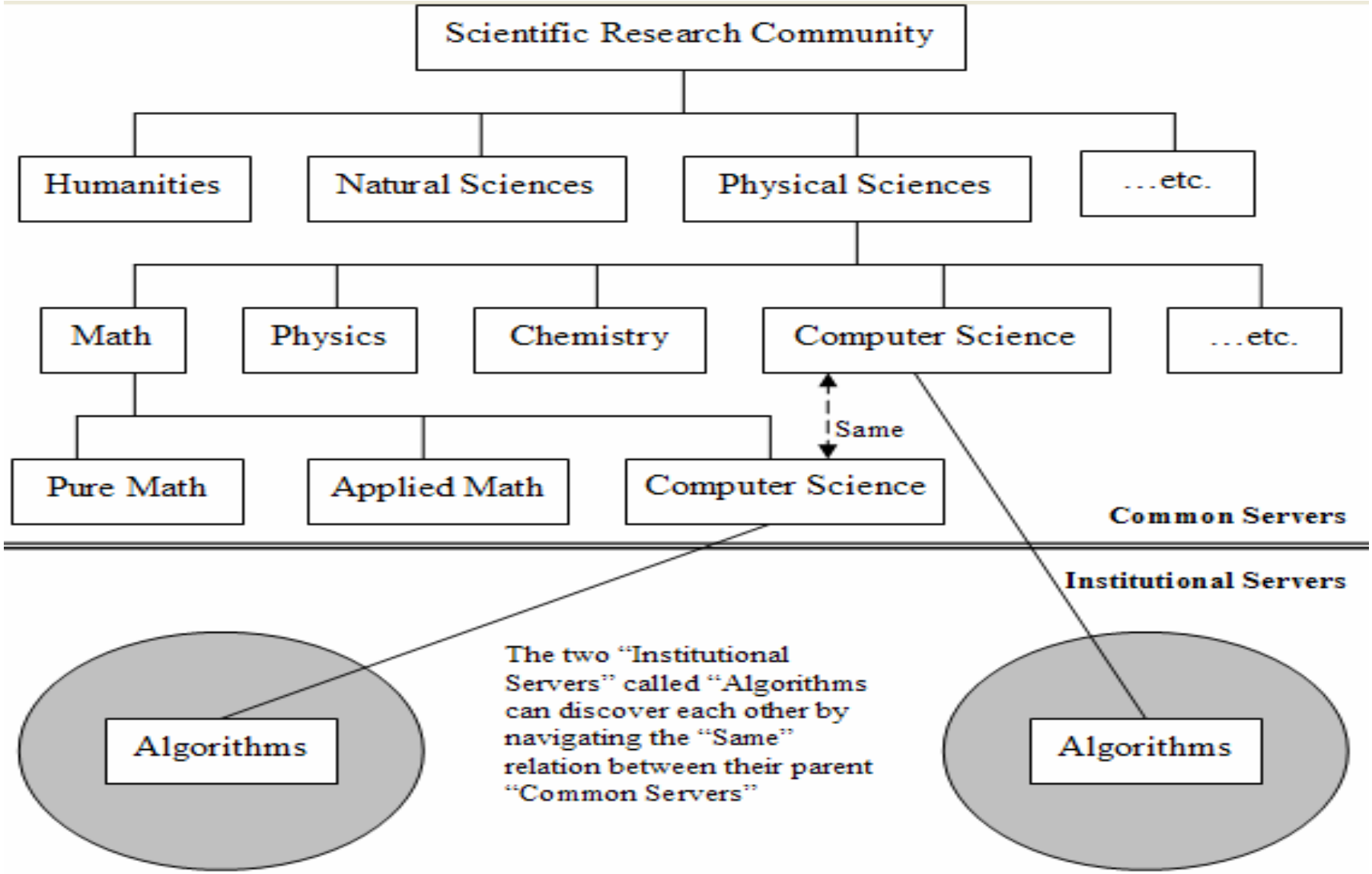
- “Same” relations between “Common Servers” will be navigated from the underneath servers to find possibly “Same” servers in other parts of the hierarchy (same topics classified differently by different institutions)
- Explorative tools will be used by new servers to nominate candidate “Same” servers. Administrators, then, should study the decision of applying for “Same” relations with the nominated servers. The nominated servers are supposed to expose a brief narrative expression of their interests
- Dynamic configuration can be done to optimize the search cost (***Deepening***). Eg. if the administrator of a "Common Server" noticed that many of its direct "Institutional Server" children are linked with “Same” relations (representing a specific branch of science) then he can take the decision of creating a direct child “Common Server” for that branch of science then inform all the “Same” “Institutional Servers” to follow the new taxonomy
- The Grid concept is introduced to our model through the three kinds of relations: "Child", "Same" and "Similar". "Child" relation maintains a simple hierarchical namespace (taxonomy) and the other two relations maintain gray-scaled hybridization to the namespace.

Proposed Model: Common and Institutional Servers



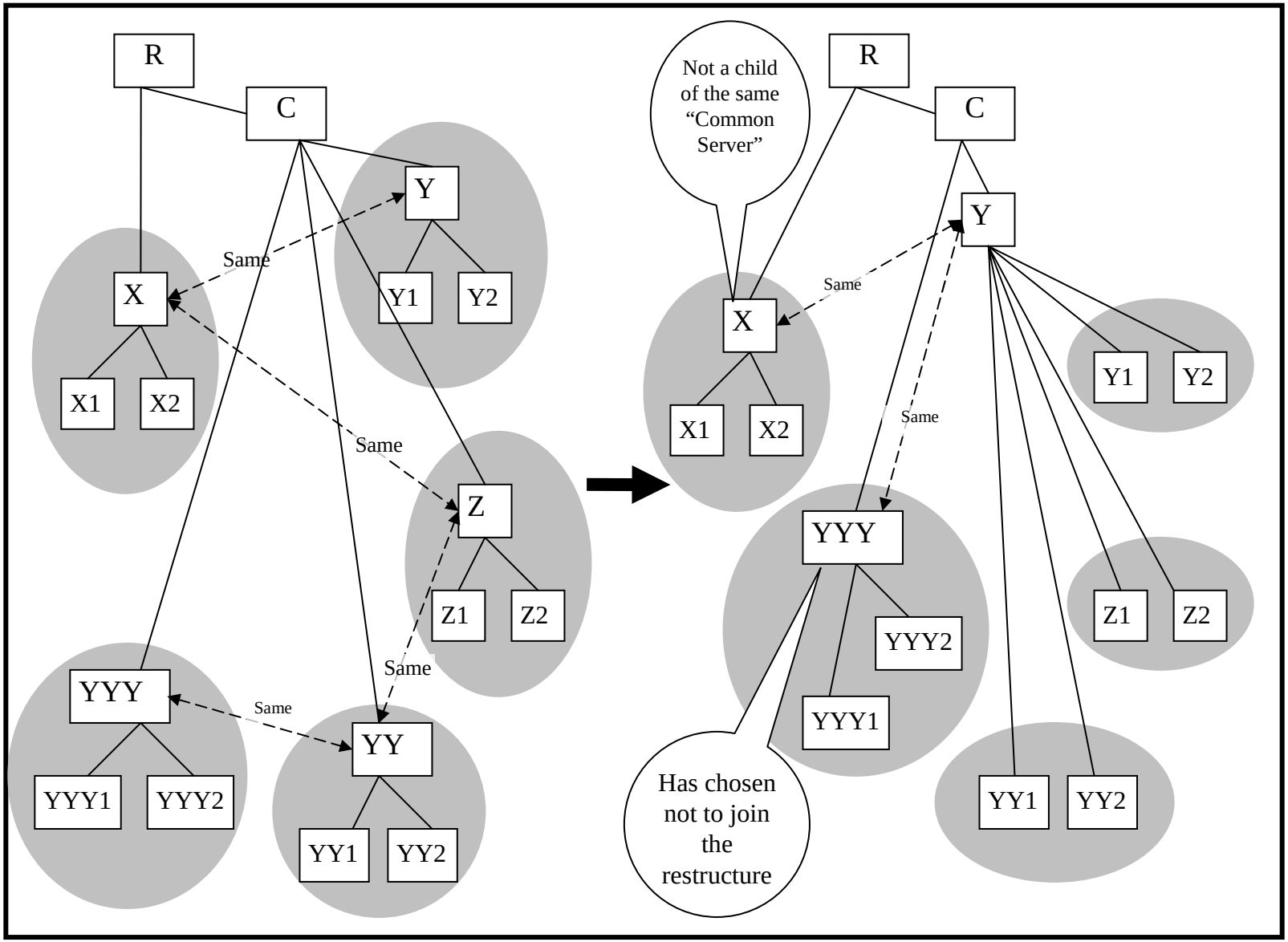
Common Servers and Institutional Servers and “Same” Relation

Proposed Model (ctd.)



Utilizing "Same" relations between "Common Servers"

Proposed Model: Deepening



Common server replacing Same Institutional Servers

Wider Scope Collaboration

- The data grids established by “Same” “Institutional Servers” collaborate, as whole data grids, in a bigger collaboration model.
- They benefit from the common name that will prefix all of their resources’ names.
- Requests in the wider collaborative model use the full resource names.
- Requests in small data grids will use the logical names without any prefix (they are traditional data grids).

Wider Scope Collaboration (ctd.)

- The larger collaborative model contain many data grids
- “Similar” “Institutional Servers” in different data grids are intentions for wider collaboration between them
- Data grids establish the wider collaborative model easily because the resource full names are unique
- Proxy agents can reside into the data grid servers to represent the other data grids
- The agents maintain the resource updating on the servers they represent by applying a suitable refresh rate according to the dynamics of those resources.

Wider Scope Collaboration (ctd.)

- The info services and resource brokerage logic of each data grid will be encapsulated into the proxy agent it exposes.
- Actually, the proxy agent, as defined by this model, is a unified interface of inter-data grid gateways.

Usability of the Model

- Explorative tools can be designed to help users navigate the huge hierarchy of common and institutional servers
- Search results are sorted according to relevance
- Sidekicks can be performed by explorative searches following “Similar” relations to each “Same” server where the resource is found

Usability of the Model (ctd.)

- A user hooked to an “Institutional Server” can login as a user of the small data grid which his server is a member of, and he will be using non-prefixed logical resource names
- He can also login as a user of a wider collaborative model which his local data grid is a member of, and he will be using prefixed logical resource names
- He can login also as a free user to surf the whole hierarchy of servers using the explorative tools to discover institutions, data grids and wide collaborations. This surfing will help server administrators make decisions on joining data grids and creating/deleting “Same” and “Similar” relations with other servers.

Security of The Model

- Data grids are responsible of their security.
- They need to implement their own security models on their servers and on the stubs that handle their exported proxy agents as well
- The model depends on the directory server security system
- The model's security will adopt recursive trust scheme for extensibility
- Each server brokers its children's authenticity at its parent's side when broadcasting their requests
- It also brokers its authenticity at its child servers when multicasting its parent's requests to them
- A server, then, inherits its authenticity from its parent server while imposing its authenticity on its child servers
- **This decentralized authentication pattern guarantees no bottlenecks known in the centralized security models**

The Design Goals of the Model

- To make the collaborative model as much attractive for servers from different institutes to join as possible
  - Allow existing data grids to join as is
  - Contributors that are currently unable to establish data grids will be able to establish relaxed collaborations through “Similar” relations
  - They can also negotiate “same” relations with each other so that they can establish data grids

Merits of the Proposed Model

- No added burden on the existing data grids to join
- It does not interfere with the data grid themselves
- The Data Grid manager server has no role to play

  need not be a member of the model at all
- Data grids can join even partially
- The servers in the model store the IDs of the Data Grids (recorded by the model) that they are members of
- A server contributing in our model can join as many Data Grids as it wants. It can be even a member of external Data Grids that are not recorded by the model
- All of these flexibilities make the decision to join the model “theoretically costless”
- Server and existing data grids have nothing to lose and everything to gain by joining this model
- Semantics increases interoperability between our model and other models' agents searching the web for information

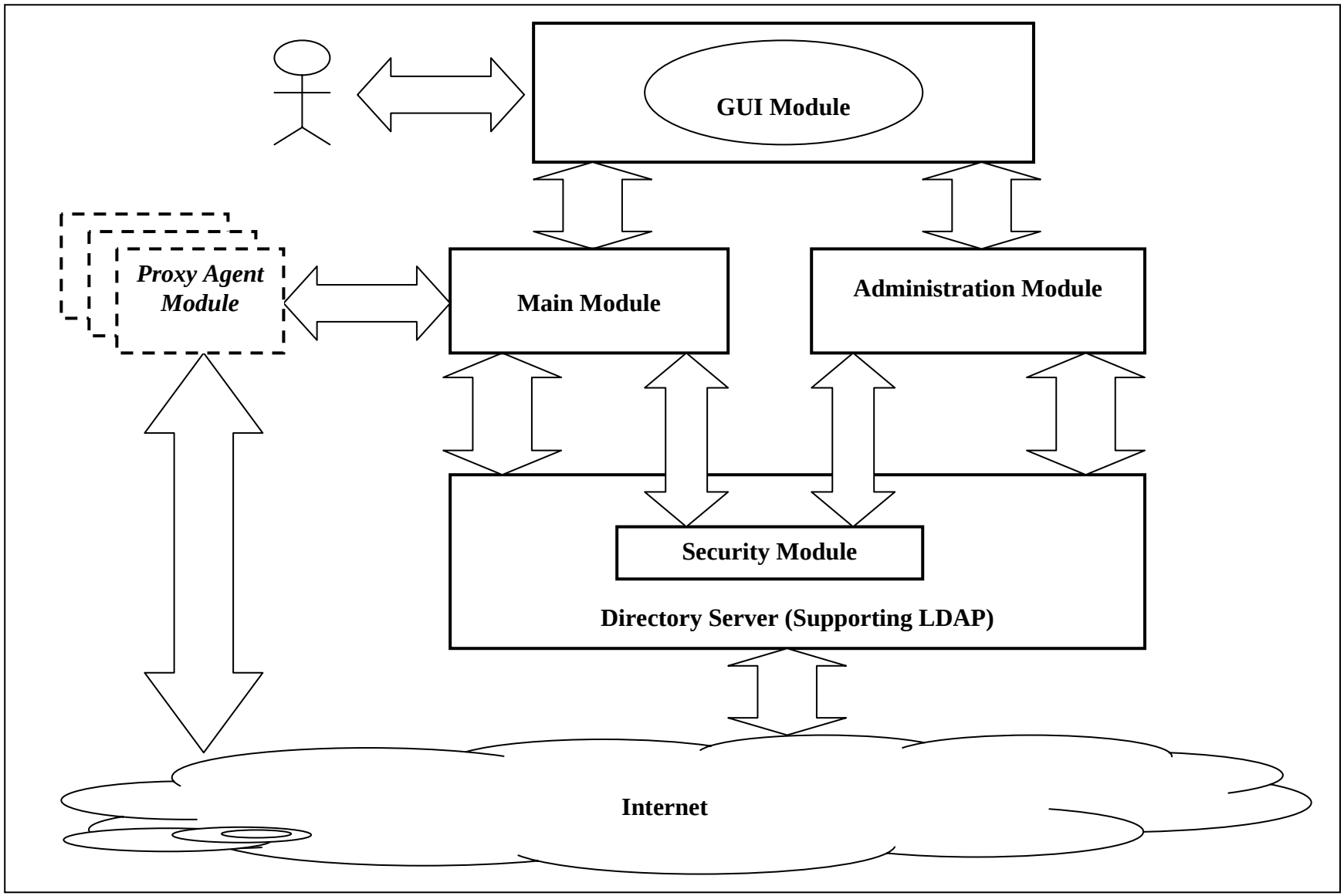
Contributions

- **Costlessness**:
  - Simple interface
  - No preparative steps needed
- **Comprehensiveness**:
  - All players can find each other and the existing data grids easily
- **Wideness**:
  - Inter-data grid brokerage for wider collaborative scope
- **Hybridization**:
  - Gathering contributors with different naming schemes (taxonomies) in one collaboration model
- **Promotion**:
  - Encouraging the unification of the namespace (deepening the common part)
- **Flexibility**:
  - Data grids can join even partially
- **Abstraction**:
  - Data grids encapsulate their details
- **Scalability**:
  - The decentralized approach helps the model to grow freely, The limited scope of each server makes it easy to build huge hierarchy since that no centralized authorities are consulted
  - Servers store logical coordinates (IDs) for "Same" and "Similar" servers only so moves easy
  - The scalability is maintained also on operational level; the processes that involve huge numbers of transactions (like exploring servers, data grids and resources) are distributed over the hierarchy

