DMIS DISTRIBUTED SERVICE ARCHITECTURE

DMIS Core Framework v3.0.1

Conceptual Overview

July 2005

Revision History

Date	Version	Author	Remarks
May 2005	1.0.1	J. Kessler	Initial version
July 2005	1.1.1	J. Kessler	

Table Of Contents

Table Of Figures

Abstract

The DMIS Core Framework continues to evolve in this latest incarnation, serving DMIS by providing a simple, modular, and scalable architecture. New features include support for COG-managed DMIS servers (running on LANs) for decentralization and scalability, flexible connections between DMIS entities that inherently support clustering and redundancy, and new tools to simplify configuration and maintenance. Updates to the internal model also improve the consistency and capability of the developer interfaces, enabling developers to create DMIS capabilities that function in a wide variety of DMIS topologies. All of these enhanced functions continue the existing emphasis on low-barrier implementation and distribution of DMIS.

1. Introduction

1.1. Purpose

This document is an overview of the DMIS Application Services architecture. It provides a conceptual explanation of the Distributed Services Architecture complete with illustrations that is suitable for a general technically-oriented audience.

1.2. Audience

The intended audience is anyone who is interested in understanding the goals, features, and design of the DMIS Distributed Services approach

2. Design Goals

The service infrastructure is intended to meet certain goals. Service developers should also take these goals into consideration when planning how to implement a service, since some of them could impact the optimal design of services.

2.1. Simplicity

Implementing a service should be as easy as possible, consisting of clearly defined steps and guidelines when creating the shell for a basic service.

2.2. Modularity

Services can be developed, deployed, and updated independently from other services (except where interdependencies exist between services due to their functionality). In most cases services can be deployed and undeployed dynamically without even restarting the corresponding DMIS AppServer. Message mapping and workflows are described in descriptor files and can be modified easily.

Services and descriptors can be designed to work in a variety of contexts (e.g. embedded within the desktop, at a LAN level, or on the CENTRAL Server) using the same code and metadata. There are also new modularity features that did not exist at all in the previous version of the middleware, such as the ability to send request to an abstract service (by its symbolic name) without knowing where it exists in the hierarchy or network. Some of these features are described in this document.

2.3. Scalability

The AppServer is inherently scalable and services automatically benefit from many of its scalability features, such as dynamic message routing, service pooling, and clustering support.

Even given these features, though, services must be designed so that they observe best practices for scalability. This document addresses some specific dangers and provides suggestions to avoid them. For example, in the case of services that are intended to run locally on each workstation, consideration must be made for low-end stakeholder machines and slow dial-up connections.

2.4. Low cost barrier

The AppServer is flexible and runs on a wide variety of machines with varying hardware capacity. Indeed, the normal DMIS desktop client runs on relatively low-end machines and contains an embedded AppServer running much of the same code as a CENTRAL Server. The main difference is in the set of services and modules that are deployed in each environment.

This level of flexibility makes it possible to install DMIS services on stakeholder LANs to allow them direct control over the installation. Given that DMIS is provided for free, coupling with an open-source database (such as McKoi) for LAN deployments allows DMIS to be installed very cost-effectively. In addition, DMIS provides tools to simply the configuration (such as the DMIS Configuration tool) and monitoring (such as the ServerDiag utility) of a DMIS AppServer installation while effectively leveraging its features.

Other similar architectures might rely more on costly hardware or software, or elements that would be too unwieldy for stakeholders to administer.

3. Architecture Overview

DMIS uses *distributed service architecture* to address entities, route messages, and handle requests. The system is “distributed” in nature from a variety of perspectives:

- It supports a multiple server hierarchy that decentralizes request handling, enhancing scalability and providing more control to organization using DMIS.
- It provides “location independence” for operators, meaning that they can connect to the DMIS network from more than one entry point (such as at home or in the field) and retain their identity and context. Logical identities are decoupled from physical connections.
- It allows arbitrary secondary inter-networking and relationships between entities on the DMIS network to be created, enabling the formation of a true DMIS “web”. This, in turn, provides for great flexibility and excellent resilience against environmental anomalies.

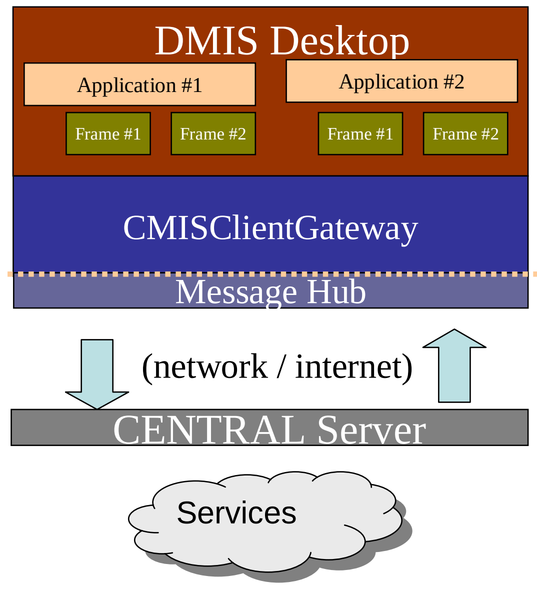
The distributed architecture borrows some of the best concepts from the Internet and builds upon them within the DMIS problem domain. The implementation is purposely kept relatively simple, small, and scalable so that it may work in a variety of situations.

A useful starting point for understanding DMIS architecture is to review the flow of messages in a relatively simple scenario. This helps to identify the major middleware pieces involved in a DMIS transaction.

3.1. Thin Client Example

Assume the simplest case in which the Desktop and is connecting to the CENTRAL Server, and there is no EMBEDDED AppServer on the client side to support offline behavior. This setup is sometimes called the “diet” or “thin” mode because it includes mainly the GUI portion of DMIS.

The Desktop client perceives these major elements:



Desktop is the high-level GUI container for all DMIS applications.

Applications are given a reference to CMISClientGateway by the desktop as an interface to the middleware (and thus all of the deployed services). Apps access services by sending messages to them and waiting for replies.

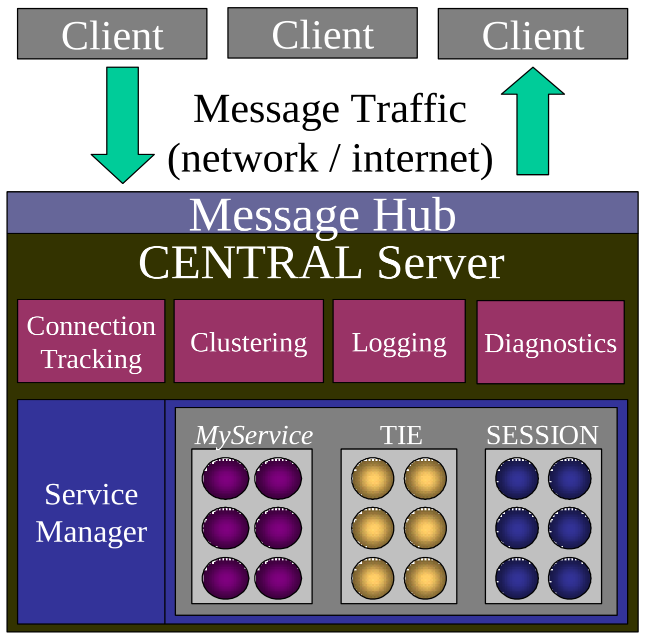
Requests are sent in the form of messages that conform to the API defined by services that are deployed. Responses are provided *in most cases* (as an acknowledgement) in the form of messages sent back to the client by the service.

