Khulna University Studies 4(1): 603-610

ONTOLOGY BASED DIRECTORY ENABLED NETWORK DESIGN USING JAVA NAMING AND DIRECTORY INTERFACE

K. S. Islam*, C. K. Roy and M. A. Rahman

Computer Science and Engineering Discipline, Khulna University

KUS-01/52-251201

Manuscript received: December 25, 2001; Accepted: May 12, 2002

Abstract: Directory Enabled Network (DEN) provides us Information Model to design an interoperable, distributed and robust service for accessing network applications, services and resources in a platform independent manner. This paper provides a detail framework for the DEN implementation from ontological perspectives where DEN has been used to help easy management of a network instead of managing individual resources and services. Java Naming and Directory Interface (JNDI) are recommended as programming tools in Java platform for implementing DEN ontology.

Keywords: Ontology, DEN, CIM, JNDI and Context.

Introduction

Technology keeps evolving, and networks have become more and more complex in an attempt to provide better service and to meet the needs of an increased number of users and more sophisticated applications. Today's Network contains a large number of network resources and services and management of those resources and services are not so easy. Network itself has not actually been managed; rather individual devices of networks have been managed.

Directory enabled network has defined the way to manage the network instead of an element within a network. DEN provides an interoperable information model for exchanging management, as well as operational and functional information describing the network and the systems the network communicate with, using a pre-defined ontology for Directory Enabled Network enables wide range of uniformity through the interoperability of network resources and services.

Ontology enables a system to integrate multiple systems and databases, helping people understand and use knowledge more effectively, and helping people reach consensus on the meaning of the domain knowledge. The focus of this study is to pay attention towards the integration of existing and future network resources and services to build Directory Enabled Network based on Java Naming and Directory Interface (JNDI).

Directory Enabled Network Architecture

Directory services are generally implemented as a client/server system. A Directory client connects to a Directory server and either queries for information or provides information that needs to be entered into the directory. The server either answers the query, or refers the query to another Directory server, or accepts the information for incorporation into the directory. A standard protocol should be chosen to provide access to the Directory, which is specifically for simple management and browser applications that provide simple read/write interactive access to the Directory.

DEN Naming and Directory Service play as an ontology server, which could be used to support ontology for the locating network resources and services. Each of the service providers will register their name to the Ontology server at the start up time. Ontology server evaluates and responses the related query within the scope of the defined ontology about the available resources and services make by client agents. Client and server agents engaged for compatible service can communicate through consultation with ontology server and they can communicate to each other using any compatible protocols in order to meet user needs (Finin *et al.*, 1997). Domain Name System (DNS), Network Information System, Netware Directory Service (NDS), Microsoft Active Directory (AD) etc. are some existing directory services, but none of them are DEN compliant.

The following figure illustrates the way of communication among DEN naming and Directory service, clients, and servers:

*Corresponding author: Tel.: 880-41-721791, 720171-3 Ext. 213; Fax: 880-41-731244; e-mail: regekuly@bttb.net.bd