Measure of Success

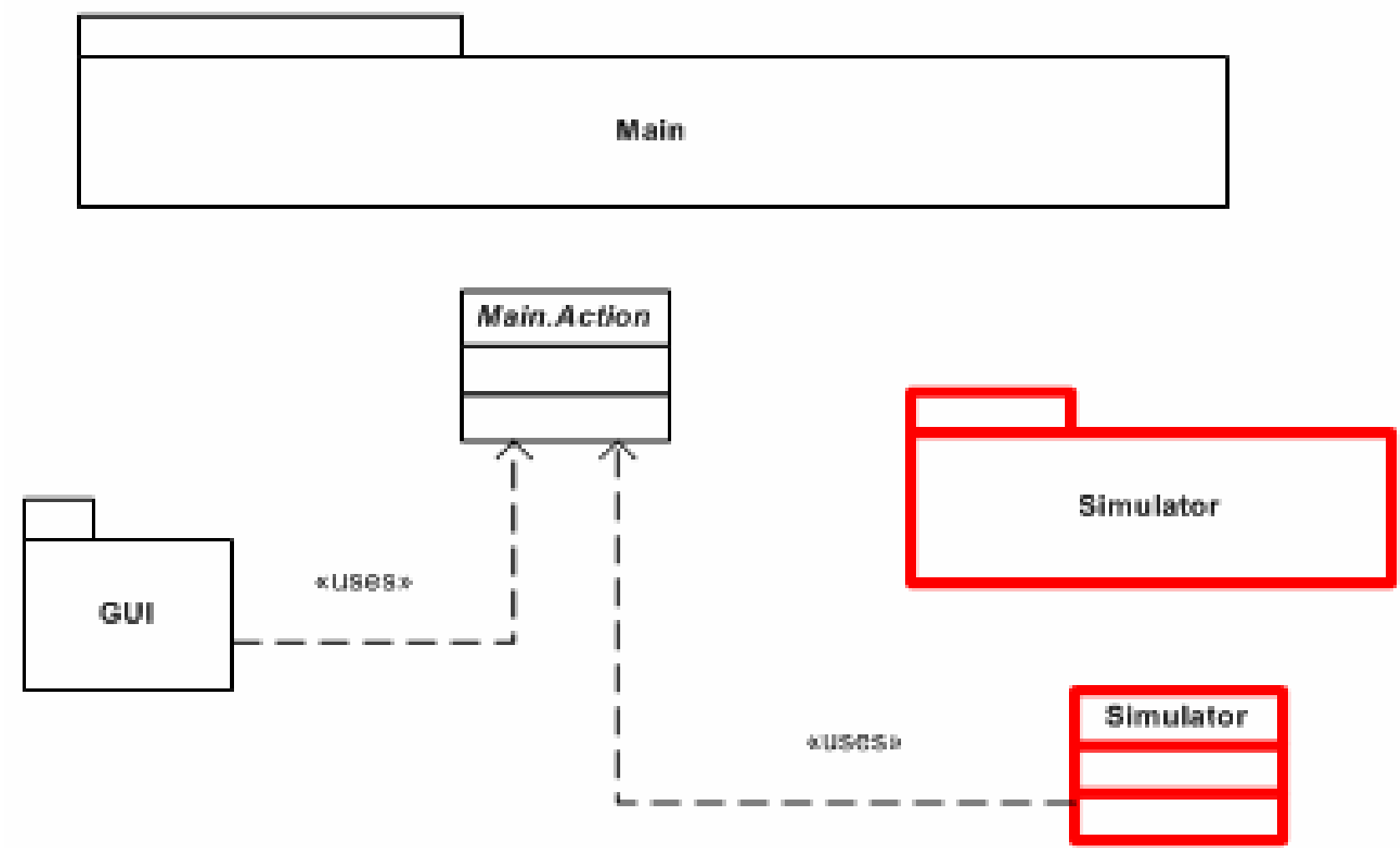
- To measure the success of the model, we can find a sponsor and implement the model for a related field (as per the sponsor's interest).
- Then we can advertise the model aggressively among all candidate contributors.
- The success can be reflected by the number of join applications and requests for further information about the model.
- Alternatively, we can build and advertise an Internet site that explains the model and poll opinions and willingness of joining.

Design



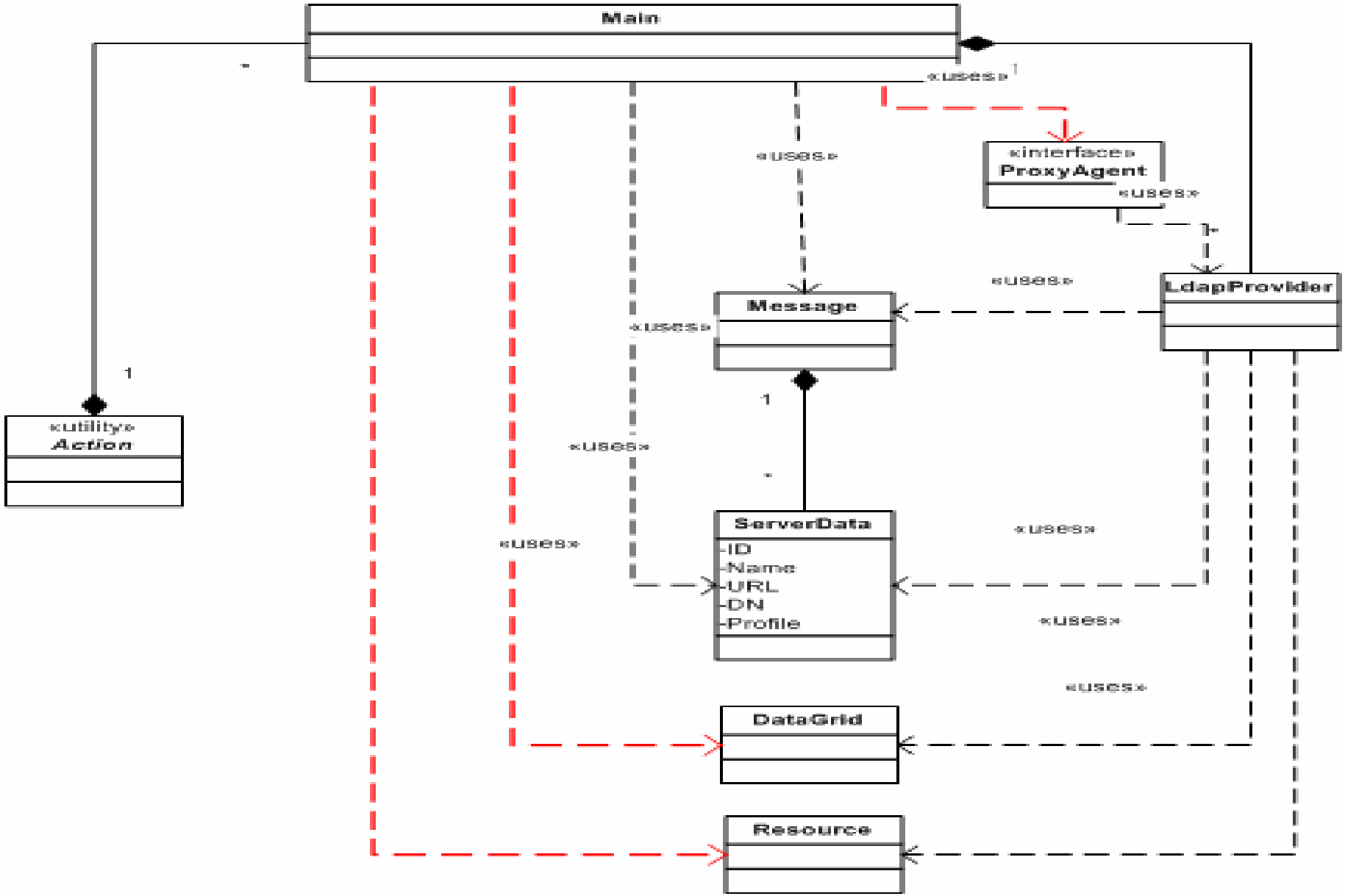
High Level Server Architectural Design

Design



High Level Class Diagram

Design



Detailed Class Diagram

Design(ctd) Server Architectural Design

- Main Module: The central daemon that should be always running. It encapsulates the directory services for flexibility. (Any technology, like LDAP, can be adopted by the servers while not affecting the whole model). Also, it limits the security boundaries to a narrower zone.
- Communications Module: This module wraps the used communications protocols like TCP/IP and the like. It should expose the communications functions like “SendMessage(Msg)”, “ReceiveMessage()”, HandleMessage(Msg).
- Security Module: This module handles the authentication and verification issues. It will store the server identity and all the local users’ credentials.
- GUI Module: This module allows the user to use the provided functionalities.
- Administration and Maintenance Module: Used by administrators to configure, set policies, deploy, recover, troubleshoot…etc.
- Proxy Agent Module: This is a framework for Data Grids to export their data content to each other.

Design (ctd.): Server Functional Design

- **Data Items**
  - ID: Unique id reflects the server location in the hierarchy
  - Name: A string to characterize the server content
  - **Type**: Type of Server C: Common, I: Institutional
  - **Profile**: A brief on the server's mission (interests, goals)
  - **Ontology**: Describes the server's mission in machine-readable form.
  - **URL**: The URL of the server.
  - **ParentURL**: The URL of the parent server.
  - **ChildList**: A list of the child servers (IDs and URLs)
  - **SameList**: A list of server IDs that are in direct "Same" relations with the server
  - **SimilarList**: A list of server IDs that are in direct "Similar" relations with the server
  - **DGRegistry**: A list of DGIDs of the Data Grids registered by the server with their profiles, Proxy Agents' code and manager server URLs. (This occurs only in Common Servers.)

Design (ctd.) Server Functional Design

- **Server Functions**
  - ***URL_FromID(ServID):*** Returns the physical location (URL) of the Server identified by ServID (the logical location inside the hierarchical name space)
  - ***IDFromURL(ServURL):*** Logins the server pointed by ServURL as guest and returns its ID
  - ***SameGroup():*** Creates the recursive collection of SameList on all the Same Servers (in the local SameList)
  - ***SimilarGroup():*** Same as for SameGroup() but for "Similar" relation.
  - ***IsSame(ServID):*** Returns true if ServID is in SameGroup
  - ***IsSimilar(ServID):*** Returns true if ServID is in SimilarGroup
  - ***AddChild():*** Creates an id by suffixing ID with ChildSequence (after incrementing it by one) then adds it to ChildList. It returns the id of the added child
  - ***AddSame(ServID):*** Adds the server identified by ServID to the SameList if not already a Same Server
  - ***AddSimilar(ServID):*** Adds the server identified by ServID to the SimilarList
  - ***PackMessage(FromServ, ToServ, ReplyToRefNo, Subj, Body):*** Returns a Message object populated with the parameters with RefNo set to the next SequenceRef and the field sentDateTime set to the system date and time

Design (ctd.) Server Functional Design

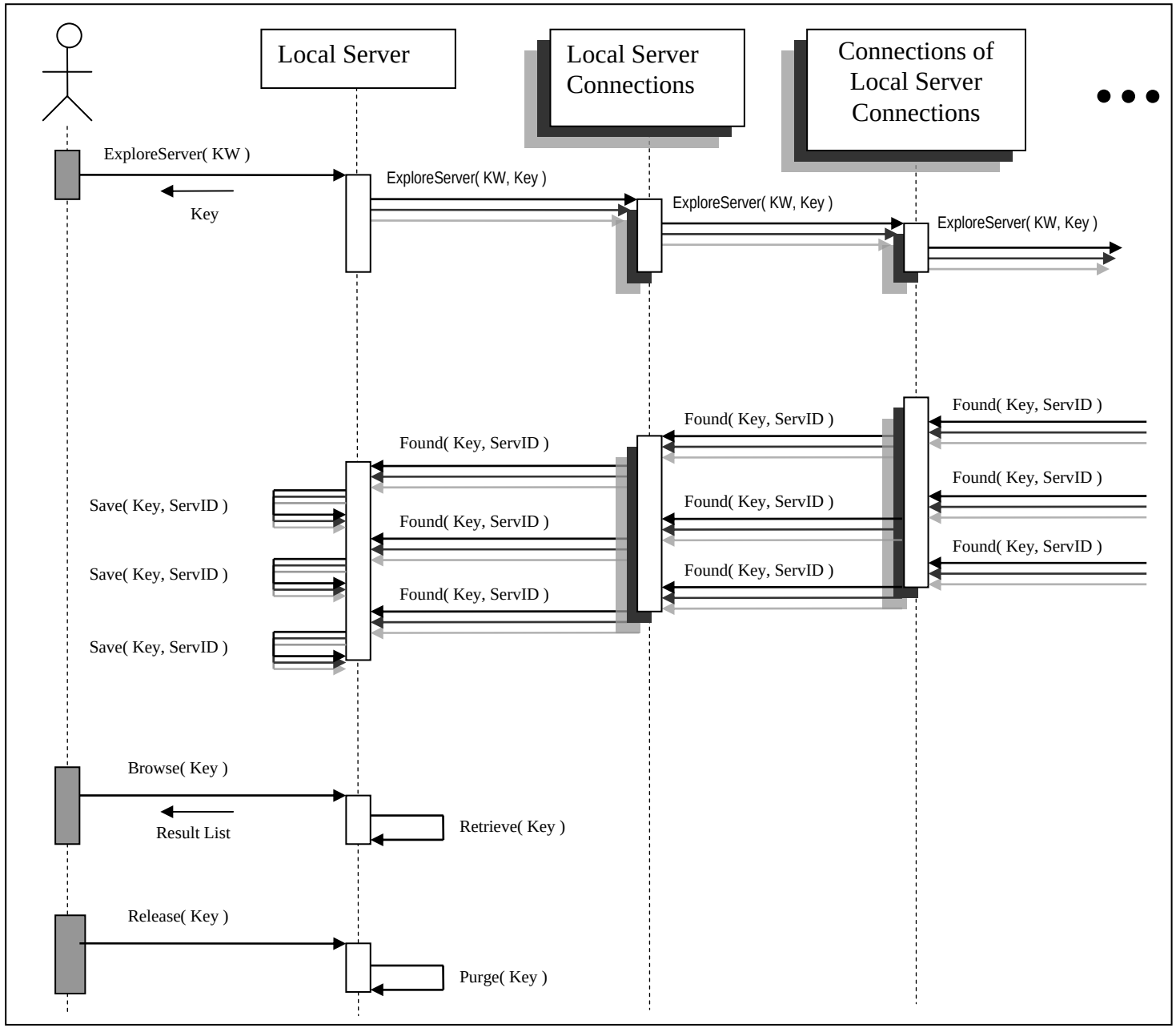
- **Server Functions**
  - ***SendMessage (Msg)****:* It attempts to login the server pointed by Msg.to.url as guest. If successful, it sets Msg.status to New and Msg.receivedDateTime to system date and time then it stores Msg in the inbox of that server. If the last steps were successful, it attempts to set Msg.status to "Sent" and store it into the SentBox of the local server. All these steps should be done into a transaction to maintain data consistency
  - ***FetchMessage():*** Returns a message that has arrived from another server (if any). The main loop of the Main Module will hook into this function periodically to fetch new messages
  - ***ReadMessage(Folder, Index):*** Returns a message from Folder at Index if any
  - ***HandleMessage(Msg):*** Message handler for received messages
  - ***RequestChild():*** Requests establishing a "Child" relation with the server pointed by ParentURL.
  - ***RequestSame(ServID):*** Requests establishing a “Same” relation with the server identified by ServID.
  - ***RequestSimilar(ServID):*** Requests establishing a “Similar” relation with the server identified by ServID.
  - ***FetchRequest():*** Returns a message from RequestsQueue if any
  - ***HandleRequest(Msg):*** Handles a request coming from another server