Everything below this line is transparent to Applications, which should deal only with messages and presentation issues.

Figure 1: DMIS Desktop perception of DMIS services

Thus, the client has access to the server via two interfaces: the CMISClientGateway that provides functional access to the messaging mechanism, and the message definitions that comprise the API for each service.

The AppServer views the operation from the other side of the world:



The AppServer awaits incoming messages from clients connected through the DMIS messaging protocol. The AppServer provides a broad array of modules and services to simplify and shield services from as many details as possible. This allows services to remain focused on processing messages.

The Service Manager is an important module within the AppServer that distributes messages to services based on rules provided in deployment descriptors.

The Service Manager keeps a “pool” of each type of service. When a new message is received from an operator, an instance of the appropriate service is extracted from the pool to handle the message. Each instance on the pool handles no more than one message at a time, so the size of the pool is related to the number of simultaneous requests that can be handled.

Services themselves can send response messages and use other facilities of the AppServer through interfaces provided for it and passed upon initialization.

Figure 2: Perspective from the CENTRAL Server

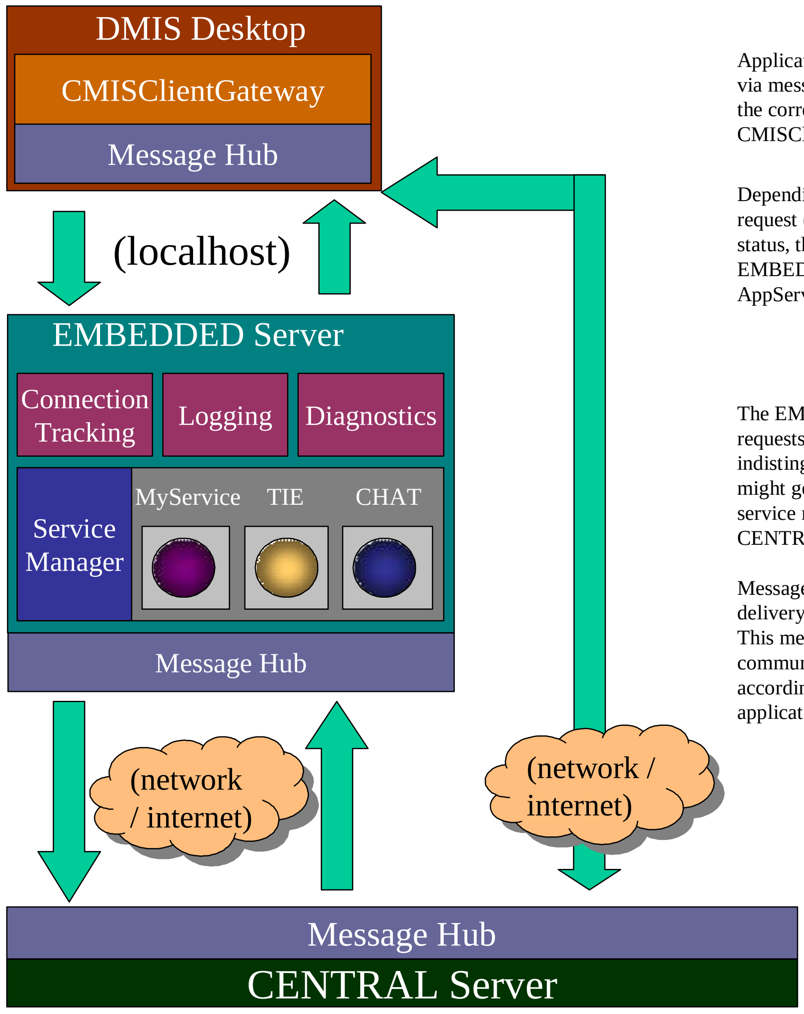
3.2. EMBEDDED Application Servers

In the typical DMIS client installation, the Desktop supports a degree of functionality when the machine is offline or network problems arise. This is accomplished by installing the EMBEDDED AppServer on the client machine that can handle certain kinds of requests (according to what the locally-installed services allow). This allows the client, in some cases, to behave as though it were online or to implement special behaviors for offline mode. From the operator’s perspective, it improves the fault tolerance of DMIS.

The EMBEDDED AppServer has the same role as the “Local Service Manager” in previous versions of the middleware. The name change is largely due to changes in the underlying architecture and for internal consistency.

Because the EMBEDDED AppServer is comprised of essentially the same code as the CENTRAL AppServer (or any other DMIS server), differing mainly by the modules deployed with it, a service can be written to run in both contexts fairly easily. The current version of the middleware provides more robust support for this approach than did the previous versions.

Assume the classic case in which the Desktop has an embedded AppServer (meaning that it supports offline mode) and is connecting to the CENTRAL Server. The following diagrams show the major pieces involved:



Applications within the DMIS Desktop communicate via messages (with content conforming to the APIs of the corresponding services) through the CMISClientGateway instance given to them.

Depending upon the workflow rules for the given request (as defined by the service developer), or offline status, the message is routed. It may go to the EMBEDDED AppServer or directly to the CENTRAL AppServer.

The EMBEDDED AppServer can process many requests and provide a reply to applications that is indistinguishable from ones the CENTRAL server might generate. Alternately, the EMBEDDED service may decide to forward the request to the CENTRAL server if it cannot process the request.

Message queuing features (such as guaranteed delivery) are applied at every step along the way. This means that transmissions can survive communication disruptions and are protected according to sensible defaults or as dictated by applications and services.

Figure 3: Positioning of EMBEDDED servers

The developer of a service does *not* need to understand all of the mechanics shown here. They are provided mainly for context. Services implement a very limited number of interfaces, and have fairly simple interaction rules that allow them to “snap into” the puzzle quite easily. Some services are more complex than others and may use more of the features provided, which does require varying degrees of architectural knowledge.

3.3. Multi-tiered Application Servers

One of the major new features in the middleware is support for more complex networks of AppServers. Although the Messaging Infrastructure supports a wide variety of topologies, the DMIS model is generally hierarchical in nature (though it does exhibit some peer-to-peer capabilities in some specific areas).

3.3.1. LAN Servers

The AppServer hierarchy builds upon the EMBEDDED <-> CENTRAL relationship by introducing a new level: the LAN server. Support for LAN installations has been a goal since the inception of DMIS.

In this version the goal is finally realized. Bringing it to fruition has taken significant time because of the complexity involved in its design, implementation, and impact.