DOI: https://doi.org/10.53808/KUS.2002.4.1.0152-E

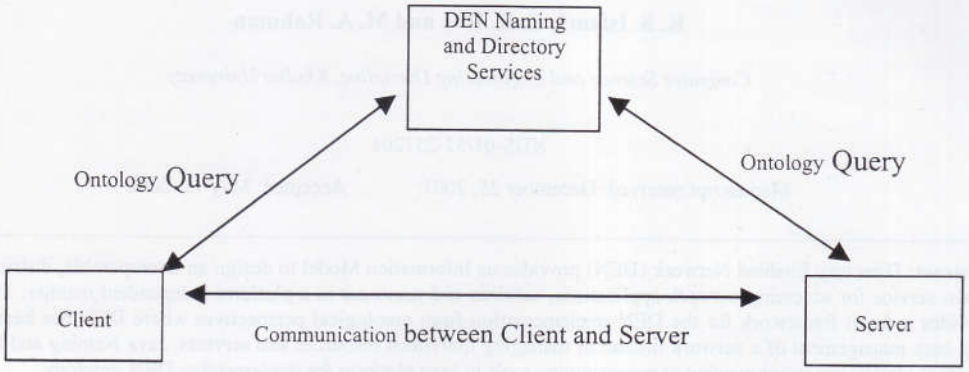

Fig.- 1. Communication among Clients and Servers using Directory Service

Common Information Model

Common Information Model (CIM) is an object-oriented information model that is designed to manage many common aspects of complex computer systems and their components defined by Distributed Management Task Force (DMTF). A primary goal of CIM is to present a consistent view of the managed environment that is independent of the various protocols, and data formats used by those devices and applications. CIM has gained acceptance as an information model for enterprise management tools by many network infrastructure and management software providers. CIM classes are used by DEN and serve as base classes for DEN.

CIM and applications written to use CIM are natural sources of information for the directory. The directory is a natural source of information for CIM and applications written to use CIM, and adds models for defining and enforcing policy. Network applications integrated with the directory benefit from CIM, and CIM applications benefit from network applications integrated with the directory, with minimal effort on the application developer's part because the directory-enabled network schema is an extension of the CIM schema. Thus, there is no need for cumbersome information mapping.

CIM defines the class hierarchy for Directory Enabled Network (Strassner, 1999). A subset of the class hierarchies is shown below.

Directory Enabled Network
DEN's Physical Element
Physical Package
Secure Package
Rack
Chassis
Network Package
Card
Physical Link
Physical Connector
Slot
Physical Component
Chip
Network ASIC
DEN's Logical Element
System
Computer System
Unitary Computer System
Network Element
Service

Switch Service
Source Routing Service
Transparent Bridging Service
Spanning Tree Service
Network Service
Forwarding Service
Route Calculation Service
Information Service
Service Access Point
Protocol End Point
IPProtocol End Point
IPXProtocol End Point
LAN End Point
Switch Port
Protocol
Network Protocol
DEN's Policy (Top Class)
Policy
Networking Policy
DiffServ Policy
Policy Condition
Networking Policy Condition
Policy Action
Networking Policy Action
DiffServ Policy Action

Directory Enabled Network Design Using JNDI

Directory Enabled Network service should enable the DEN client to access the naming and directory service in order to bind and lookup the instances attributes and associations defined in different DEN classes. A Java package as an extension of JNDI would provide foundation classes and necessary interfaces to implement DEN client for accessing DEN services. JNDI API provides the context and directory context interface, which can be used by DEN client. Attribute and Attributes classes could be used for defining attributes of DEN classes. To implement association among classes we could also implement DirContext interface that will provide all common attributes and behavior for association objects. DEN objects with attributes, and association can be integrated into a single JNDI DirContext.

A hierarchical naming system of DEN is to an implementation of JNDI Context in Java platform and each of the services and resources is to an implementation of Context or DirContext. The services and resources will register their name in DEN naming system at startup time and inform naming server about the reference to the object. Each client can invoke look up method to locate related services from its initial DEN name space and use the service in a standard way. The look up method return an object reference to the context of service provider.

The client will create an instance of InitialContext and search for a specific service. If Printer class have a method print(String fileName), which prints the contents of a file specified by fileName and printer is an instance of Printer class which is available in Namespace of InitialContext, then the client can print a file by writing the following Java code.

```
DirContext ctx = new InitialDirContext();
Printer printer = (Printer) ctx.lookup (printerName);
printer.print(fileName);
```

The following sections describe formats for name in DEN Naming System, representation of DEN directory tree using JNDI context, binding DEN objects, attributes and relationships in DEN directory tree and few programming examples.

Formats for Name in Naming System

DEN directory or object names are represented by JNDI Compound Name and components of Compound Name are separated by forward slash ('/') character from left to right. The full name of an object, including

all the associated naming contexts, is known as a compound name. The first component of a compound name gives the name of a naming context, in which the second component is accessed. This process continues until the last component of the Compound Name has been reached. To make the Compound names human-readable format, we use component of a compound name is a string and components are separated by a forward slash ('/') character. The following grammar provides the rules to resolve compound names.