Design (ctd.) Server Functional Design

Server Functions

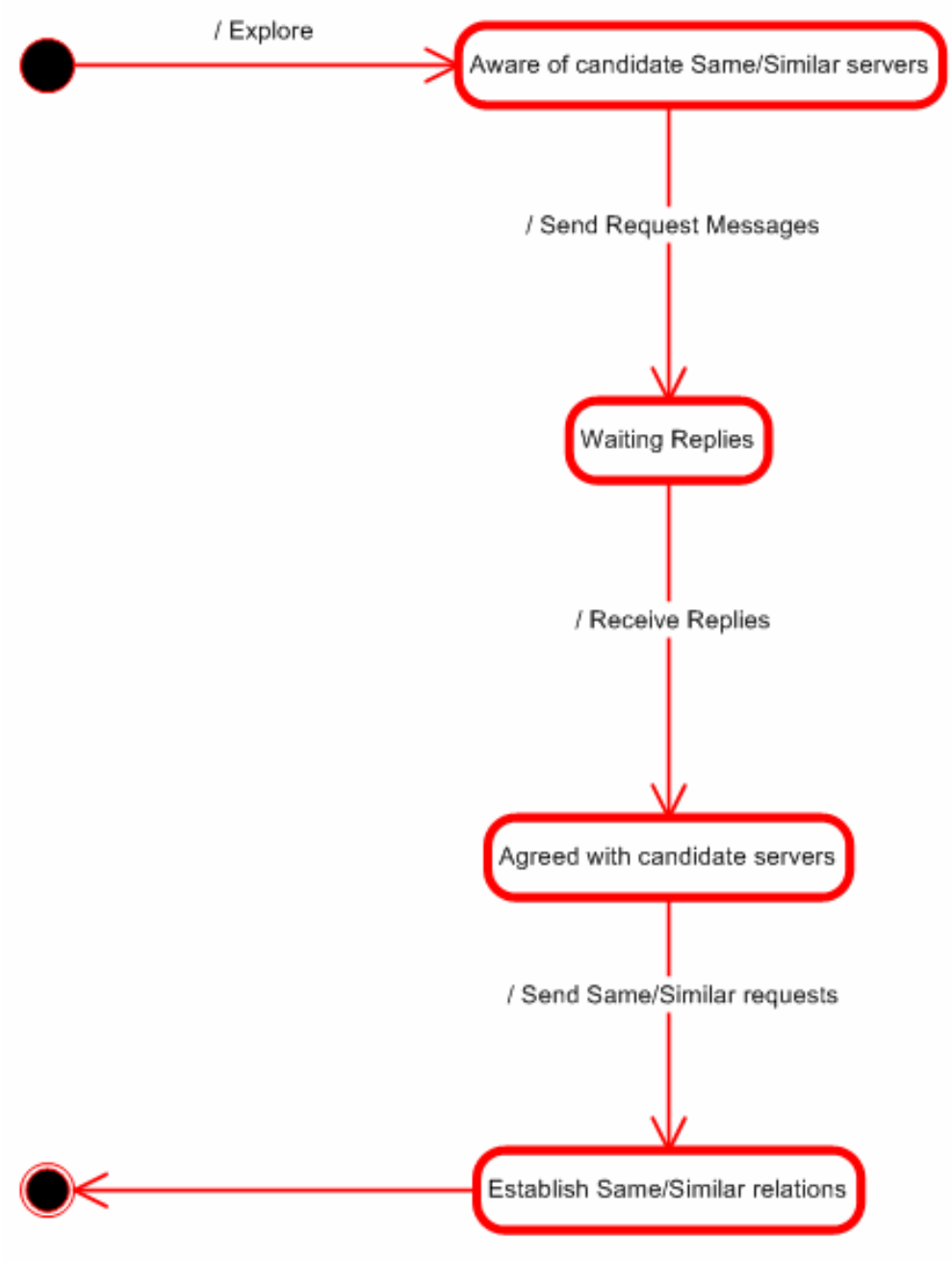
- ***RegisterDG(DGID, MgrSrvURL, Profile, ProxyAgent):*** Creates a new entry in the Data Grid Record after assuring the uniqueness of DGID
- ***QueryResource(Res):*** If Name is prefixed with a Server ID or a DGID, it consults the corresponding server or Proxy Agent, otherwise, it propagates the query to all proxy agents on board and servers in SameList
- ***QueryDGResource(ServID, DGID, ResName):*** Logins the server identified by ServID and downloads the proxy agent of the data grid identified by DGID from it then queries ResName from that proxy agent. (the data grid DGID is supposed to be registered on ServID)
- ***QueryServerResource(ServID, ResName)***
- ***QueryResourceAlias(ResName)***
- ***QueryServerResourceAlias(ServID, ResName):*** Queries the resource alias ResName on the server identified by ServID

Major Server Components: Explore Server Tool

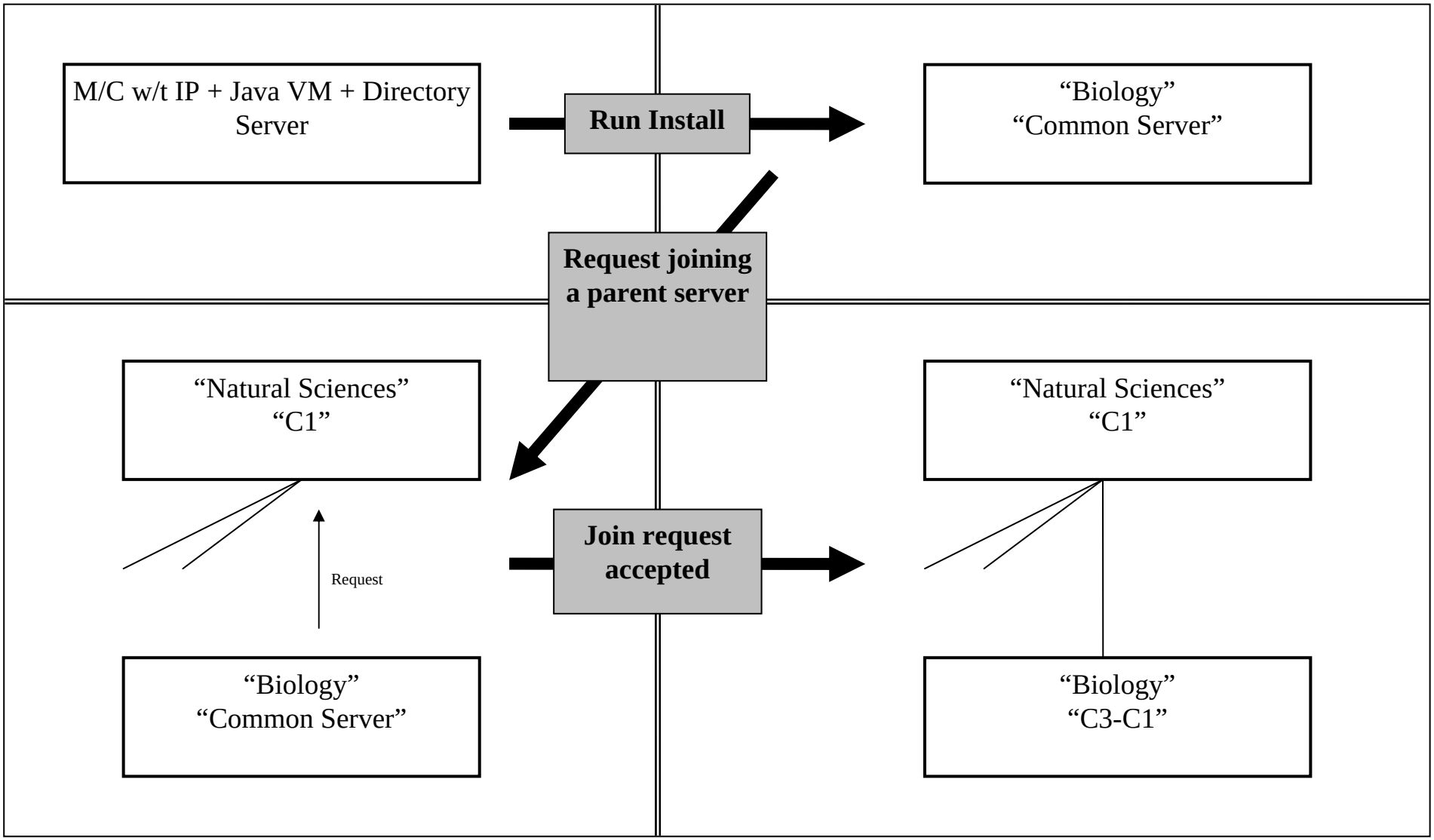


Sequence Diagram

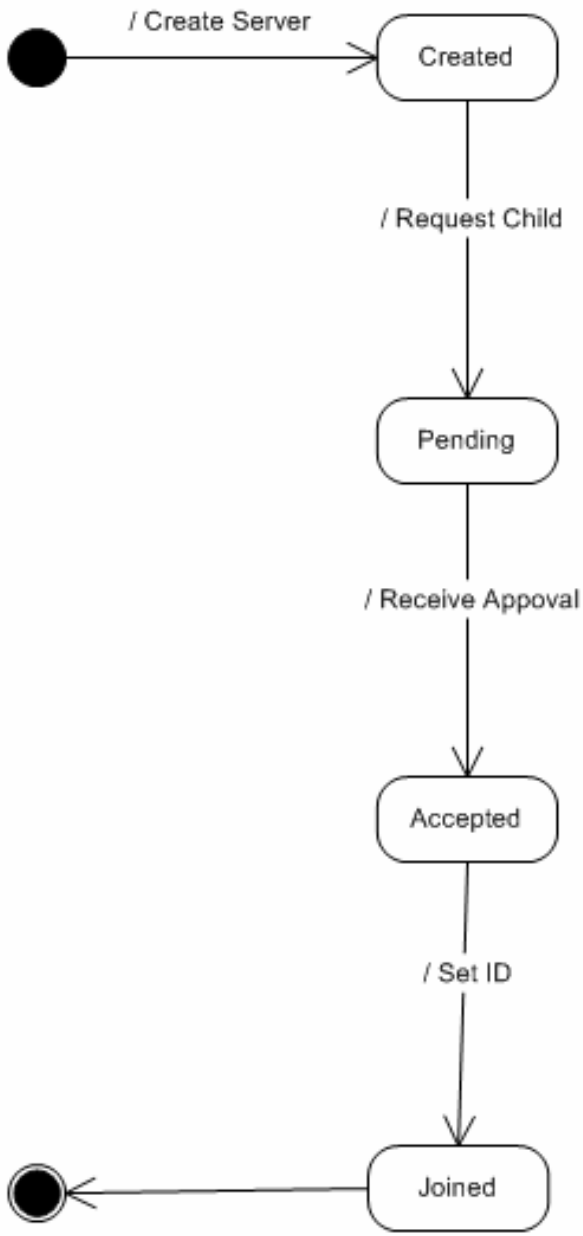
Major Server Components: Explore Server Tool *State Diagram*



Use Scenarios: *Installation (Admin)*

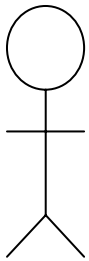


Use Scenarios: *Setup (Admin)*



Setup state diagram

Use Scenarios: *Resource Discovery (users of leaf servers)*



User connected as a data grid user

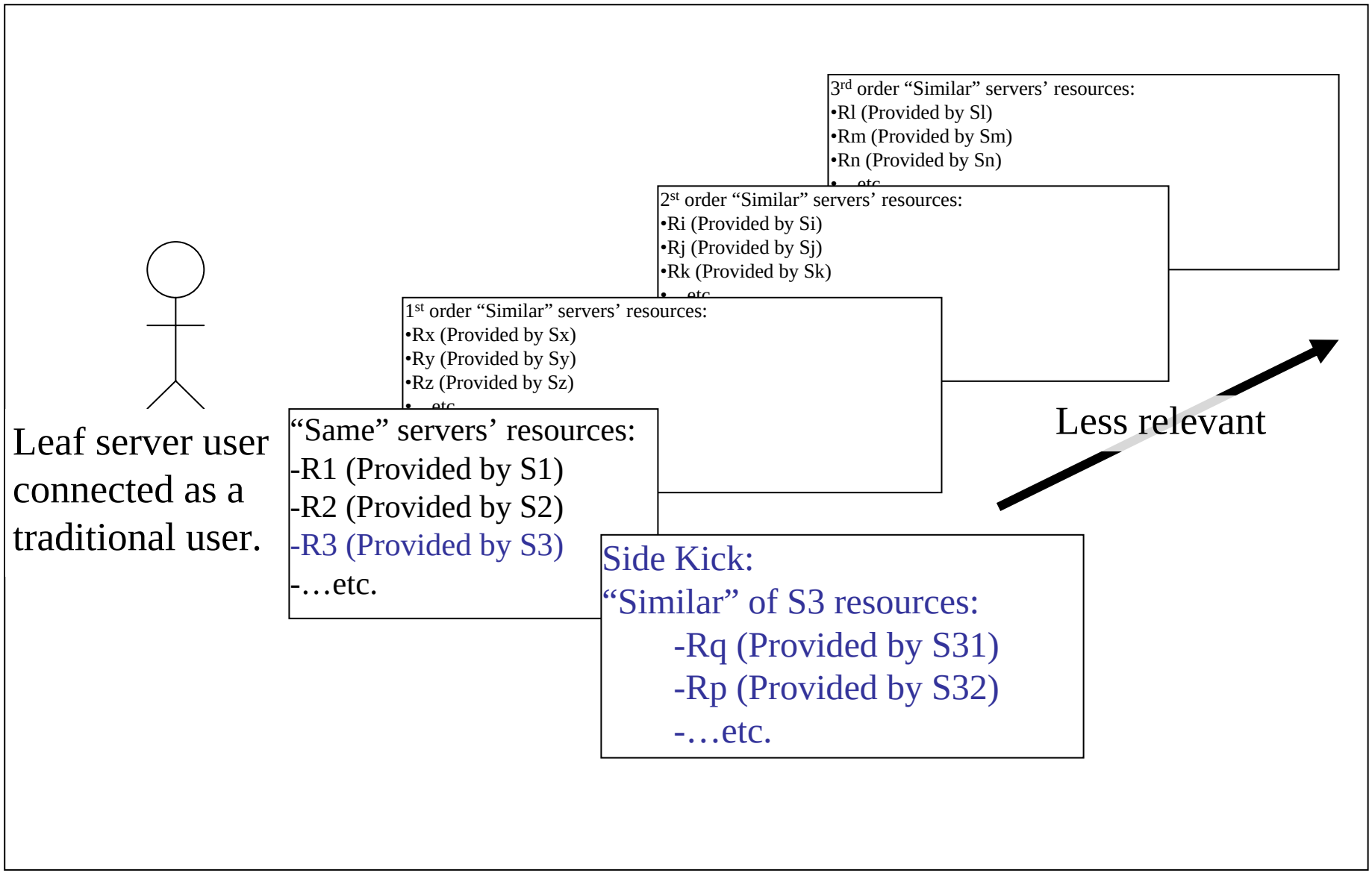
Local data grid resources:
XX-Data

Server resources:
S1.XX-Data
S2.XX.Data
S3.XX.Data
…etc.

Other data grids' resources (Provided by proxy agents):
DG1.XX-Data (Provided by DG1_Proxy_Agent)
DG2.XX-Data (Provided by DG2_Proxy_Agent)
DG3.XX-Data (Provided by DG3_Proxy_Agent)
…etc.