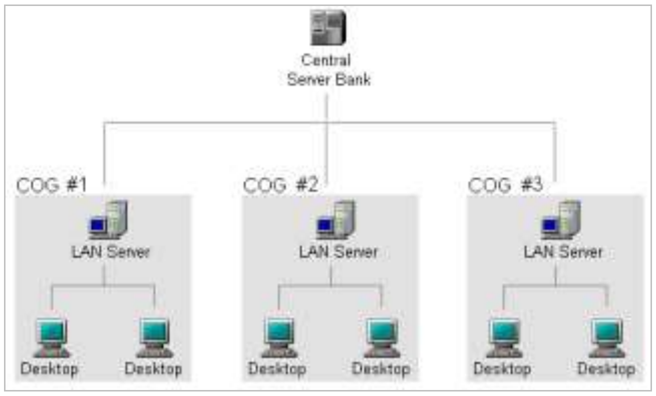


Figure 4: LAN servers deployed in hierarchy

The addition of LAN servers offers several benefits:

- Operating groups can now operate, administer, and control their own DMIS AppServer installations as they see fit.

- Additional fault tolerance by mitigating the single point of failure that up until now has been the CENTRAL AppServer.

- Improved scalability by distributing the transaction load across installations rather than channeling all of them through the CENTRAL installation.

A new layer in the AppServer hierarchy does complicate service development because it requires more design discipline in order to keep things clean and consistent among tiers, and more rigorous testing of services. The good news is that, along with the additional capabilities and complexities, new facilities have been created to keep pace and assist with service design and development.

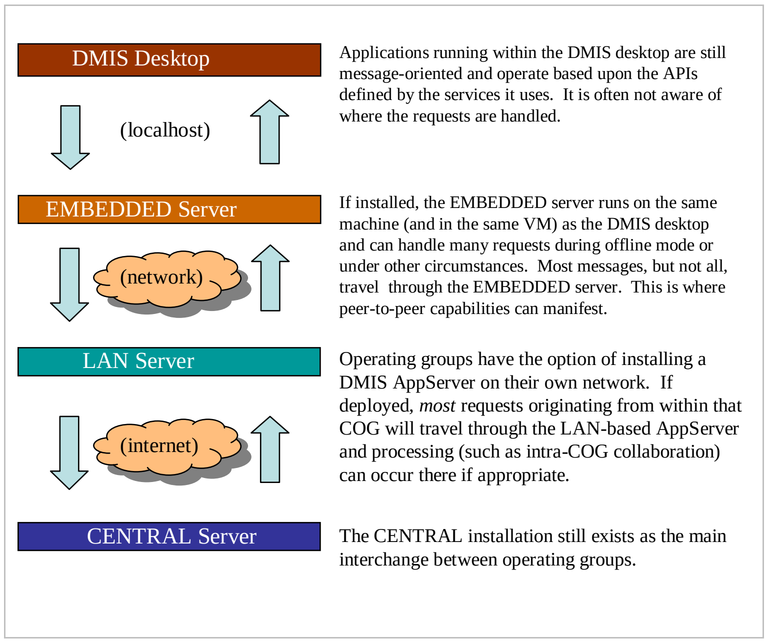

DMIS Desktop
(localhost)
Applications running within the DMIS desktop are still message-oriented and operate based upon the APIs defined by the services it uses. It is often not aware of where the requests are handled.
EMBEDDED Server
(network)
If installed, the EMBEDDED server runs on the same machine (and in the same VM) as the DMIS desktop and can handle many requests during offline mode or under other circumstances. Most messages, but not all, travel through the EMBEDDED server. This is where peer-to-peer capabilities can manifest.
LAN Server
(internet)
Operating groups have the option of installing a DMIS AppServer on their own network. If deployed, *most* requests originating from within that COG will travel through the LAN-based AppServer and processing (such as intra-COG collaboration) can occur there if appropriate.
CENTRAL Server
The CENTRAL installation still exists as the main interchange between operating groups.

Figure 5: DMIS hierarchy including LAN tier

It is important to note that all of the AppServers, exception the CENTRAL Server, may or may not be present in any given hierarchy. Facilities for discovering and handling the hierarchy from a design and implementation perspective are discussed in this document. *In most cases it is possible to develop a service that functions in all contexts and combinations using the same code*. Part of the job of the AppServer is to simplify this, so is important to ask the appropriate experts when a problem arises rather than immediately duplicating code. If the right facility does not already exist, then perhaps it should (and will)!

3.3.2. Hierarchy Depth

Though the "official" DMIS server hierarchy is defined as a maximum of three levels deep, EMBEDDED <-> LAN <-> CENTRAL, the middleware actually supports any arbitrary number of levels.

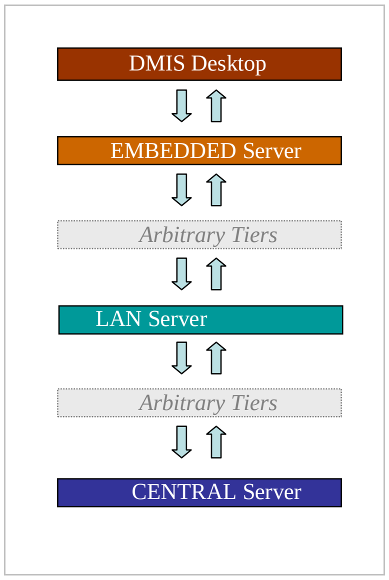


Figure 6: DMIS hierarchy concept with arbitrary tier depth

Thus, the three-level hierarchy is a relatively arbitrary and was chosen to meet the specific needs of DMIS. Since more levels may be introduced in the future, it is important for service developers to observe the best practices described in this document all of the facilities provided in order to keep their modules flexible enough for future growth.

3.3.3. Parallel Hierarchies

Not all operating groups will have the same relative hierarchy. Some will operate LAN servers, some will support offline mode through the use of EMBEDDED servers, and some may connect directly to the CENTRAL Server. Further, any combination of these connection modes is possible even within a single COG.

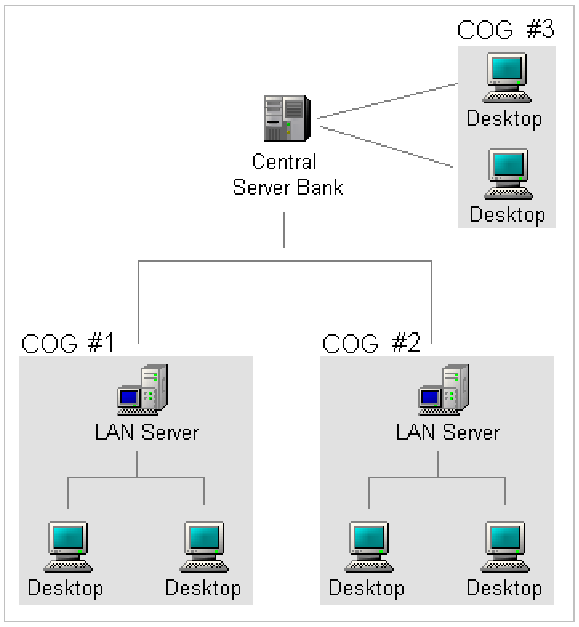


Figure 7: LAN servers deployed in hierarchy

Members of the same COG may choose to access the DMIS network in different ways depending upon their requirements. It may be easier for someone working in the field using a laptop and cellular uplink, for example, to connect to the CENTRAL Server directly even though their COG operates a LAN.