CompoundName = NameSequence | " "
NameSequence = Name | Name "/" NameSequence
Name = ContexName | DirContexName
ContexName = Letter [letter | Digit]*
DirContextName = Letter [letter | Digit]*
Letter = any one of 52 alphabetic characters A through Z in upper case and a through z in lower case.
Digit = any one of ten digits 0 through 9

DEN Directory Tree using JNDI Context

In the naming service, names are associated with two types of objects: a context or an application object. A naming context is an object in the naming service within which the names of application objects can be resolved. Naming contexts are organized into a naming graph, which forms a naming hierarchy similar to a hierarchical file system. A name that is bound to a naming context corresponds to a directory and a name that is bound to an application object corresponds to a file.

JNDI performs all naming operations relative to a context. To assist in finding a place to start, the JNDI specification defines an InitialContext class. DEN naming service should provide sub context functionality. Like a sub-directory in a file system, a sub context is a context within a context. This hierarchical structure permits better organization of information. For supporting sub context, the DEN naming services Context class defines methods for creating and destroying sub contexts. The following figure shows a naming sub-graph for Directory Enabled Network context.

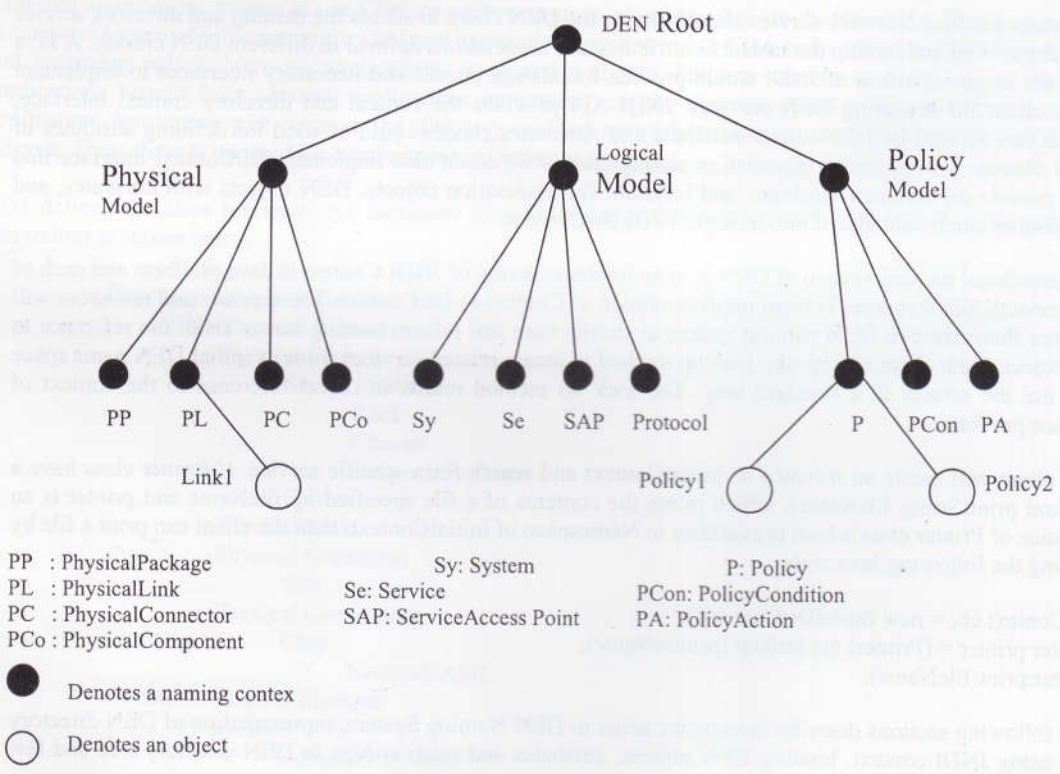

Figure 2. A Naming Context Group

DEN Context represents the root to DEN naming service. According to the above figure, a binding in PhysicalModelctx/PhysicalLink Subcontext of the DEN Context is Link1, where Link1 is an instance of Physical Link class. Again, PolicyModelctx/Policyctx context has binding Policy1 and Policy2, where

Policy1 and Policy2 are instance of Policy class. If we are required to remove an object from a Context e. g. to remove the object PolicyModelctx/Policyctx/policy1 from DEN Context, unbind method of DEN Context is invoked. The Compound names of the objects, e.g., PolicyModectx/Policyctx/Policy1 are resolved though intermediate Contexts.