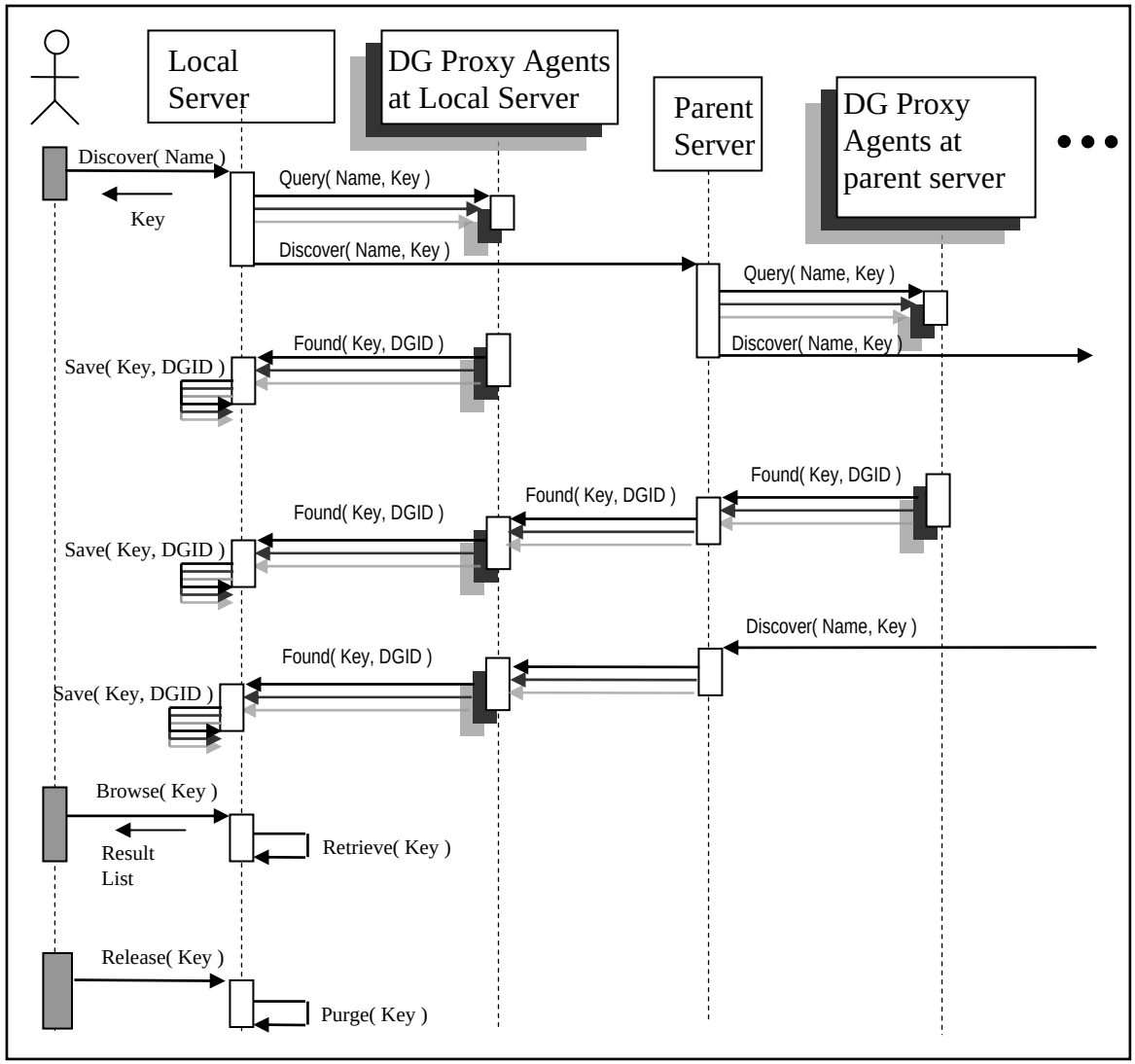
Data Grid Resource Discovery

Use Scenarios: *Resource Discovery (users of leaf servers)*



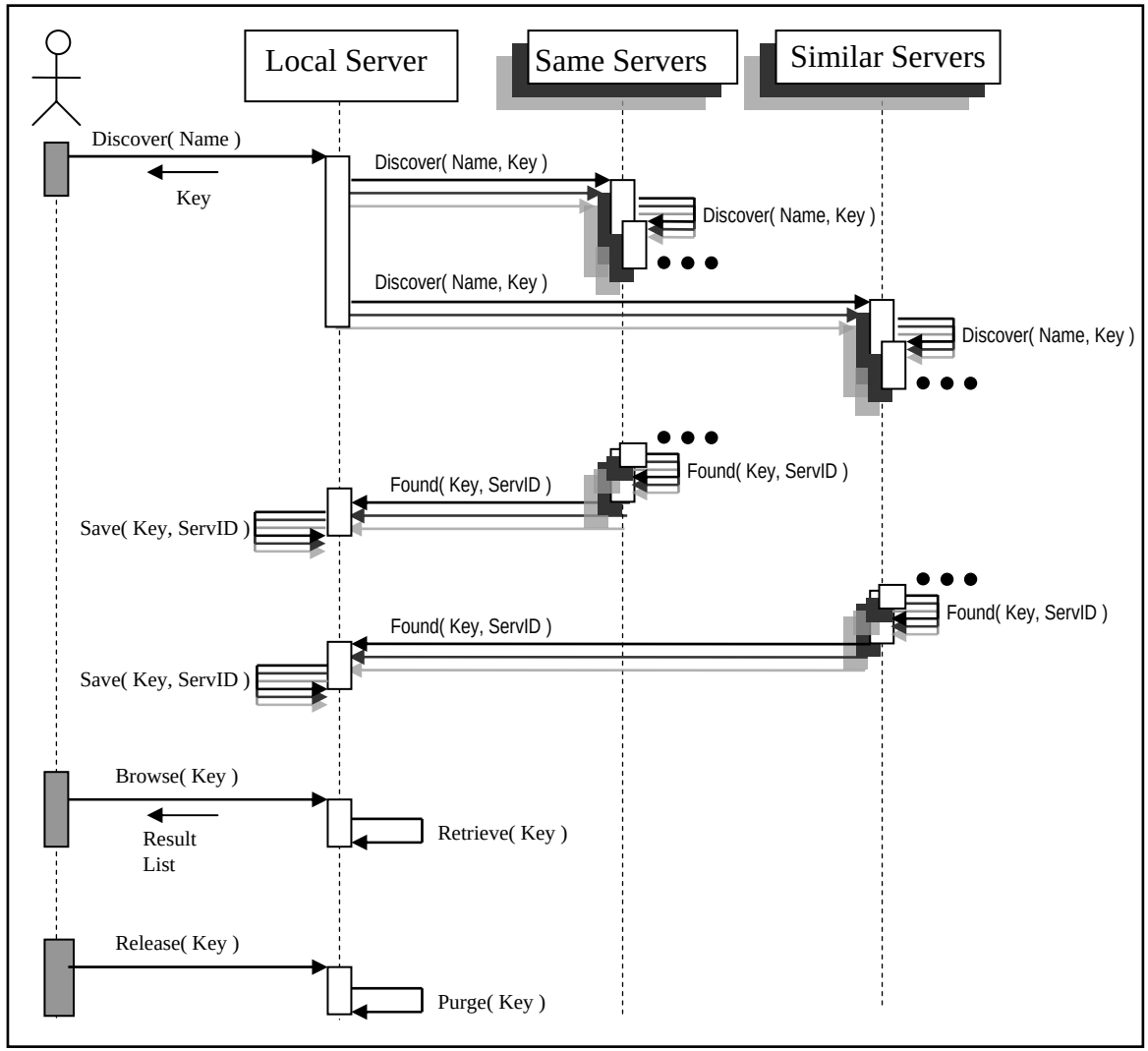
Server Resource Discovery

Use Scenarios: *Resource Discovery (Data Grid User)*



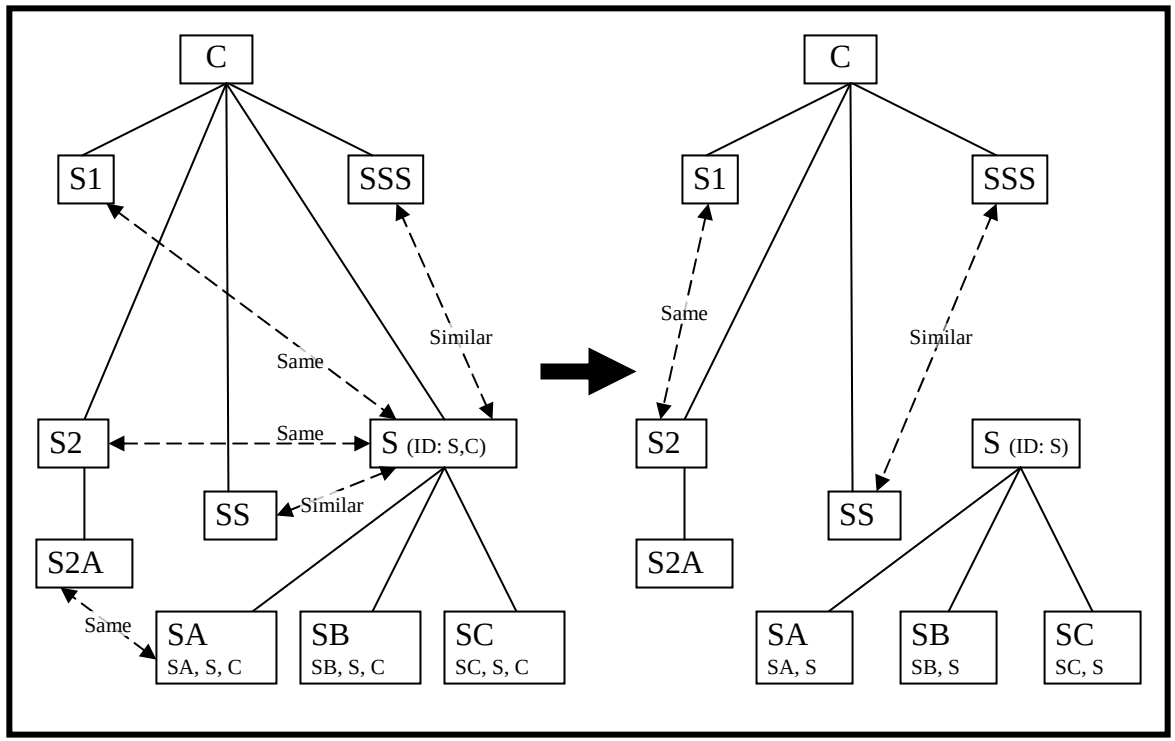
Sequence Diagram

Use Scenarios: *Resource Discovery (users of leaf servers)*



Sequence Diagram

Use Scenarios: *Leaving The Model (Admin)*



Server S is leaving the model

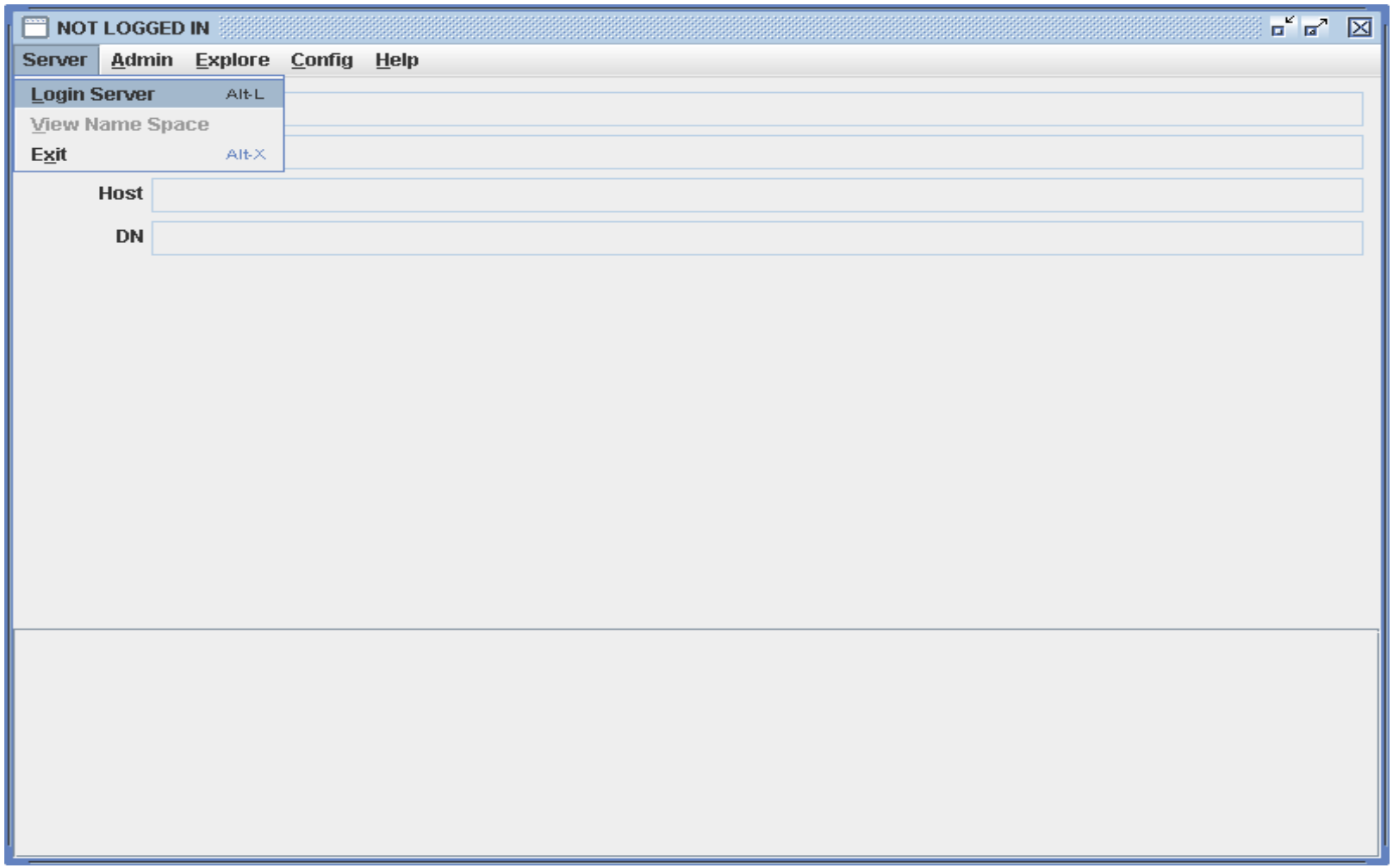
Implementation

- All the Internet Taskforce Standards have been adopted in the implementation: Java is used for development, LDAP for directory services and XML will be used for data exchange when third parties (application developers) develop add-ins in the future (e.g. more relations other than "Same" and "Similar").

- All modules of the server software have been written in Java for heterogeneity and platform independence. For the proof-of-concept purpose in this thesis, a simulator module was also developed to simulate many servers.

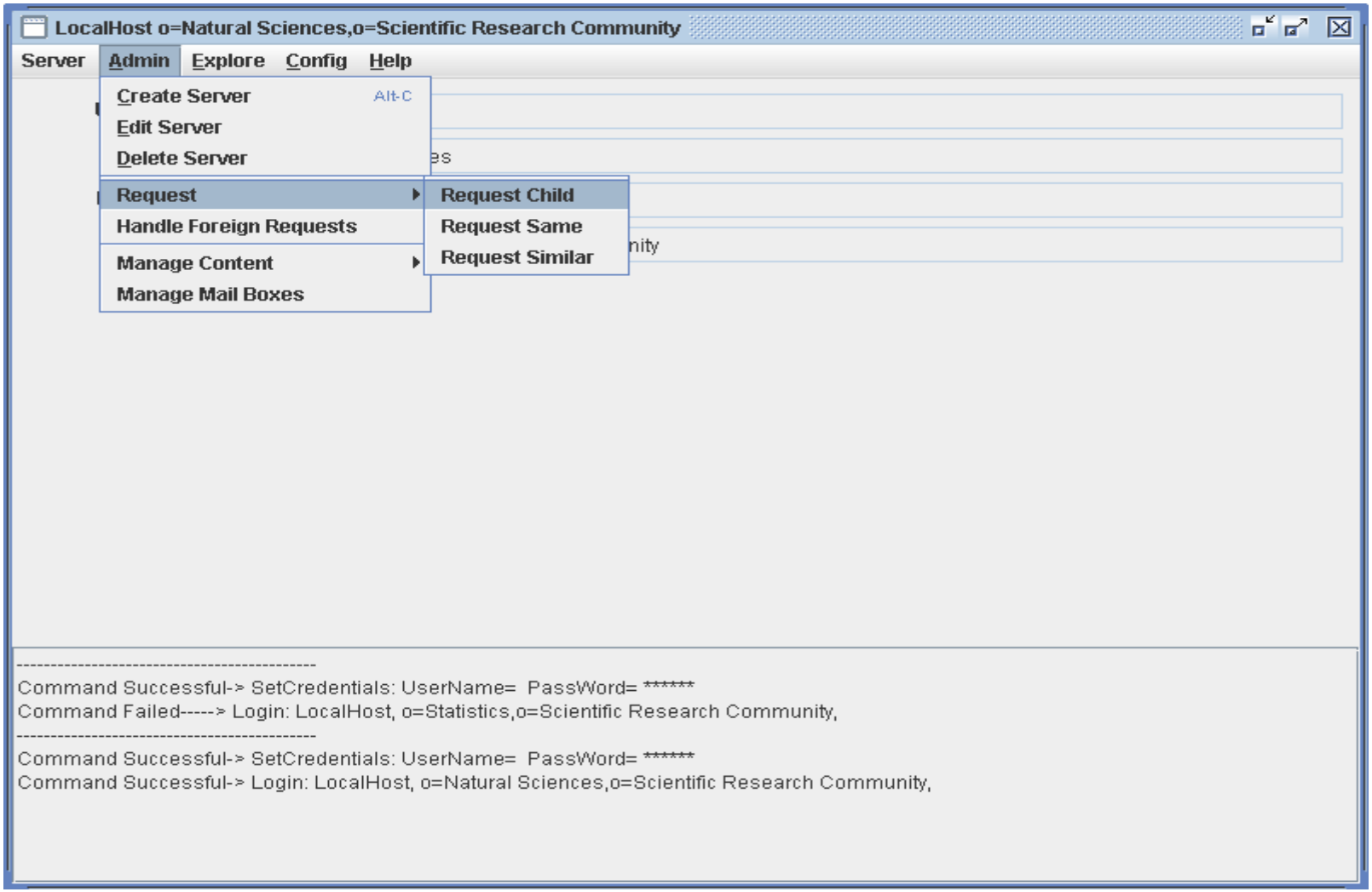
- Data Grids mentioned in this model can be of any implementation. The model keeps record for them and for the manager server of each of them. The user of this model will need to follow the rules of each data grid as imposed via its manager server. The model will not keep record for membership of the Data Grids. If a server needs to know any info about the Data Grid, it should call GetDataGridManager (DGID) then contact the server identified by the returned global ID (e.g. IP of the manager server of the data grid).