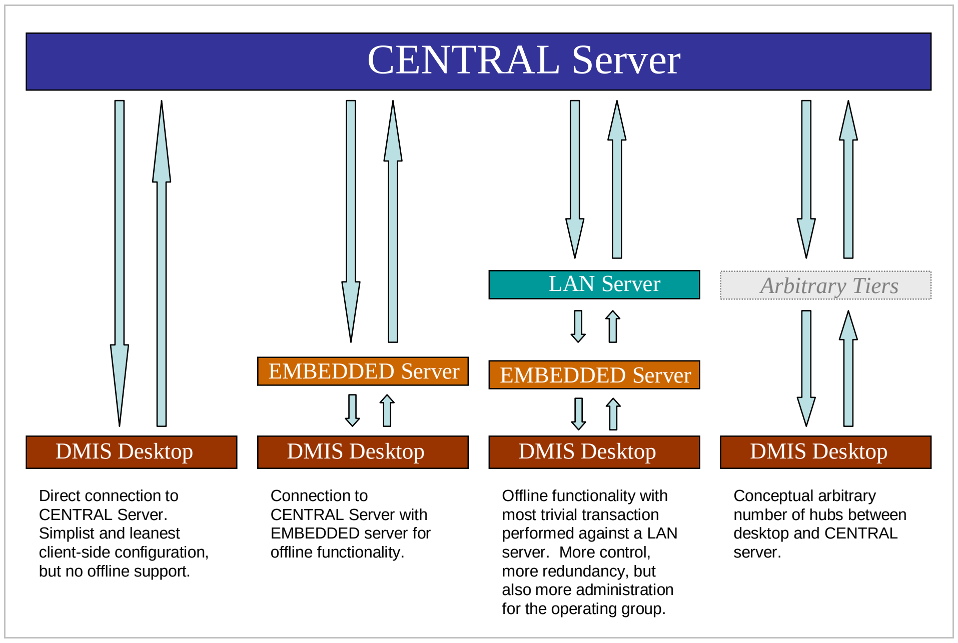

CENTRAL Server
LAN Server
Arbitrary Tiers
EMBEDDED Server
EMBEDDED Server
DMIS Desktop
DMIS Desktop
DMIS Desktop
DMIS Desktop
Direct connection to CENTRAL Server. Simplist and leanest client-side configuration, but no offline support.
Connection to CENTRAL Server with EMBEDDED server for offline functionality.
Offline functionality with most trivial transaction performed against a LAN server. More control, more redundancy, but also more administration for the operating group.
Conceptual arbitrary number of hubs between desktop and CENTRAL server.

Figure 8: COG hierarchies in parallel - different routes to the CENTRAL Server

For this reason, services need to be tested in a variety of contexts. Using the facilities provided by the middleware, developers can create services that work in all contexts and deployment models.

3.4. Automatic Context Detection

The DMIS messaging subsystem is designed such that entities (such as the Desktop or an AppServer somewhere in the hierarchy) can automatically detect their position and negotiate relationships with entities immediately up and down the hierarchy. This is made possible by the dynamic messaging subsystem that uses symbolic names (rather than host URLs) to address specific entities. If fact, the desktop need not be connected directly to a LAN server but can operate as though it were as long as an indirect connection is present.

3.5. Collaboration in a Distributed Environment

3.5.1. Decentralization

The distributed service model mitigates many of the risks inherent in the centralized approach by providing more collaboration channels and reducing reliance on the central installation. When a LAN server is deployed for a COG, for example, most collaboration within that unit does not require services from the CENTRAL Server at all. This tends to make local operations more responsive, and removes processing burden from the central installation. Some organizations may also prefer having an installation over which they have direct physical ownership.

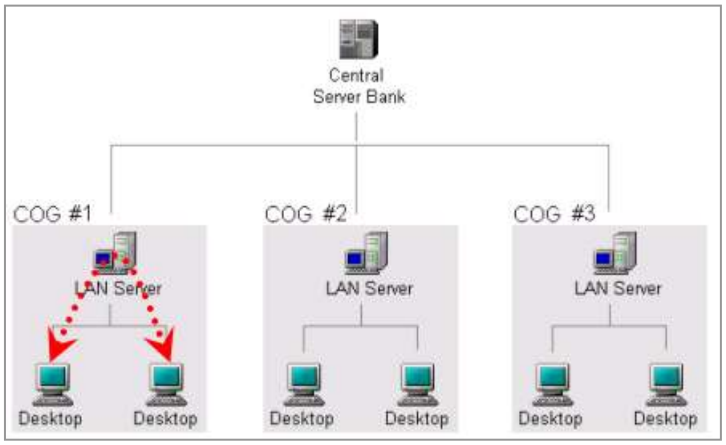


Figure 9: Collaboration within a COG via LAN server

In the “official” distributed deployment model provided by DMIS, operations that span multiple COGs (such as posting an incident or cross-COG chatting) still generally require mediation from the CENTRAL Server. This is simply because the CENTRAL Server is the only node to which both COGs are directly or indirectly connected.

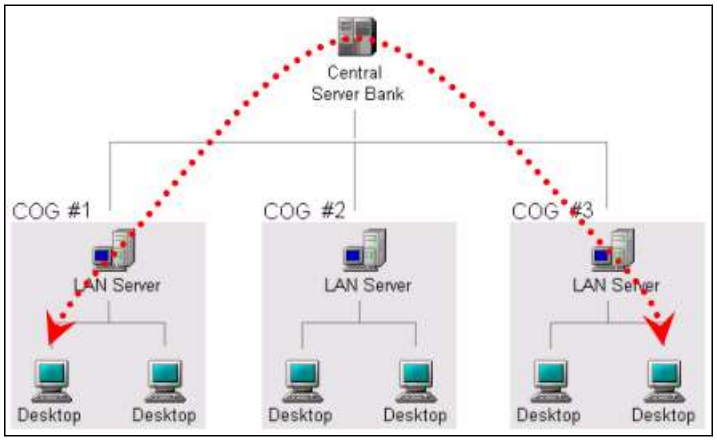


Figure 10: Collaboration across COGs

Although not a officially supported communication mode, it is technically possible to exchange information directly between COGs without involving the CENTRAL Server. Out of the box, the DMIS network is a hierarchy but in its most basic form it is not necessarily restricted to that layout. Direct connections between COGs can be established and thereby provide more direct channels between organizations that may interact closely.

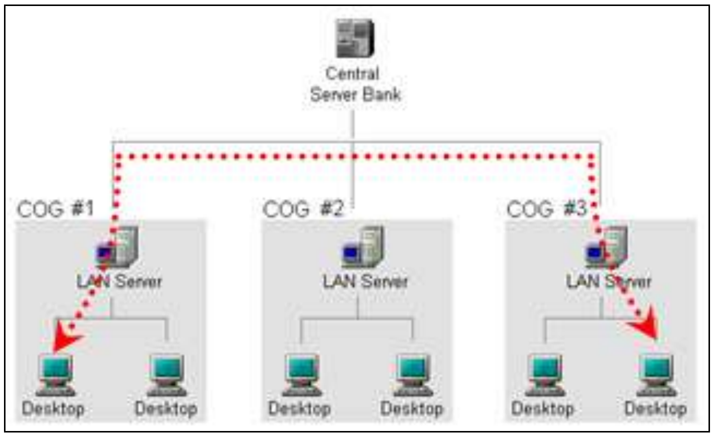


Figure 11: Direct collaboration between COGs

These features can greatly enhance the scalability, security, and flexibility of DMIS, and ultimately serve to meet the needs of stakeholders.