DEN IntialContext is created based on DEN IntialContextFactory using the information stored in the DEN directory database for constructing DEN logical directory tree. The DEN InitialContext represents the root of DEN context for the naming service. DEN InitialContext restores the directory structure by creating sub-contexts and instantiating DirContexts for objects to be stored in the DEN root directory and subdirectories. DEN Context should keep up a frequent correspondence with the DEN database for most recent information. The DEN directory tree is the data source for DEN Context. For each subdirectory in DEN directory tree, a corresponding sub-context is essential in the naming service, e.g., the root DEN Context should bind three sub-contexts: PhysicalModel, LogicalModel and PolicyModel. The objects stored in different DEN subdirectories are instances of DEN classes for the corresponding subdirectory. The instances of DEN classes are objects to be stored in the directory and DEN Contexts or sub-contexts bind these objects.

DEN Ontology Representation and Programming issues using JNDI

DirContext object could be used to represent different objects in DEN directory. It is recommended that all the classes in class hierarchy of DEN ontology should be the implementations of the Dircontext interface. If it is needed to add an instance of a DEN class in the network i.e., DEN service, that object should be bound in the naming service. An ObjectFactory should be implemented for InitialDenContextFactory so that the initial DEN tree structure containing the existing network resources and services could be restored at the startup time of DEN service based on directory database.

In order to create an initial context for accessing the DEN service of the --- domain, it is assumed that INITCTX variable should provide the appropriate Factory class name for the initial DEN Context and there is DEN server running in localhost and waiting for den request at "PortNumber" of server port. The sample program shown below creates and returns an instance of InitialDenContext.

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",INITCTX);
env.put("java.naming.provider.url","localhost:PortNumber");
Dencontext denctx = new InitialDenContext(env);
```

Each binary relationship between objects is represented as an instance of Attribute class in DirContext, and a value of an Attribute object represents the target instance for the binary relationship. Multiple values are allowed by the instance of Attribute class, which allows the representation of one-to-many relationships in directory service. There are many programming issues, which should be discussed for explaining the DEN design using JNDI. Few programming examples of DEN Ontology representation issues using JNDI are reported below.

Example1: Let us consider a Directory Enabled Network having a router csrouter1, an instance of Chassis class. Chassis is the subclass of PhysicalPackage class. The PhysicalPackage class participate the PackageInConnector relationship. An Attribute named PackagInConnector could be added in Chassis class to represent this relationship. To get the list instances of PhysicalConnector those participate the PackageInConnector relationship with csrouter1, the following code fragments could be used.

```
DenContext chassisctx = (DenContext)
denctx.lookup("PhysicalModelctx/PhysicalPackagectx/SecurePackagectx/Chassisctx");
Chassis csrouter1 = (Chassis) chassisctx.lookup("csrouter1");
BasicAttribute packageInConnector= csrouter1.getAttributes().get("PackageInConnectx");
NamingEnumeration PhysicalConnectors= packageInConnector.getAll();

DenContext physicalConnectorctx= (DenContext)
denetx.lokup("PhysicalModelctx/PhysicalConnectorctx");
While(physicalConnectors.hasmore()){
        PhysicalConnector physicalConnector= (PhysicalConnector)
        physicalConnectorctx.lookup(physicalConnectors.next());
```