Implementation : GUI



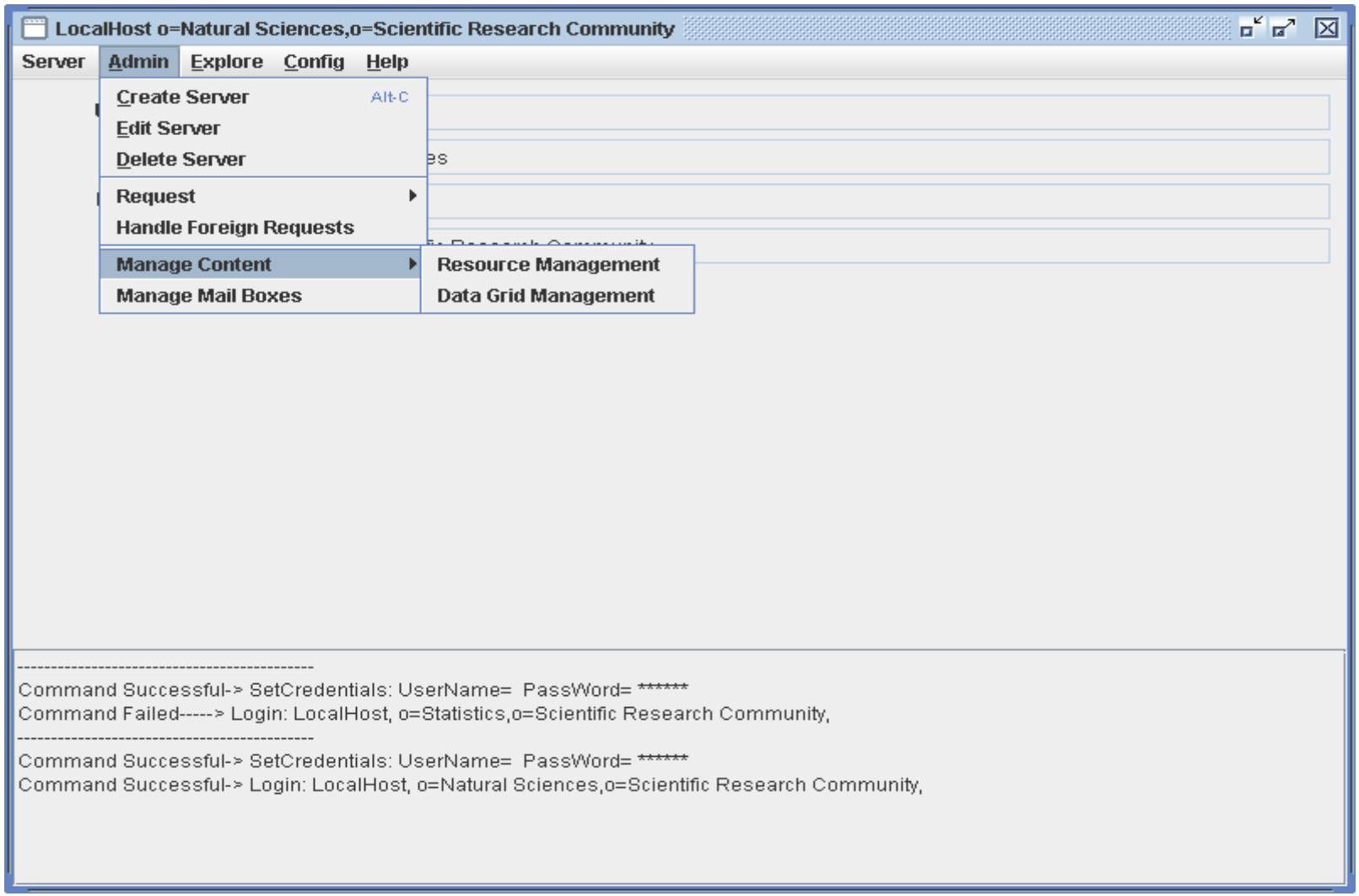
Main Menu: Login Server

Implementation : GUI



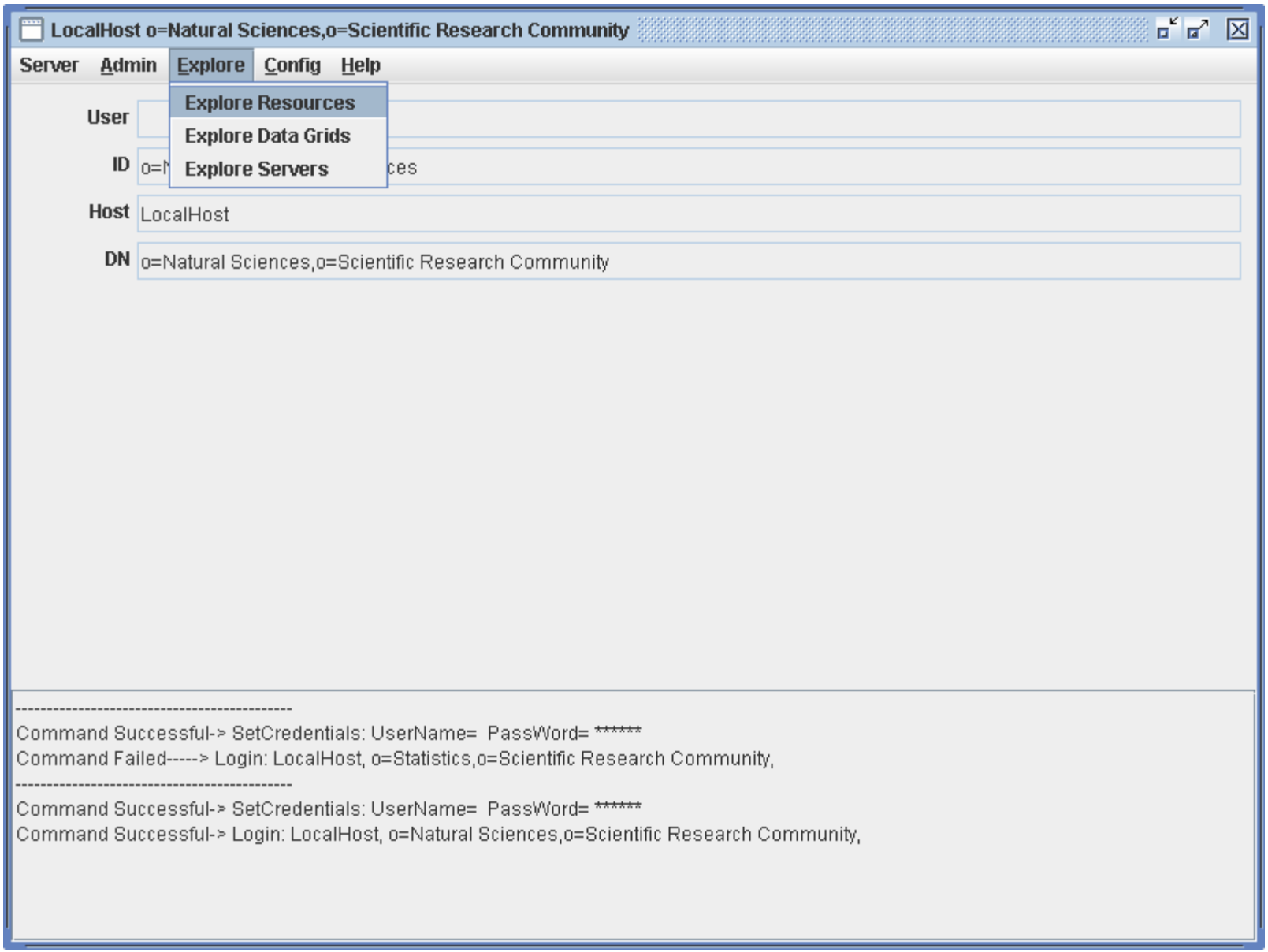
Admin Menu: Request Child/Same/Similar

Implementation : GUI



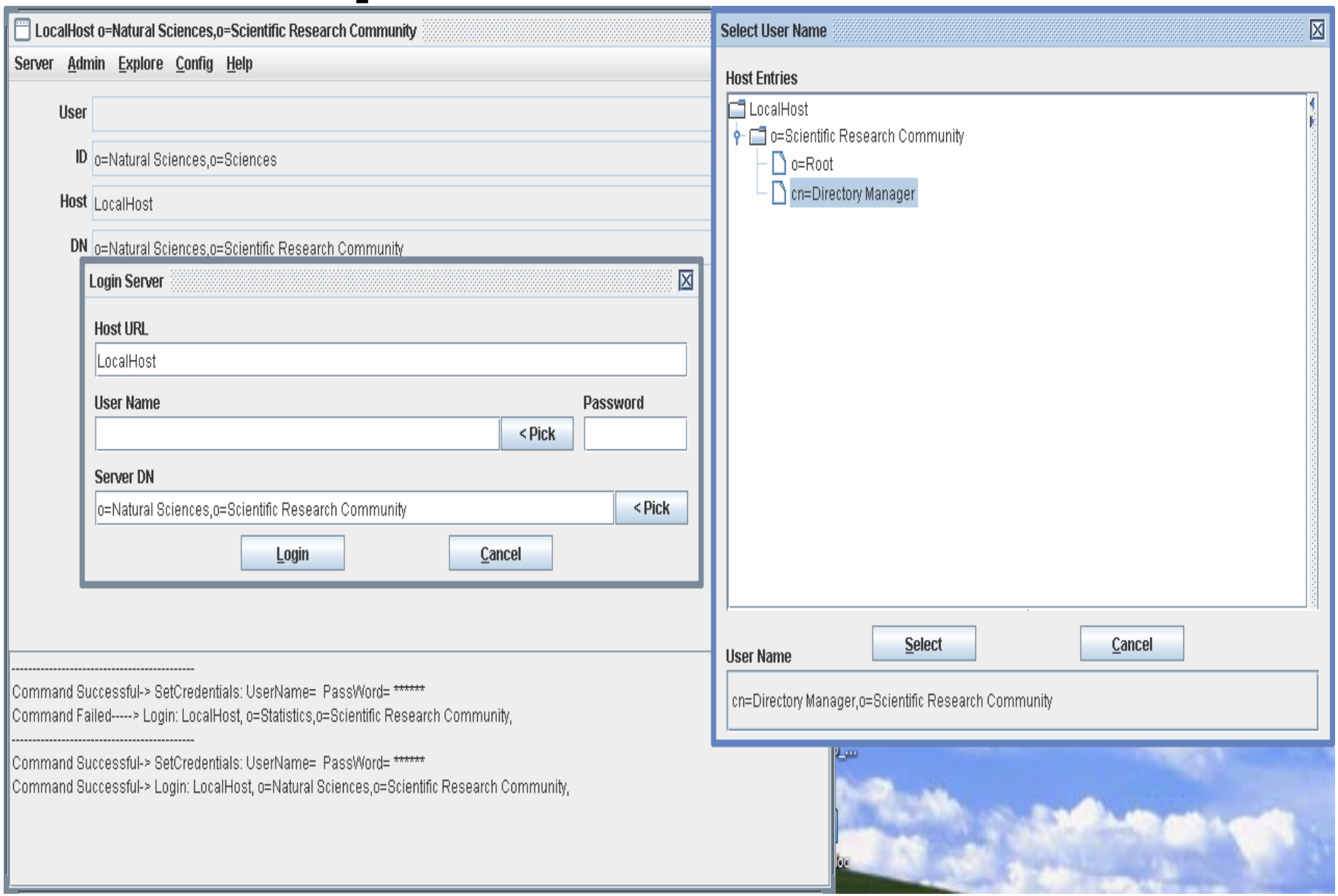
Admin Menu: Content Management

Implementation : GUI



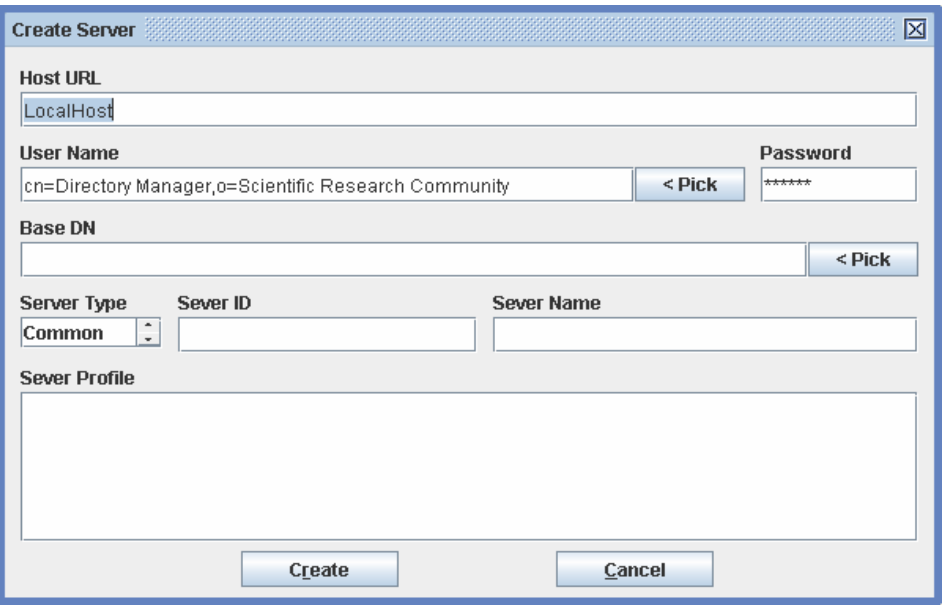
Explore Menu: Resources/Data Grids/Servers