3.5.2. Supported COG Deployments

The following illustrations depict the various entry points into the DMIS network that are officially supported as part of the distributed service model.

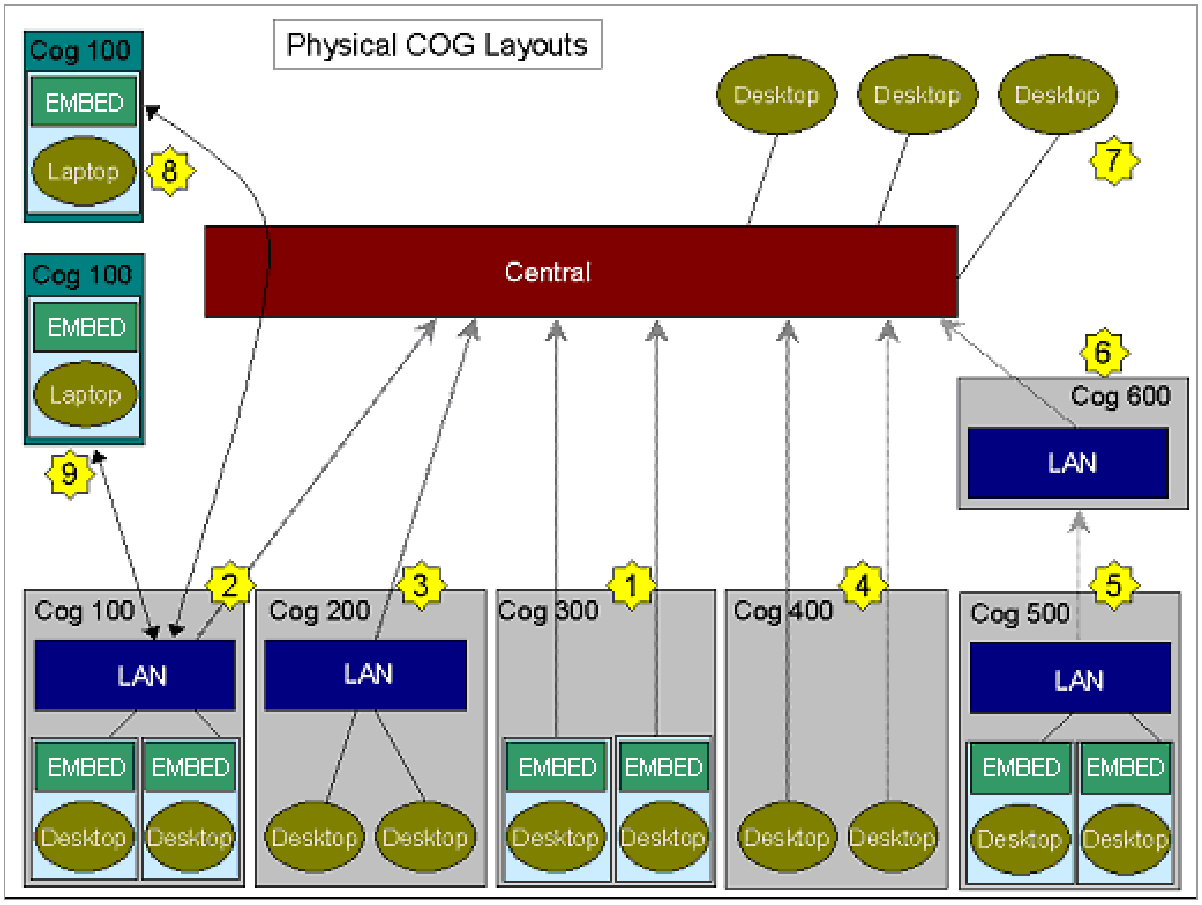


Figure 12: Physical connection points into the DMIS network

1

COG 300

COG configured in the "classic" manner, each workstation having a private database and EMBEDDED server running on the desktop.

- Each operator has offline capability
- Server is maintained by the central DMIS staff
- Inexpensive to install and operate

- *Full DMIS desktop may be too rich for older machines*
- *Limited in-COG collaboration when Central unavailable*

2

COG 100

COG with offline-capable desktops, but connected directly to a private LAN server. The LAN server is connected in hierarchy, in turn, to the CENTRAL Server.

- Each operator has offline capability
- Organization has in-COG collaboration even when CENTRAL Server unavailable
- Organization retains ownership of all COG-centric data

- *Organization assumes responsibility for LAN server maintenance*

3

COG 200

4

COG 400

Desktops with *no offline capability*, but a LAN server that operates independently from the CENTRAL Server. The LAN server may physically reside on one of the desktops or on a dedicated machine.

- Desktop has a lightweight footprint relative to rich client
- Desktops can operate as long as LAN server is available

- *No offline capability at the desktop level; tethered to LAN*
- *Organization assumes responsibility for LAN server*

Desktops with *no offline capability* and connected directly to the CENTRAL Server.

- Desktop has a lightweight footprint relative full client
- Server is maintained by the central DMIS staff
- Inexpensive to install and operate

- *No offline capability at the desktop level*

5 COG 500

COG that is linked in hierarchy to another COG.

- Server structure can mirror organizational hierarchy
- COGs can be organized in as a tree or as a network
- Information goes to CENTRAL Server when it leaves the cluster of interrelated COGs via posting

- *Fairly complex to establish*
- *Requires clear delineation of process flow with participating organizations*

6 COG 600

COG that serves as a clearing point for other, subordinate COGs.

.

- Functions as a normal COG, with same configuration options
- Serves as logical center for one or more subordinate organizations
- Information goes to CENTRAL Server when it is posted

- *Requires clear delineation of process flow with participating organizations*

7 Desktops Connected Centrally

Desktops connected directly to the CENTRAL Server. They may be configured for offline behavior, or not (depending upon what best suits the operator). The configuration that is shown does not include offline behavior.

8 COG 100 Tunneling Access

Workstation in the field connected to its home COG by "tunneling" through the CENTRAL Server. In this configuration, the operator is taking advantage of the CENTRAL Server as a messaging router in order to provide location-transparent access to his home COG. All operations proceed as though connected at home base on COG 100.

	• Location transparency offers flexibility and resilience • Capability available wherever access to DMIS is found, with some limitations • *Dependent upon configuration, security, and firewall settings*

4. Messaging Subsystem

4.1. Messaging Features

The messaging subsystem offers a wide variety of features and behaviors that form the basis of the DMIS architecture. It is performance-oriented and designed for ease of use.

4.1.1. Asynchronous Point-to-Point Messaging

The messaging subsystem rests on an asynchronous transmission model. Messages travel from source to destination across nodes, and are queued at each "hop" along the pathway. Messages make their way, point by point, to their destination in a "lazy" fashion. Because of the efficiency of the DMIS messaging system and the way applications are built upon it, though, messaging often appears to occur in real-time much like in a client-server system. The underlying asynchronous nature of it, however, is crucial in allowing DMIS to survive certain kinds of network and server outages while remaining responsive.