```
/* physicalConnector provides the instance of PhysicalConnector class. We can use search method to find
more specific list of instances of PhysicalConnector that participates PackageInConnector relationship. */
        .....
}
```

As each relationship name is defined between two classes, we could get the instance of Chassis class, which participate in PackageInConnector relationship with a particular instance of PhysicalConnector class.

Example 2: In this example we have considered that in network, there may have several printers installed in different locations. PhysicalElementLocation relationship allows us to choose the appropriate printer suitable from our current location. In Physical Model, any Printer is an instance of Chassis class. Chassis is a subclass of PhysicalPackage class and PhysicalPackage is the subclass of PhysicalElement class. We could browse the list of network printers and select the printer resides appropriate Location viewing the PhysicalElementLocation relationship. Assuming that PhysicalPrinter1 is an instance of Chassis class representing a printer in the network. PhysicalPackage class participate ComputerSystemPackage relationship with UnitaryComputerSystem class of Logical Model. UnitaryComputerSystem class is a subclass of System class. System class participate HostedService relationship with Service class. If PrintService is an instance of Service class and LogicalPrinter1 is an instance of UnitaryComputerSystem then LogicalPrinter1 participates HostedService relationship with printService object. We can access printService from the DEN root context through Physical Model by writing the following set of codes.

```
NamingEnumeration items =
denctx.list("PhysicalModelctx/PhysicalPackagectx/SecurePackagectx/Chassisctx");
while(items.hasmore()){
        Object item=(Object) items.next();
        if(item instanceof Chassis)
        {
        BasicAttribute location=item.getAttributes().get("PhysicalElementLocation");
        if(location.contains(new Location("Location1")))
        {
        /* code for display/select the instances of Chassis at location Location1. User can choose the
        appropriate Chassis for physical printer */
        ...
        }
      }
}
```

Assuming that after executing the above code, user selected the printer PhysicalPrinter1 object to use print service.

```
BasicAttribute computerSystemPackage=
PhysicalPrinter1.getAttributes().get("ComputerSystemPackage");
NamingEnumeration unitaryComputerSystems= computerSystemPackage.getAll();

DenContext unitaryComputerSystemctx= (DenContex)
denctx.lookup("LogicalModelctx/Systemctx/ComputerSystemctx/UnitaryComputerSystemctx");

while(unitaryComputerSystems.hasmore()){
        UnitaryComputerSystem unitaryComputerSystem= (UnitaryComputerSystem)
        unitaryComputerSystemctx.lookup(unitaryComputerSystems.next());

        /* unitaryComputerSystem provides the instance of UnitaryComputerSystem class. We can use
        search method to find more specific list of instances.*/
        ...
}
```

Assuming that unitaryComputerSystem has a value equals to LogicalPrinter1 and we selected LogicalPrinter1 object to use print service.

```
BasicAttribute hostedService= LogicalPrinter1.getAttributes().get("HostedService");
NamingEnumeration services= hostedService.getAll();
DenContext servicectx= (DenContext) denctx.lookup("LogicalModelectx/Servicectx");
while(services.hasmore()){
```

```
        /* Service provides the instance of Service class. We can use search method to find more specific
        list of instances.*/
        ...
}
```

Assuming that we want to use the printService service from LogicalPrinter1 and this service has a method print() for adding a print job in print queue.