Implementation : GUI

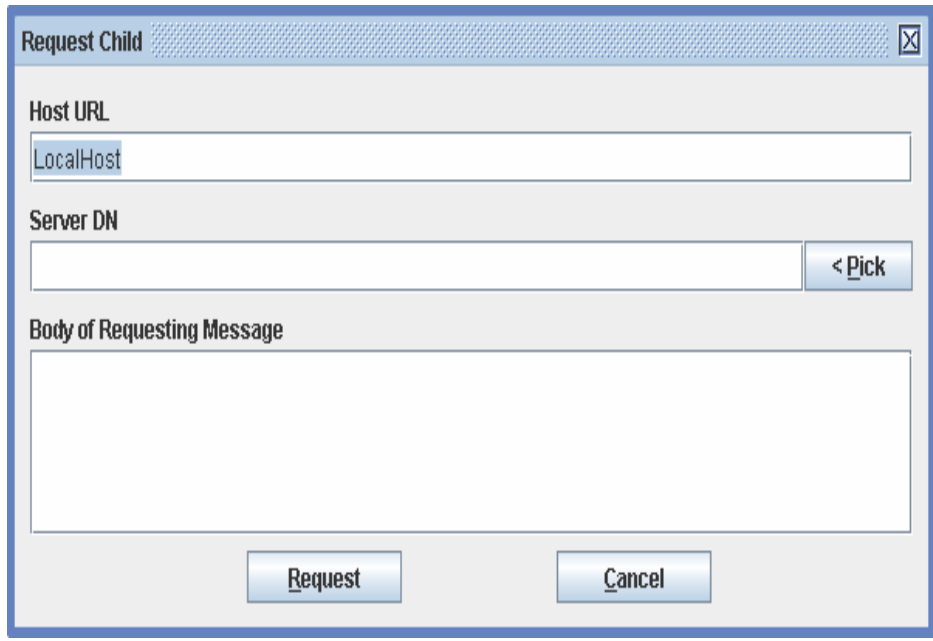


Login Server Form

Implementation : GUI

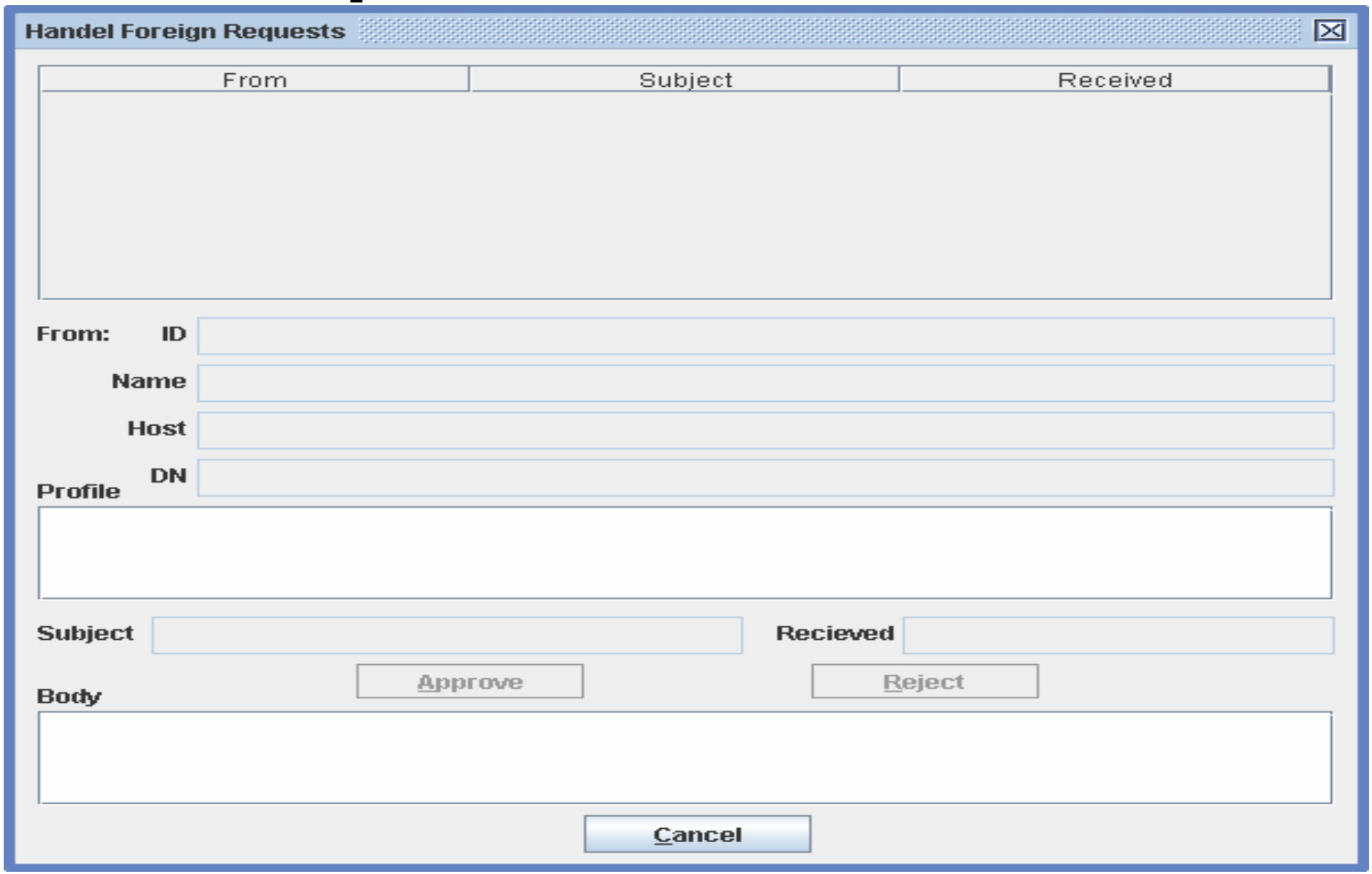


Create Server Form



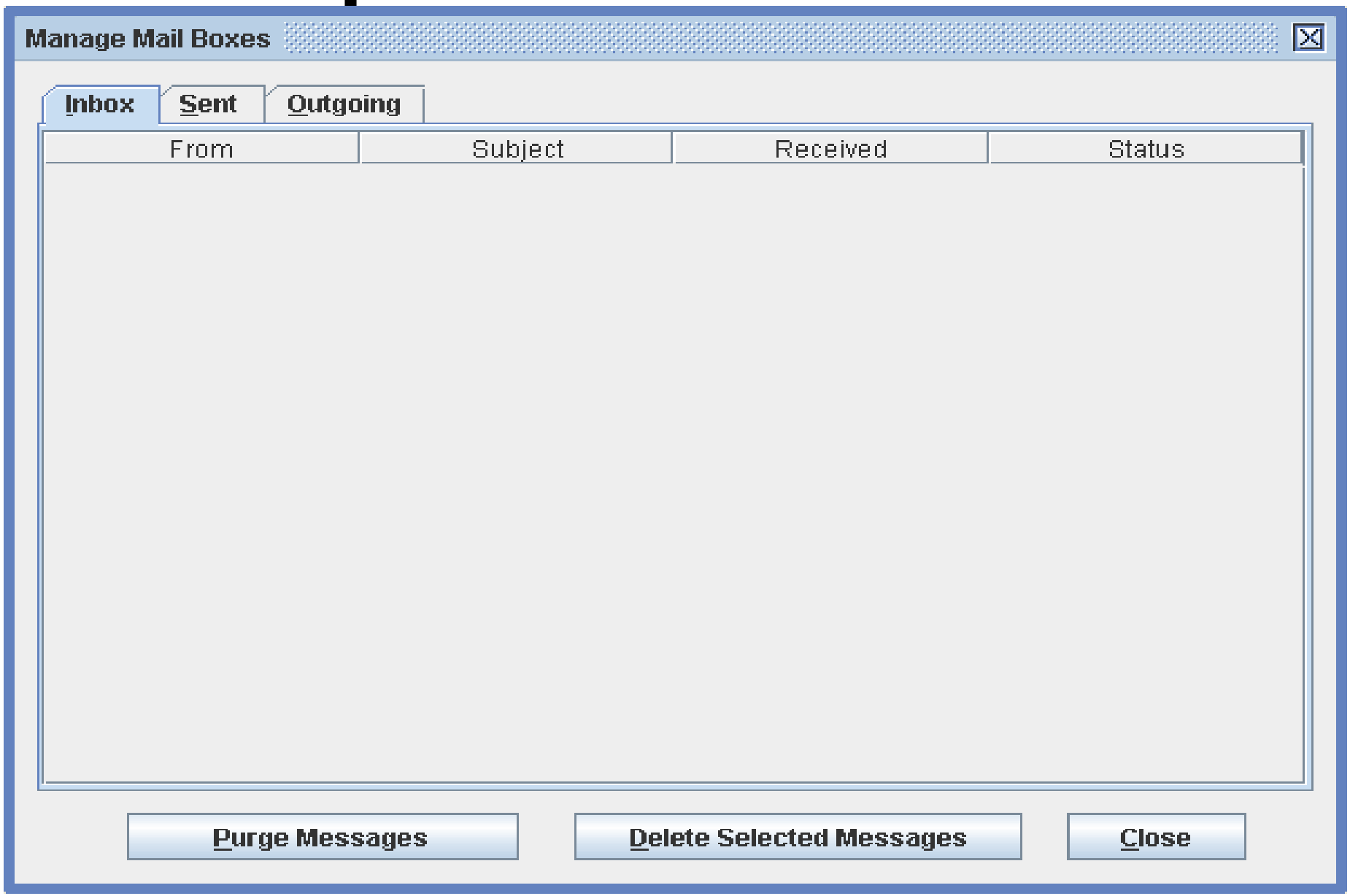
Request Child Form

Implementation : GUI



Handle Foreign Requests Form

Implementation : GUI



Manage Mail Boxes Form

Testing Strategy

- **Test Data**
  Fields of Science

- **Experimental Setup**
  Five machines with Windows XP/Java/LDAP/Our System

- **Functional Testing:**
  Each Individual Function was tested separately

- **Deployment Testing:**
  Overall system testing to test longer scenarios like Deepening, etc.

- **Simulation Testing:**
  Many servers and events were simulated to test the overall system functionality

Test Data : Scientific Research Taxonomy

- *"Common" Servers*
- *Natural Sciences*
  - *Mathematics*
    - *Pure Mathematics*
      - *Algebra*
      - *Geometry*
      - *…*
    - *Applied Mathematics*
      - *Mathematical Physics*
      - *Mechanics*
      - *…*
  - *Physics*
    - *Acoustics*
  - *…*
- *Humanities*
  - *Anthropology*
    - *Archaeology*
  - *…*

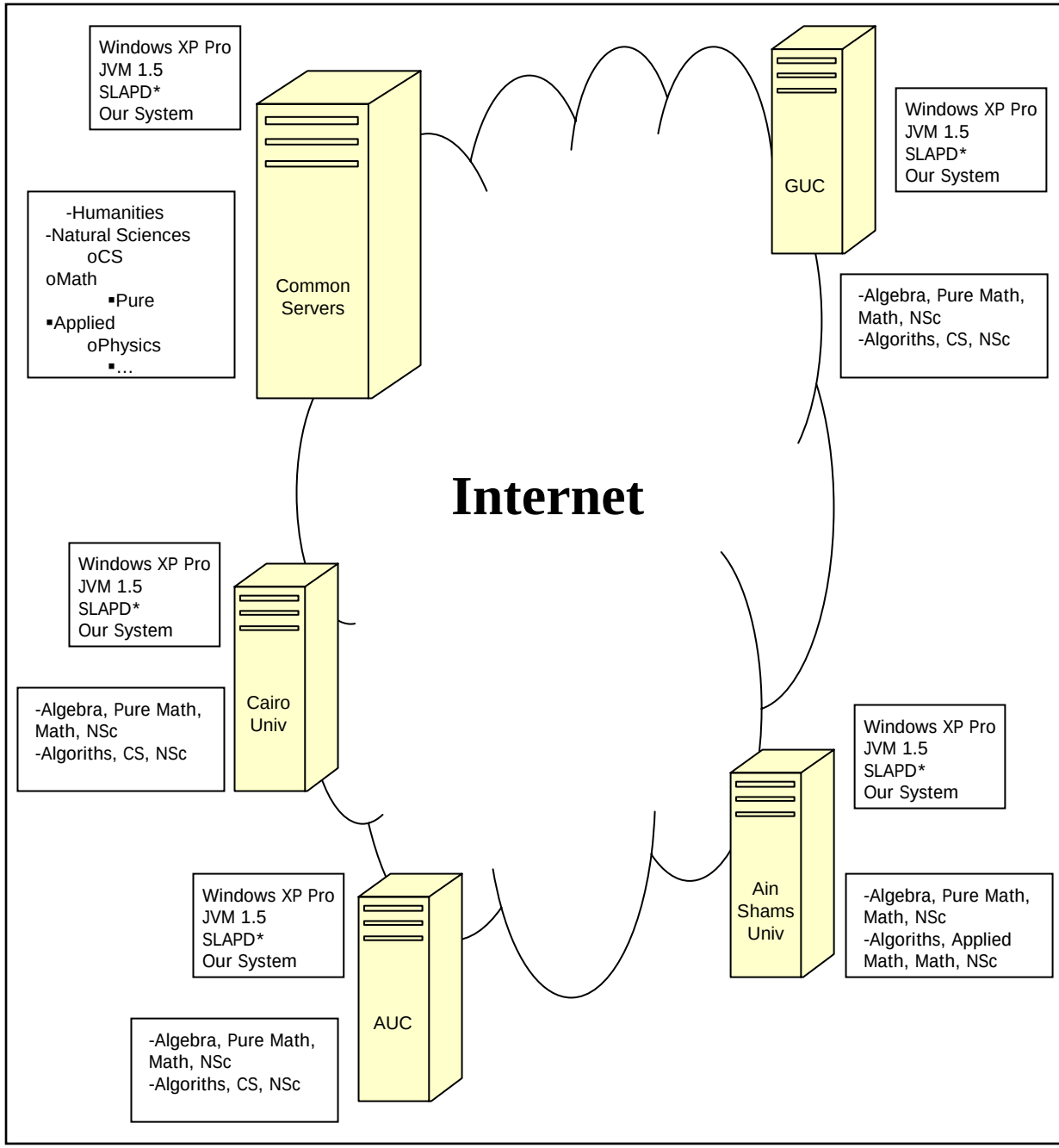
"Institutional" Servers

1. *Cairo University*
   a. ***"cn=Noise Reduction Coefficient,o=Noise control,o=acoustics,o=Physics, o=Natural Sciences,o= Sciences"***

2. *AUC*
   a. "**cn=NRC**, o=Noise control,o=acoustics,o=Physics, o=Natural Sciences,o= Sciences" (same as 1.a).

3. *Ain Shams university*
   a. "**cn=Co-Articulation**,o=Phonetic Segmentation,o=Speech Segmentation,o=Speech Recognition,o=Speech Processing,o= Audio signal processing,o=Acoustics,o=Physics,o=Natural Sciences,o= Sciences".

Experimental Setup



*SLAPD is a directory server that adopts LDAP protocol

Deployment Test

- Five machines (running Windows XP Pro) were setup and named: Common, AUC, GUC, Cairo, and Ain Shams to represent institutional servers. On each machine, our system plus JVM 1.5, a SLAPD (a directory server that adopts LDAP protocol), were installed. Test data as the hierarchy shown in section 7.3 above were fed to each machine (each machine held servers of one university). The common machine had the common servers, that's to say the common part of the taxonomy, and the other 4 machines representing the 4 universities had the institutional servers. Then, different transactions and scenarios (e.g. create new server, request relation, add/query resource, explore…etc.) were fed to the system. A number of hypothetical data grids were also registered at some servers. The Figure below shows the experimental setup.

- An overall system testing was done to test the integrated end-to-end system functionality. Major functions like creating servers, requesting relation, deepening, registering DG, discovery of DGs, servers and resources, server leave, handling less relevant data were tested and the results were recorded.

Simulation Test

- Test data was prepared for scientific research taxonomy and fed to the system through simulation interface (all servers are created on one physical LDAP server). Each created server represents a "common" server acting for a science branch. Screen shots, then, were taken for the GUI module to show the hierarchy of servers as built. "Institutional" servers were suggested and test data was created and appended to the "common" server data. Each group of institutional servers represents a university or a research center. Each "institutional" server serves a research topic in the institution and is hooked to the system through a "child" relation with a "common" server. Deeper branching exists through "institutional" servers where no "common" servers at that depth exist.

- Server names follow the LDAP entry name convention (Name=Value). The used LDAP server (AE SLAPD) does not support adding new names, so, available names are used. "o=" is used for "Common" servers and "cn=" for "Institutional" servers.

Functional Testing 1 – Creating The Hierarchy

Creating Servers

Creates common and institutional servers.

- **Create Sciences Common Sciences LocalHost %BASEDN% "Science Taxonomy“**
- **Create "Natural Sciences" Common "Natural Sciences" LocalHost %BASEDN% "Natural Sciences“**
- **Create Physics Common Physics LocalHost %BASEDN% Physics**
- **Create Humanities Common Humanities LocalHost %BASEDN% Humanities**
- **Create History Common History LocalHost %BASEDN% History**
- **Create "Social Sciences" Institutional "Social Sciences" LocalHost %BASEDN% "Social Sciences"**

Functional Testing 2 – Creating The Hierarchy

Requesting Relations

Requests *Child*, *Same* and *Similar* relations

- **Login LocalHost o=Humanities,%BASEDN%**
- ***RequestChild* LocalHost o=Sciences,%BASEDN% "Hi There, accept me as a child please!"**

- **Login LocalHost "o=Natural Sciences,%BASEDN%"**
- ***RequestChild* LocalHost o=Sciences,%BASEDN% "Hi There, accept me as a child please!"**