Although DMIS messaging has, at its core, an asynchronous model, synchronous messaging can be performed as well. This RPC-like pattern is very common and the DMIS middleware has considerable support for it in the sendAndWait( ) series of functions described in this document.

DMIS provides for the flow of requests from one entity to another by providing *message queues*. Messages are held in queue until they can be

4.1.2. Message Persistence / Guaranteed Delivery

DMIS offers guaranteed delivery of messages. Messages queues are built upon a transactional model similar to those used by DBMSs such as Oracle that is designed to protect the messages at every step from their transmission to their delivery. At each stopping point along the way, messages are stored safely and managed as encoded flat files until they can be delivered or moved along to the next location.

4.1.3. Intelligent Routing and Tunneling

Messages in DMIS are addressed through the application of symbolic RoutingIDs. These RoutingIDs denote the logical layout of the DMIS hierarchy, and may or may not map exactly to the physical TCP/IP connection points defined between machines. The messaging subsystem is designed to route messages to reach their destination by the most efficient path according to the network of DMIS entities. In some cases this may mean that messages pass through several intermediate "hops" prior to reaching the final logical target. The messaging subsystem keeps a dynamic internal map that allows it to find logical locations on the physical network in a seamless manner.

These routing features allow great flexibility in connecting to DMIS. An operator can connect to DMIS from through a LAN server at his "home base" or connect through the CENTRAL Server when working in the field. The logical layout for an operator is always the same regardless of the TCP/IP connection into the network.

4.1.4. Dynamic Discovery of Topology

DMIS can automatically detect when a LAN server is present for a given operating group, and whether an EMBEDDED server (for enabling offline mode) is configured, and act accordingly. The overall network topology (which for DMIS is roughly a hierarchy) is contained in a descriptor file (called, oddly enough, *NetworkTopology.wfd*) which the messaging subsystem interprets for the particular context. This descriptor file also defines the default "workflow" for messages when no specific patterns are specified by an application (e.g. the Destination for a message is not explicitly filled out).

4.1.5. Configurable Message-Level Workflow

Applications can now define *workflows* using **.wfd* (workflow descriptor) files rather than coding them directly within the corresponding services. Prior to this version of the middleware, it was common for services to check for connectivity to the next-higher server and optionally forward the request for processing higher up the chain. Since this can now be described by a **.wfd* file, the service is relieved of this duty and can be coded as though messages are always delivered specifically to that instance. This can simplify the service code considerably and allow it to work in a broader set of contexts than the same logic when hard-coded.

Workflow descriptors are usually configured one-per-application and can be deployed "hot" (on a Desktop or AppServer that is already running). This allows applications to be installed or deployed as self-contained units, including all of the messaging rules that they require, with minimal impact to other installed applications.

4.1.6. Message Protocol Features

The messaging protocol is the unit that transmits messages from point to point using the Routing Table (described in this document) as a guide. There are a number of important features to this module:

- **Transactional transmission model**. DMIS message queuing is transactional in nature, and message transmission is the most likely point of failure as a message travels from source to destination. The core protocol must be able to handle and recover from anomalies in order to support the promise of Guaranteed Delivery. Transmissions between entities are tightly choreographed to ensure proper handling in the event of network anomalies and protocol collisions.

- **Non-blocking I/O for scalability**. A feature of particular interest to AppServers, the messaging protocol is design to handle a large number of clients while utilizing as few threads to service them as possible. A dedicated, self-pruning ThreadPool services the set of all clients connected through a given channel by scheduling polls based the sensation of incoming data or the availability of new outbound messages. Because, unlike a traditional socket servicing approach, the number of threads is not linearly related to the number of connected clients, the overhead for servicing connections is reduced exponentially.

- **Multiplexed framing for simultaneous transmission**. The message transmission protocol is able to send and receive multiple messages at once through a single node-to-node socket connection. This allows large messages to be transmitted without blocking other messages, and also allows message priorities to be honored at the transmission level. This is accomplished by breaking all messages into "chunks" or a predetermined size and transmitting them one "chunk" at a time until the entire message has been transferred. This approach allows multiple messages to flow in any direction because the transfer of "chunks" can be *interleaved* and *prioritized* to allow some messages to flow more quickly while never completely block the link.

- **Distinguishing normal/abnormal disconnects**. This version of the message protocol can also now distinguish normal (controlled) disconnections from unexpected ones (presumably to due hardware or software failure), and can provided this information to running services and applications so they may also act accordingly. This can simplify certain problems such as session handling by providing more information regarding the nature of the event.

4.1.7. Inherent Support for Clustering

The symbolic nature of message addressing (using RoutingIDs) is well-suited to a clustered environment. Messages can be targeted at a cluster of AppServers as easily as they can focus on a specific machine, since multiple AppServers (each a superset of a given RoutingID format) may match a given symbolic name. The dynamic routing features of the messaging subsystem allow messages to flow freely across AppServers, even with one AppServer directly addressing operators connected to other AppServers, as the whole network functions as a single logical unit.

4.1.8. Low Bandwidth Support

The message transmission protocols used by DMIS are designed to perform very well under low bandwidth conditions. Through the application of custom serialization methods (known in DMIS as "binary serialization"), dynamic ZIP compression, and "chunk" optimization based upon load, message transmissions are extremely efficient.

4.1.9. SSL Support

DMIS supports the use of SSL encryption between all TCP/IP connection points, either through hardware or through built-in software. In addition, DMIS can perform encryption at the message-level, which can be useful when messages may be sensitive and the physical route of the message throughout the DMIS network cannot be assumed. Whereas SSL security applied only during transmissions from node to node, message-level security is applied from the source to the intended destination no matter how many nodes are in between.

4.2. Peer-to-Peer Characteristics

In the discussion so far, the interaction between entities (clients and various AppServers) has been presented essentially as a hierarchy, but DMIS is not strictly limited to this model. Some, more flexible, attributes do exist and can manifest even from within the basic hierarchy that defines DMIS. Most of these features arise from the flexibility of the underlying point-to-point messaging implementation.