```
DenContex servicectx= (DenContext) denctx.lookup("LogicalModelctx/Servicectx");
Service printService= (Service) Servicectx.lookup("printService");
printService.print(new FileInputStream(fileName));
```

Conclusion

Naming and directory services play a vital role in network by providing network-wide sharing of a variety of information about users, machines, networks, services and applications. Applications can share the common storage area provided by the directory. This sharing makes applications that are installed across the network system, and the network, more consistent and manageable. Several classes and relationships define the DEN Ontology which could be extended by adding new classes and relationships depending on application needs. An explicitly defined Ontology for Directory Enable Network would facilitate an integrated network management in a multi-vendor network. An experimental evaluation of the proposed framework would be carried on a prototype network in order to enable reliable DEN services over different existing networks.

References

Ahmed, Z., 2000. An Overview of Active Directory, Available at <URL:http://www.winntmag.com/Articles /Index.cfm?ArticleD=8178>

Anderson, A., 1996. The Network Information system (NIS). Available at <URL: http:www.linuxdoc.org/LDP /nag/node 130.html>

Blum, D. J., 1999. Understanding Active Directory Services, Microsoft Press.

Butterfield, J., 2000. Directory Assistance: A basic understanding of LDAP directory operations and how they work with JNDI will help you organize your data. Available at<URL:http:/www.javapro.com/upload /free/feature/javapro/2000/12dec00/jboo12/jboo12.asp>

Chadick, D.W., 1996. Understanding X.500- The Directory, Available at <URL: http:/www.salford.ac.uk /its024/Version.Web/Contents.htm>

CISCO, 1999. Directory Enabled Networking Available at <URL: http://www.cisco.com/univercd/cc/td/doc /cisintwk/ito_doc/diren.htm>

CISCO, 2000. Directory Enabled Networks (DEN)-Frequently Asked Questions (FAQs). Available at <URL:http://www.cisco.com/warp/public/cc/techno/network/drsrv_qp.htm>

DMTF, 2001. Distribute Management Task Force (DMTF) Homepage. Available at <URL: http://www. dmtf.org>

Finin, T., Mekay, R.F.D., and McEntire, R., 1994. KQML as an Agent Communication Language. *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, ACM Press.

Finin, T., Labrou, Y., and Mayfield, J., 1997. KQML as an agent communication language, in Jeff Bradshaw (Ed.), *Software Agents*, MIT Press, Cambridge

Gruber, T., 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human and Computer Studies, 43(5/6): 907-928.*Gruninger, M., and Fox, M.S., 1995. Methodology for the Design and Evaluation of Ontologies, in Workshop on Basic Ontological Issues in Knowledge Sharing, *International Joint Conference of Artificial Intelligence.*

Guarino, N., 1995. Formal Ontology, Conceptual Analysis and Knowledge Representation. International Journal of Human and Computer Studies, special issue on Format Ontology in the Information Technology edited by N. Guarino and R. Poli, vol 43 no. 5/6.

Guarino, N., and Giaretta, P., 1995. Ontologies and Knowledge Bases : Towards a Terminological Clarification. In N.J.I. Mars (ed.), Towards Very Large Knowledge Bases, IOS Press.

Jones, D.M., Bench-capon, J.J.M., and Visser, P.R.S., 1998. Methodologies for Ontology Development, *In Proc, IT&KNOWS Conference, XV IFIP World Computer Congress*, Budapest

Judd, S., and Strassner, J., 1998. Directory Enabled Networks (DEN)-Information Model and Base Schema draft version 2.0.1-3

Knowledge Based System Inc., 2000. DEF5 Ontology Description Capture Overview. Available at <URL: http://www.idef.com/idef5.html>

Microsoft Press, 2000. Implementing Directory Enabled Networks using Windows 2000 Technology: [Active Directory, DEN, Windows 2000], 2000. Available at <URL: http://www.microsoft. com/ windows2000/libray/technologies/communications/denuse.sp>

Poli, R., 1995. Bimodality of Formal Ontology and Mereology, International Journal of Human and Computer Studies, special issue on *Formal Ontology in the Information Technology edited by Guarino, N., and Poli, R., vol 43 no. 5.6.*

Scot, G., and Joseph, C., 1999. Use JNDI as your network's "white pages" for easy access to distributed resources, Making Enterprise Connection with JNDI and LDAP, Available at <URL: http://www.devx.com/upload/free/features/javapro/1999/12dec99/sg1299.asp>

Strassner, J., 1999. Directory Enabled Networks, Machmillan Technical Publishing, USA.

Sundsted, T., 2000. JNDI overview, Part 1: An Introduction to naming services- Use JNDI to better manage your distributed applications, Available at <URL: http://www.javaworld,com/javaworld/jw-01-2000/jw-01-howto.html>

Sun Microsystems, Inc, 2000. Java Naming and Directory Interface Application Programming Interface (JNDI API). Available at <URL:http://java.sun.com/products/jndi/does.html>

Uschold, M., 1996a. Converting an Informal Ontology into Ontolingua: Some Experiences, ECAI-96 Workshop on Ontological Engineering, Budapest, August.

Uschold, M., and King, M., 1995. Towards A Methodology for Building Ontologies. IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada.

Yeong. W., Howes, T., and Kille, S., 1995, RFC-1777: Lightweight Directory Access Protocol, Available at, <URL: http: //www.unich.edu/-dirsvcs/ldap/doc/rfc/rfc1777.txt>