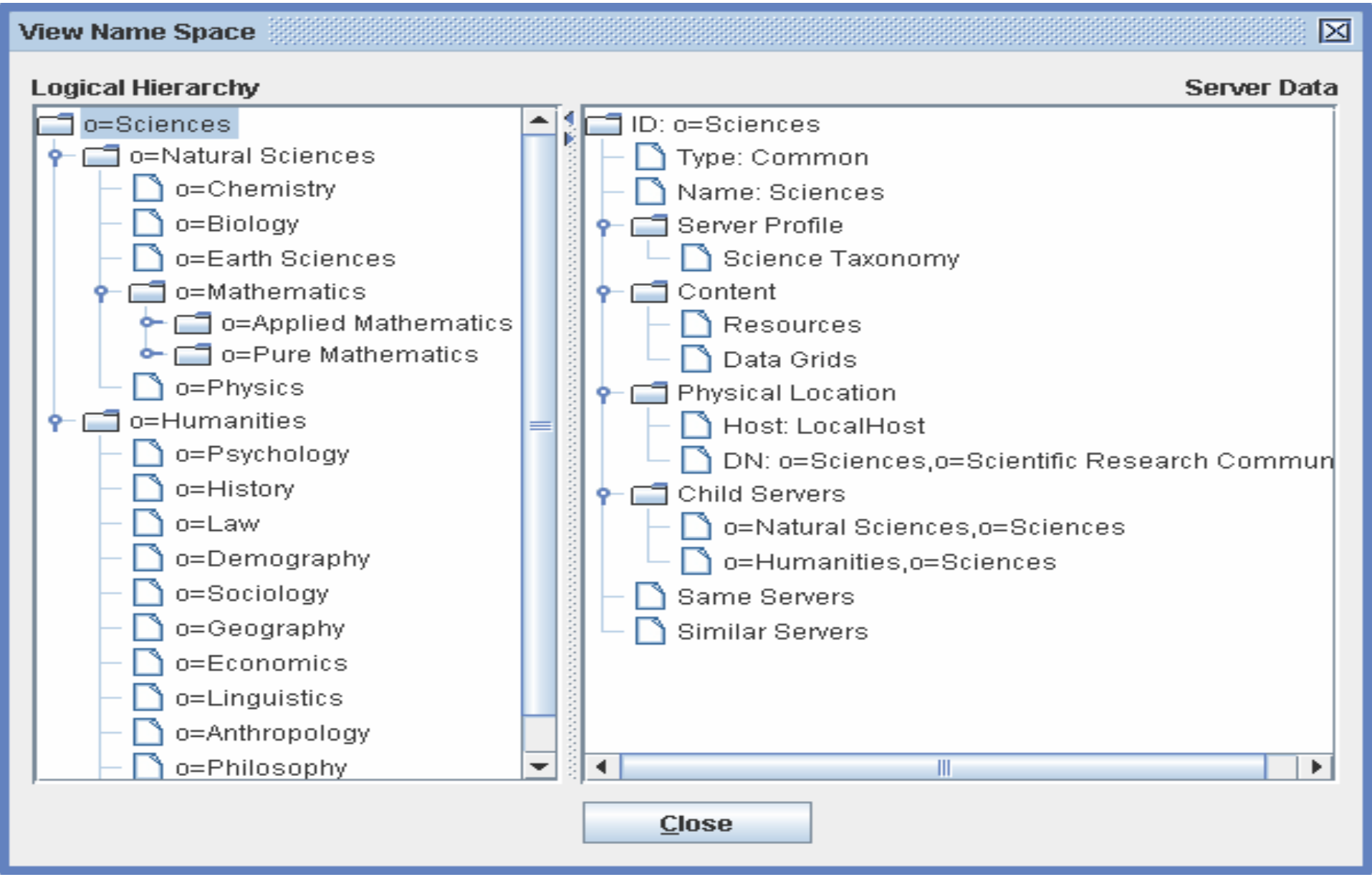
- **Login LocalHost "o=Social Sciences,%BASEDN%"**
- ***RequestChild* LocalHost o=Sciences,%BASEDN% "Hi There, accept me as a child please!"**
- ***RequestSame* LocalHost o=Humanities,o=Sciences "Hi There, accept me as Same please!"**

- **Login LocalHost o=History,%BASEDN%**
- ***RequestChild* LocalHost o=Humanities,%BASEDN% "Hi There, accept me as a child please!"**

- **Login LocalHost o=Physics,%BASEDN%**
- ***RequestChild* LocalHost "o=Natural Sciences,%BASEDN%" "Hi There, accept me as a child please!"**

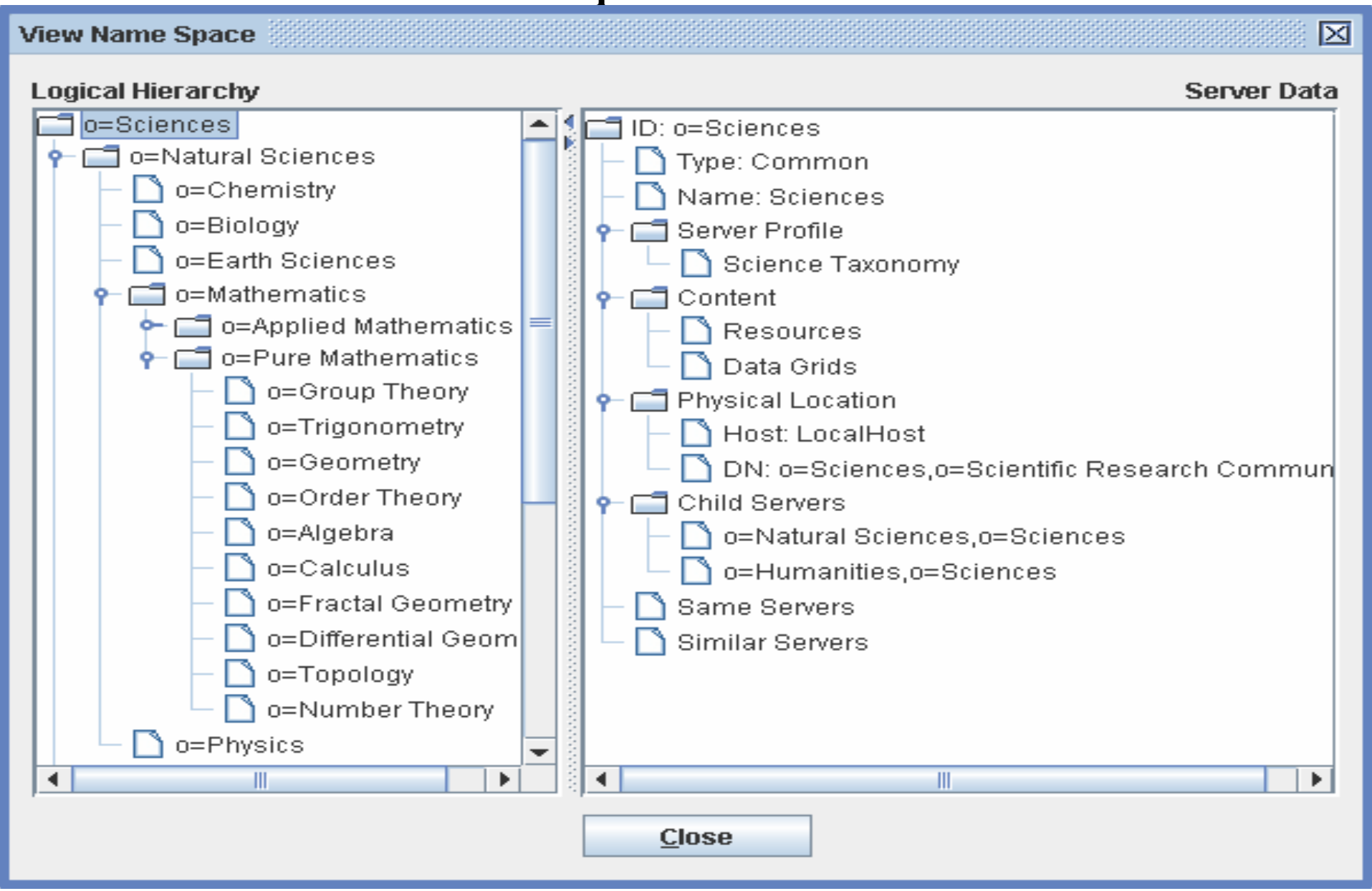
- **Login LocalHost o=dynamics,o=Physics,%BASEDN%**
- ***RequestSimilar* LocalHost "o=dynamics,o=Mathematics,o=Natural Sciences" "Hi There, accept me as Similar please!"**

Functional Test Output 1 –The Hierarchy Created



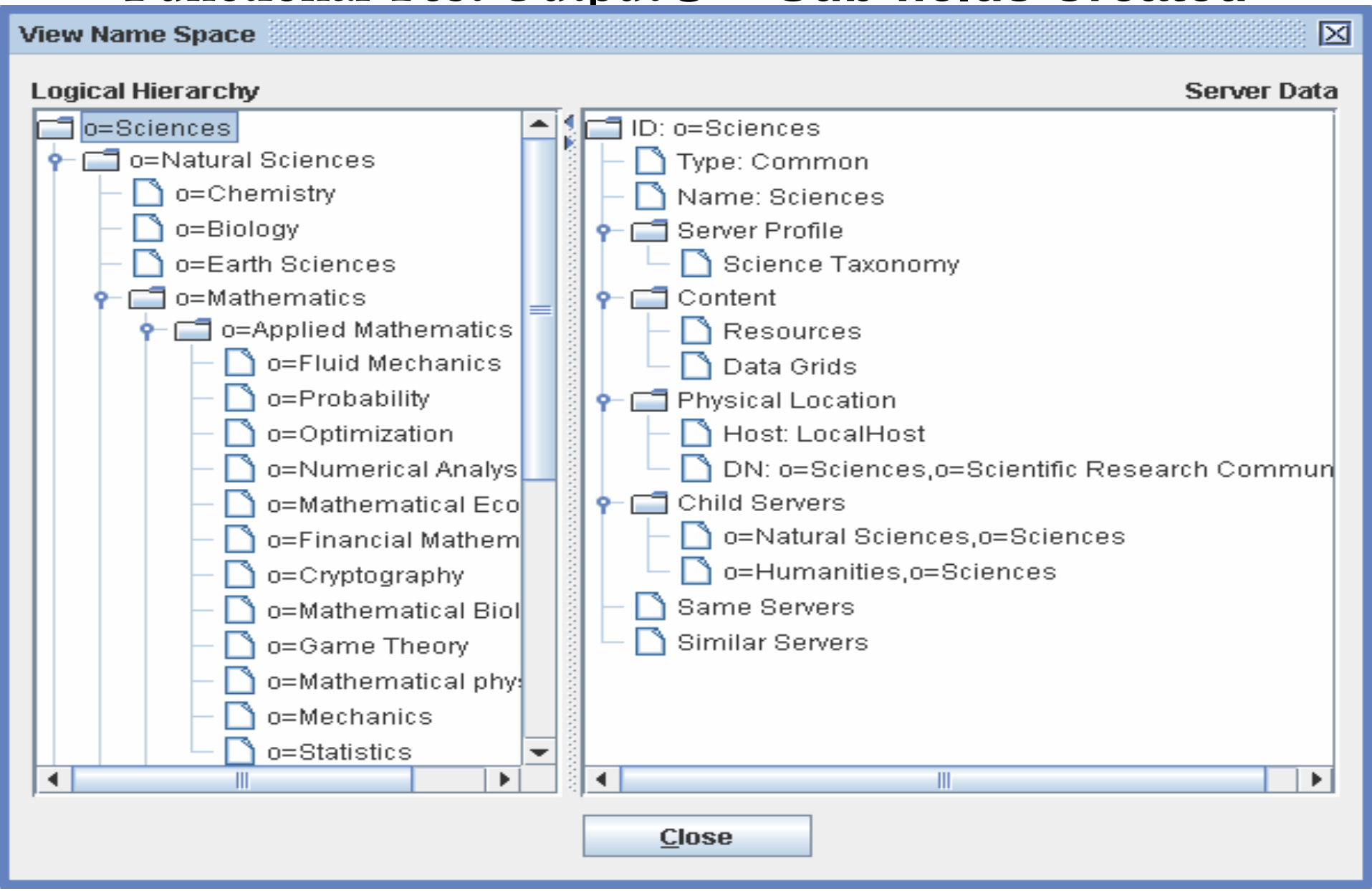
Servers Representing Main Science Fields.