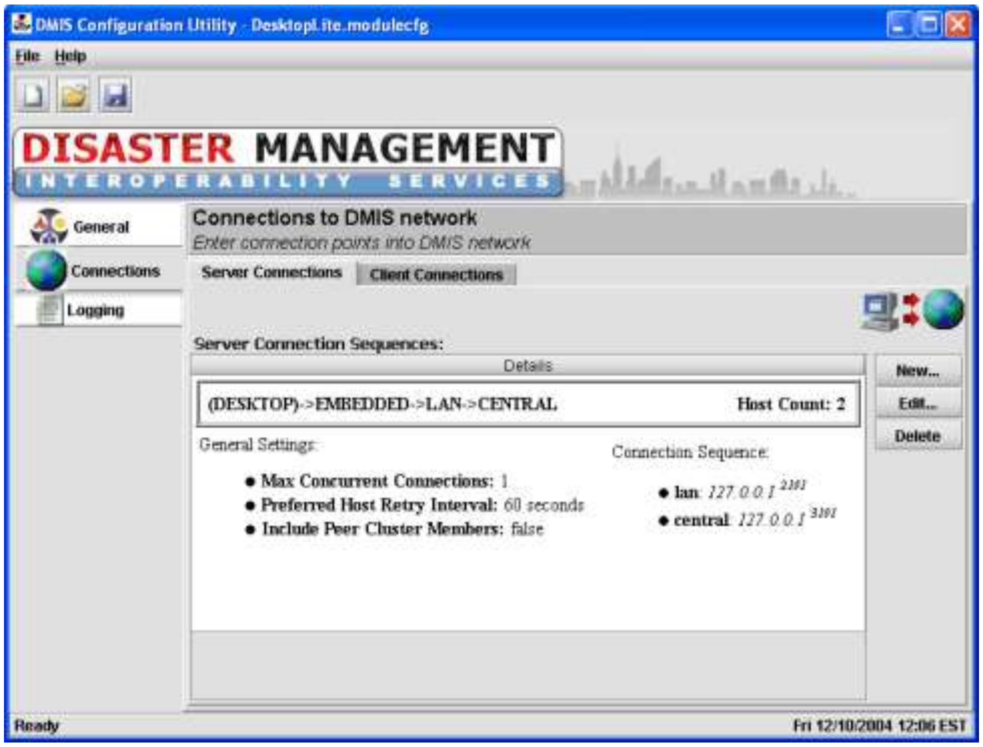
4.2.1. Homogeny of Entities

All entities on the DMIS network, including the DMIS Desktop and all forms and installations of AppServers, are logically equivalent at the messaging level. Therefore, the DMIS architecture is *not strictly a client-server architecture, but rather a network architecture that is serving a hierarchical model*. The model itself is described in deployment descriptors and can be expanded to include peer relationships between entities as well as the existing tree.

4.2.2. Multiple Inbound and Outbound Connections

Every entity in the DMIS network, including both the DMIS Desktop and all AppServers, can (in theory) *uplink* to multiple other entities, create links to peer entities, and accept *downlinks* from any number of entities.

These relationships, as they apply to DMIS, can be configured through config files or using the DMIS Configuration utility that is part of this version of the middleware. The entry point for this utility is the *org.cmis.util.dmisconfig.DMISConfigFrame* class.



Connections to DMIS network
Enter connection points into DMIS network

Server Connection Sequences:

(DESKTOP)->EMBEDDED->LAN->CENTRAL Host Count: 2

General Settings:

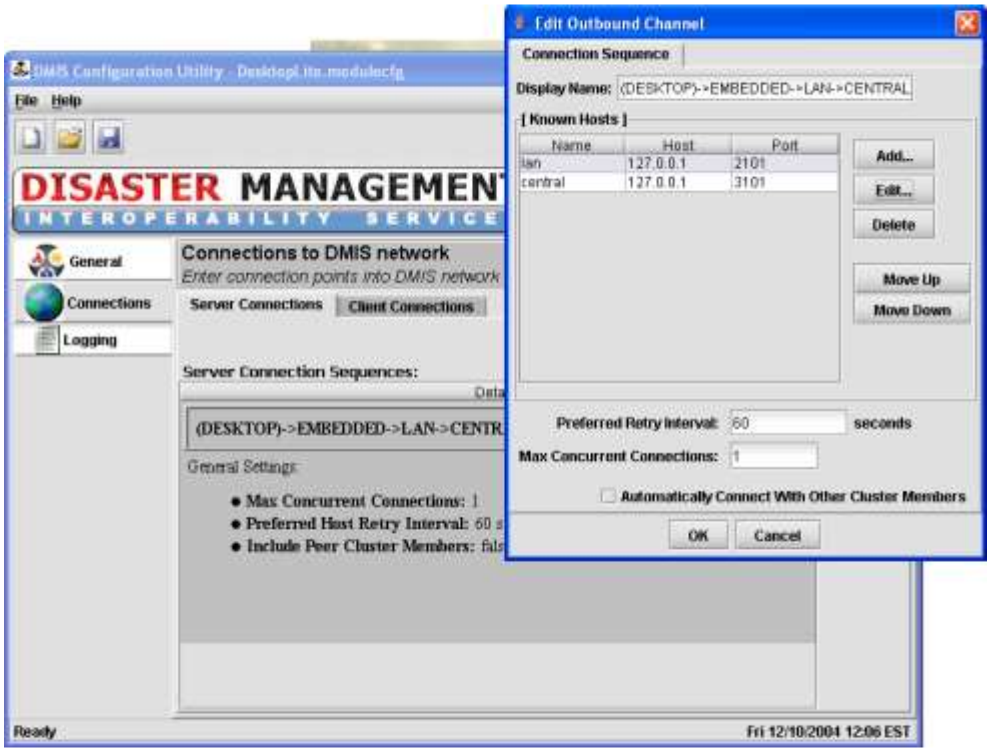
- Max Concurrent Connections: 1
- Preferred Host Retry Interval: 60 seconds
- Include Peer Cluster Members: false

Connection Sequence:

- lan: 127.0.0.1 3101
- central: 127.0.0.1 3101

Ready Fri 12/10/2004 12:06 EST

Figure 13: DMISConfig utility showing uplinks from a desktop installation



Connections to DMIS network

Enter connection points into DMIS network

Server Connection Sequences:

(DESKTOP)->EMBEDDED->LAN->CENTR

General Settings:

- **Max Concurrent Connections:** 1
- **Preferred Host Retry Interval:** 60 s
- **Include Peer Cluster Members:** fals

Display Name: (DESKTOP)->EMBEDDED->LAN->CENTRAL

[Known Hosts]

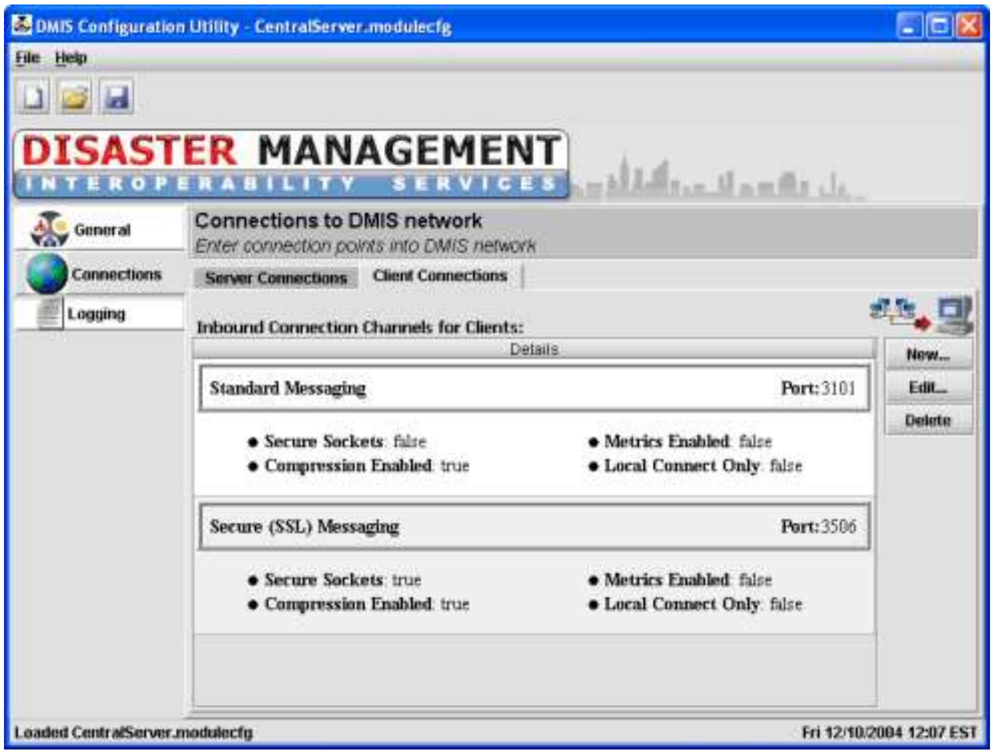
Name	Host	Port
lan	127.0.0.1	2101
central	127.0.0.1	3101