Functional Test Output 2 –Sub-Fields Created



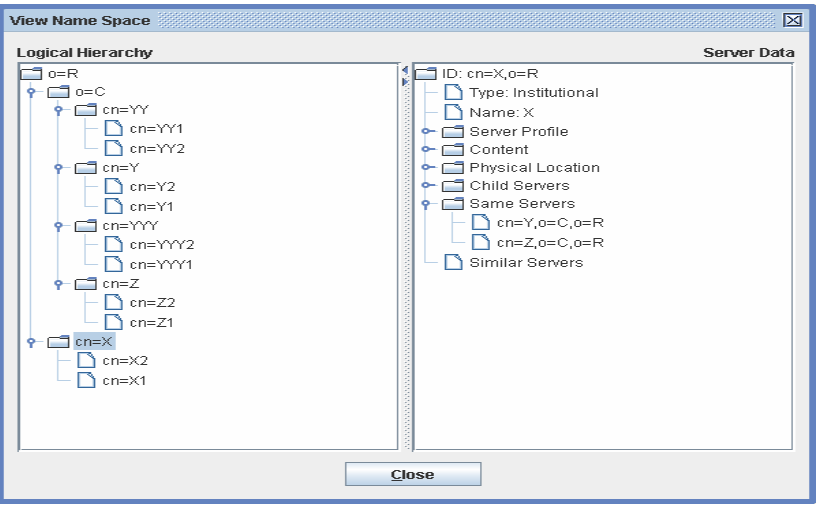
Pure Mathematics Sub-fields

Functional Test Output 3 – Sub-fields Created

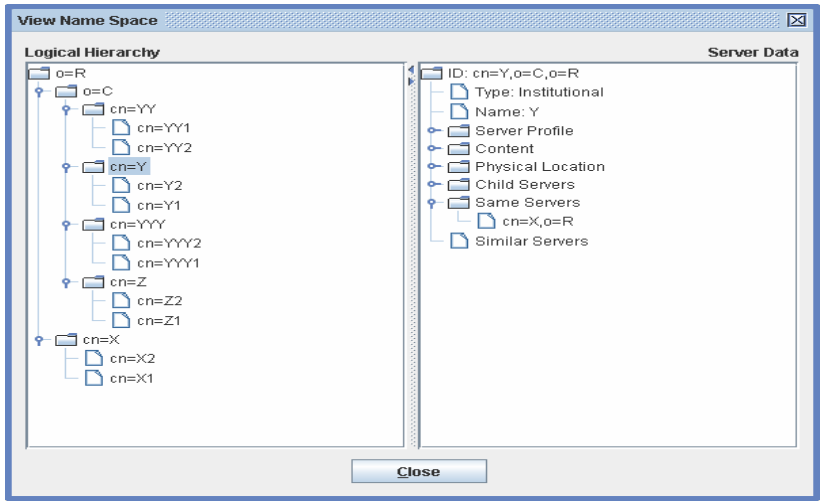


Applied Mathematics Sub-fields

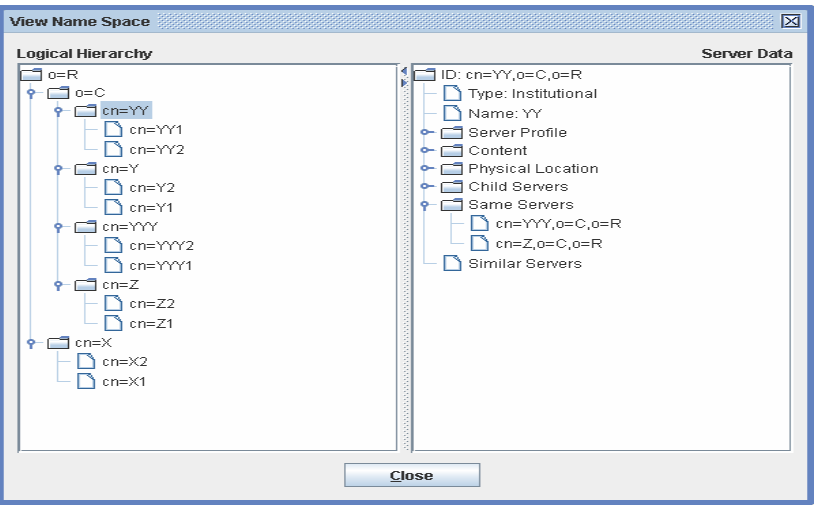
Deployment Test –*Deepening*



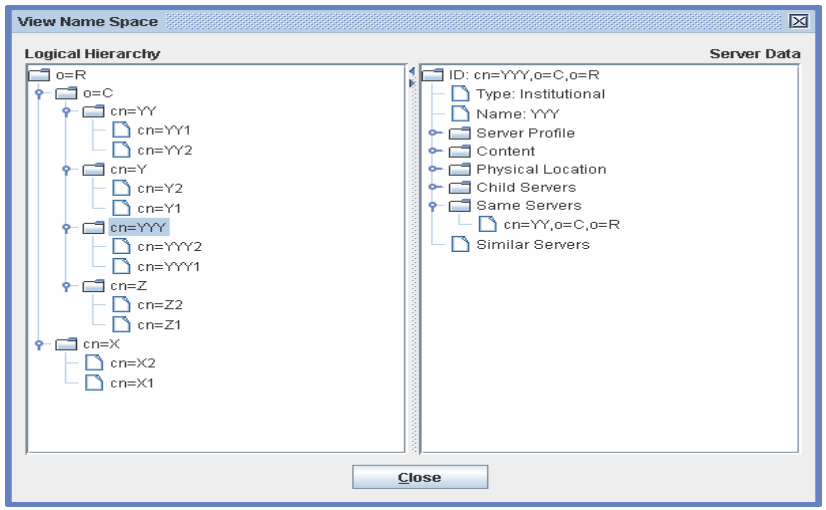
Servers Y and Z are Same to Server X



Server X is Same to Server Y



Servers YYY and Z are similar to Server YY

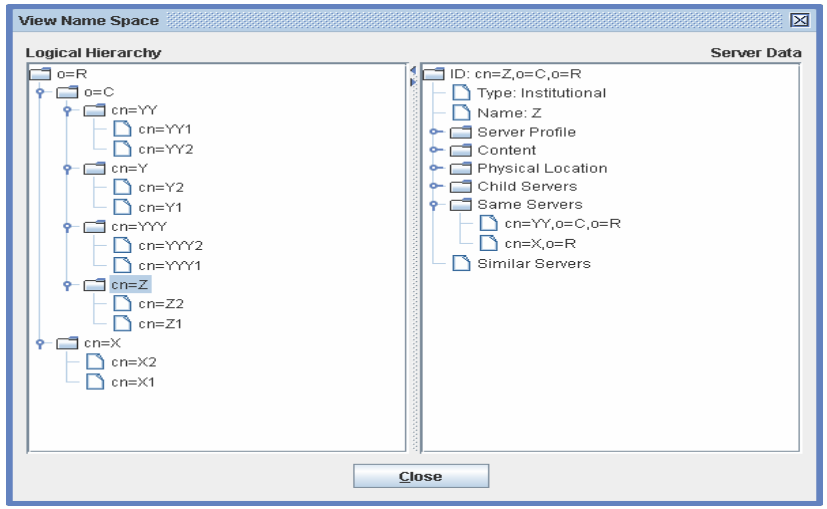


Server YY is similar to Server YYY

Shows the hierarchy before the deepening process

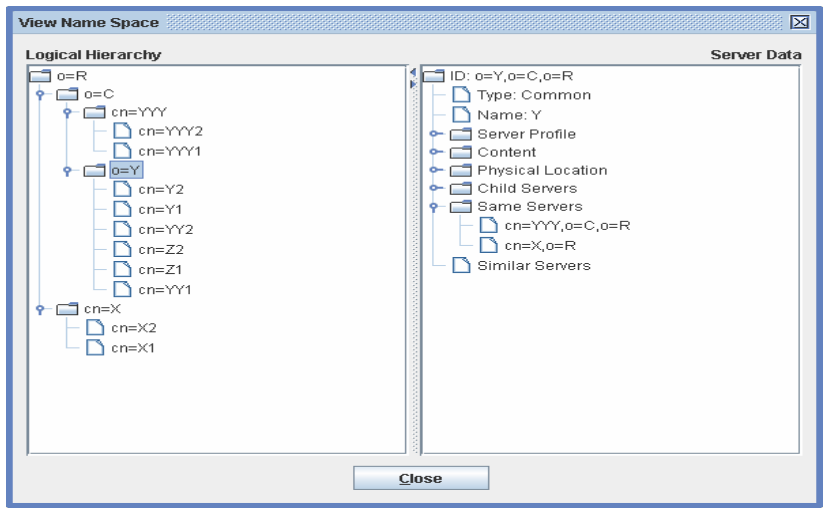
Deployment Test – *Deepening (2)*

Shows the hierarchy before the deepening process (Ctd.)



Servers YY and X are similar to Server Z

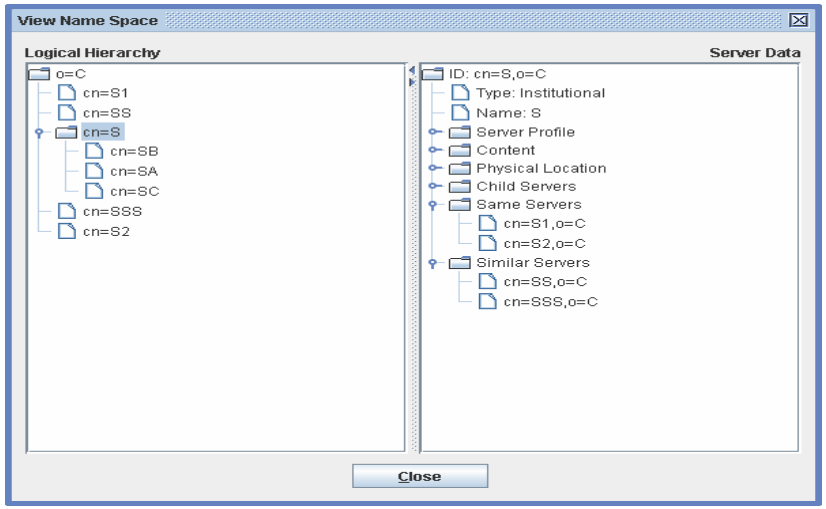
Shows the hierarchy after deepening process – deepening of common part



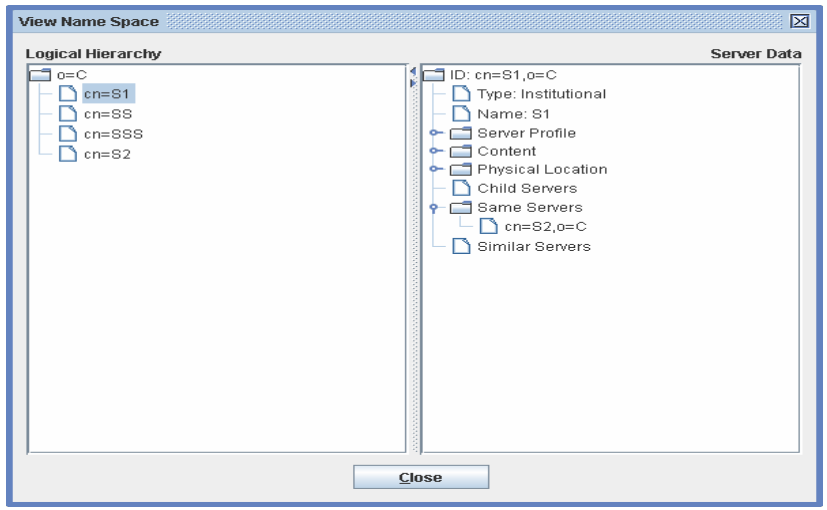
New Common Server Y replacing same servers YY and Z and taking their children for itself

Deployment Test–*Before Leaving*

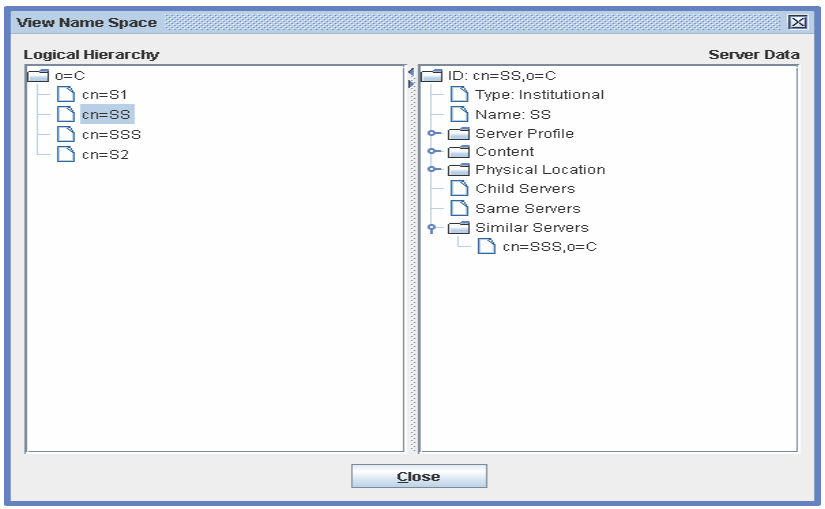
Hierarchy, Same & Similar relations before leaving



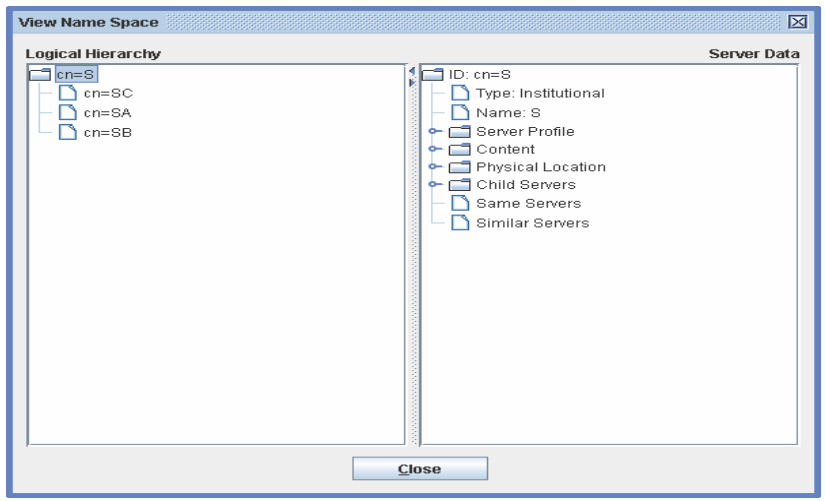
Server ‘S’ wants to leave the model



After Server ‘S’ has left the Model: ‘S1’ is *Same* to ‘S2’ only



After “S” has left the Model : Only ‘SSS’ is similar to ‘SS’



After ‘S’ has left : it left with its *children* but has lost its *Same* and *Similar* relations to the model servers

Simulation Test – Sample Output

```
// Setting environment's variables
// ------------------------------
SetLog on
SetVariable %BASEDN% "o=QueryExplore,o=Scientific Research
    Community"
SetCredentials "cn=Directory Manager,o=Scientific Research
    Community" secret
//
// Creating root entry
// ------------------------------
DeleteRoot LocalHost %BASEDN%
CreateRoot LocalHost %BASEDN%
//
// Creating servers
// ------------------------------
Create S0 Common S0 LocalHost %BASEDN% "N"
Create S1 Institutional S1 LocalHost %BASEDN% "X"
Create S2 Institutional S2 LocalHost %BASEDN% "X Y"
Create S3 Institutional S3 LocalHost %BASEDN% "X Y Z"
Create S4 Institutional S4 LocalHost %BASEDN% "X Y Z P"
Create S5 Institutional S5 LocalHost %BASEDN% "X Y Z P Q"
Create S6 Institutional S6 LocalHost %BASEDN% "X Y Z P Q R"
```

Simulation Test – Sample Output

```
// Building Hierarchy
// ------------------------------
Login LocalHost cn=S1,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

Login LocalHost cn=S2,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

Login LocalHost cn=S3,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

Login LocalHost cn=S4,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

Login LocalHost cn=S5,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

Login LocalHost cn=S6,%BASEDN%
RequestChild LocalHost o=S0,%BASEDN% "Hi There, accept me as a child please!"

//
// Exploring servers
// ------------------------------
ExploreServers X
        [cn=S1,o=S0]
        [cn=S2,o=S0]
        [cn=S3,o=S0]
ExploreServers Y
        [cn=S2,o=S0]
        [cn=S3,o=S0]
ExploreServers Z
        [cn=S3,o=S0]
```

Simulation Test – Sample Output

```
// Adding resources
// ------------------------------
Login LocalHost cn=S1,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF

Login LocalHost cn=S2,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF

Login LocalHost cn=S3,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF

Login LocalHost cn=S4,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF

Login LocalHost cn=S5,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF

Login LocalHost cn=S6,%BASEDN%
AddResource XXX 24000 "11/10/96 12:20 pm" "11/10/96 12:20 pm" "J. D.
        McDonald" PDF ftp://aucegypt.edu/docs/XXX.PDF
```

Simulation Test – Sample Output

```
// Querying resources
// ------------------------------
Login LocalHost cn=S1, %BASEDN%
QueryResource XXX
      [Local  cn=S1,o=S0]
//
// Discovering resources through Same relation
// ------------------------------
RequestSame cn=S2,o=S0 "Hi There, accept me as Same please!"
QueryResource XXX
      [Local  cn=S1,o=S0]
      [Same  cn=S2,o=S0]
//
// Discovering resources through Similar relations
// ------------------------------
RequestSimilar cn=S3,o=S0 "Hi There, accept me as Similar please!"
QueryResource XXX
      [Local  cn=S1,o=S0]
      [Same  cn=S2,o=S0]
      [Similar(order:1)  cn=S3,o=S0]
Login LocalHost cn=S3, %BASEDN%
RequestSimilar "cn=S4,o=S0 "Hi There, accept me as Similar please!"
Login LocalHost cn=S1, %BASEDN%
QueryResource XXX
      [Local  cn=S1,o=S0]
      [Same  cn=S2,o=S0]
      [Similar(order:1)  cn=S3,o=S0]
      [Similar(order:2)  cn=S4,o=S0]
```

Simulation Test – Sample Output

```
// Side Kick: Similar relation from Same relation
// ------------------------------
Login LocalHost cn=S2, %BASEDN%
RequestSimilar cn=S5,o=S0 "Hi There, accept me as Similar please!"

Login LocalHost cn=S1, %BASEDN%
QueryResource XXX
     [Local  cn=S1,o=S0]
     [Same  cn=S2,o=S0]
              [Side Kick:  cn=S5,o=S0]
     [Similar(order:1)  cn=S3,o=S0]
     [Similar(order:2)  cn=S4,o=S0]
//
// Side Kick: Same relation from Similar relation
// ------------------------------
Login LocalHost cn=S4, %BASEDN%
RequestSame cn=S6,o=S0 "Hi There, accept me as Same please!"

Login LocalHost cn=S1, %BASEDN%
QueryResource: XXX
     [Local  cn=S1,o=S0]
              [Side Kick:  cn=S5,o=S0]
     [Same  cn=S2,o=S0]
     [Similar(order:1)  cn=S3,o=S0]
              [Side Kick:  cn=S6,o=S0]
     [Similar(order:2)  cn=S4,o=S0]
```

Simulation Test – Sample Output

```
//
// Adding data grids
// ------------------------------
Login LocalHost cn=S1, %BASEDN%
//                dgID    ServerURL      Profile        Proxy Agent URL
AddDataGrid DG1 cs.aucegypt.edu   "X"   ftp://cs.aucegypt.edu/DG/proxyAgent1.class

Login LocalHost cn=S2, %BASEDN%
AddDataGrid DG1 cs.aucegypt.edu "X Y" ftp://cs.aucegypt.edu/DG/proxyAgent2.class

Login LocalHost cn=S3, %BASEDN%
AddDataGrid DG1 cs.aucegypt.edu "X Y Z" ftp://cs.aucegypt.edu/DG/proxyAgent3.class
//
// Querying data grid
// ------------------------------
Login LocalHost cn=S1, %BASEDN%
QueryDataGrid DG1
        [Server URL: cs.aucegypt.edu]
        [Profile: X]
        [Proxy Agent URL: ftp://cs.aucegypt.edu/datagrid/proxyAgent1.class]
//
// Exploring data grids
// ------------------------------
ExploreDataGrids: X
        [cn=S1,o=S0 /// DG1]
        [cn=S2,o=S0 /// DG2]
        [cn=S3,o=S0 /// DG3]
ExploreDataGrids: Y
        [cn=S2,o=S0 /// DG2]
        [cn=S3,o=S0 /// DG3]
ExploreDataGrids: Z
        [cn=S3,o=S0 /// DG3]
```

Conclusion

- **The model is proven to *work* with almost no cost on contributors**
- **A by-product of the design goals (*Decentralization*) is the perfect performance measures. Since a server knows nothing except its content and the coordinates of its neighbours (parent, children, "Same" and "Similar" severs), the operations are done faster.**
- ***Semantics* which is expressed in as the field "Ontology" describing the server's mission and the resource content and interests allow better resource discovery by automating it, hence results in easier and faster resource discovery, which in turns results in higher *interoperability***
- **The heavy search operations are done by many servers through propagation patterns that follow child/parent relations and/or "Same"/"Similar" relations. This propagation involves more servers in the search operations exponentially.**

  For example, at time ***t1*** neighbours of the first server will be involved (a matter of 10 servers). At time ***t2***, neighbours of those neighbours will be involved (a matter of 10 x 10 servers) and so on. This exponential invocation reduces the *search time* to logarithmic cost (***log n***) instead of linear cost (***n***, where n is number of servers).

  This guarantees no bottlenecks and fast searches

Future Horizons

XML could be used to define the relations in terms of a set of standard attributes. Hence standardizing an extensible framework for inter-server relations that allows new kinds of relations to be added in the future

Adding new relations by third parties will create new roles for the model as per other models' needs

Moving the complete, extensible and open messaging unit in our current design to be a separate middleware layer or subsystem. Makes it easier for third parties to join and collaborate using our model.

Designing a mobile agent that moves around the model to collect information that would facilitate the future search of a specific server. This agent can be augmented by some provisions about the future needs of its owner server that guides its tours towards more fruitful navigations. Provisions can be stated directly by server owners (human intelligence) or by heuristics (artificial intelligence).

Caching is another possibility for performance enhancement in the future