Preferred Retry Interval: 60 seconds

Max Concurrent Connections: 1

☐ Automatically Connect With Other Cluster Members

Figure 14: DMISConfig utility showing uplink details



Connections to DMIS network

Enter connection points into DMIS network

Inbound Connection Channels for Clients:

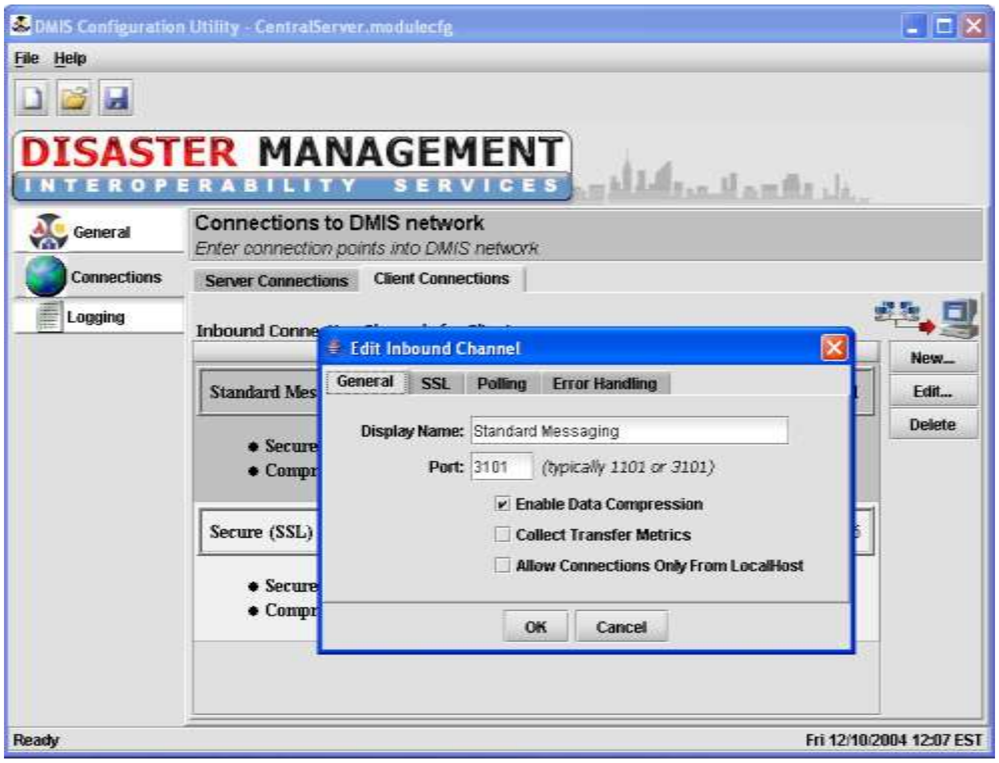
Standard Messaging Port:3101

- **Secure Sockets:** false
- **Compression Enabled:** true
- **Metrics Enabled:** false
- **Local Connect Only:** false

Secure (SSL) Messaging Port:3506

- **Secure Sockets:** true
- **Compression Enabled:** true
- **Metrics Enabled:** false
- **Local Connect Only:** false

Figure 15: DMISConfig utility showing inbound messaging channels



Display Name: Standard Messaging
Port: 3101 (typically 1101 or 3101)
☑ Enable Data Compression
☐ Collect Transfer Metrics
☐ Allow Connections Only From LocalHost

Figure 16: DMISConfig utility showing inbound messaging channel details

By default, the DMIS Desktop is configured to only perform uplinks to the nearest AppServer and not accept inbound connections – though more capability exists underneath.

4.2.3. Intelligent Connection Cycling

Entities on the DMIS network, including the DMIS Desktop, include support for intelligent connection cycling. Where previous versions allowed only one fixed desktop-to-server connection to be defined, it is now be possible to configure a set of *possible* connection points. Connections with another node (usually an AppServer) are made according to a priority-ordered list. When a connection is lost, the other defined points will be attempted in a cyclic fashion until an uplink can be established. The software can also continue trying a preferred host while a connection with a secondary host is used.

This connection cycling behavior is available not only at the DMIS desktop, but on servers as well. Therefore, a LAN server can define multiple connection points in order to make it more resilient. Connections from LAN server are generally made only to the CENTRAL Server, but technically they can also link directly to another LAN server (or another entity on the DMIS network).

4.3. Intelligent Message Routing

One of the most powerful new features of the DMIS middleware is its ability to perform *Intelligent Message Routing*. This capability is the engine behind much of the new distributed functionality in DMIS.

4.3.1. Symbolic Addressing via RoutingIDs

4.3.1.1. Flexibile in Addressing

Much of the flexibility in message routing comes from the use of symbolic IDs that are assigned to every entity on the DMIS network. These symbols allow one entity to communicate with another entity no matter how they are tied into the DMIS network. The entities that are communicating need not be directly connected; any path between them is sufficient for them to exchange messages. Connecting to any node on the network provides a path to other nodes on the network that are available indirectly.

4.3.1.2. RoutingID Format

A RoutingID is consists of a series of *dimensions* that serve to narrow down a specific operator or entity within the DMIS network. Dimensions are specified as a String containing a series of KEY=VALUE pairs, and using the '/' character as the delimiter between pairs. For example:

DIMENSION1=VALUE1 / DIMENSION2=VALUE2 / DIMENSION3=VALUE3

Dimensions are organized for clarity into four core areas:

- **ENTITY** – A specific operator (OP), organization (ORG), service (SVC), module (MOD), or function (FUNC). If not otherwise qualified, this refers to the nearest available match. For example, the RoutingID "SVC=TIE" may resolve to the nearest instance of the TIE service. A combination of OP and ORG keys refers to an operator without regard to where (or whether) the operator is connected.

- **LOG** – A logical location on the DMIS network, expressed using the ITYPE, TIER, DEPT, and (sometimes) the ORG dimensions. A value of "TIER=CENTRAL" refers to any AppServer in the CENTRAL server bank (and not a specific installation).

- **PHYS** – A physical DMIS installation, normally tied to a specific server or workstation via the MID or IID dimensions.

- **CTX** – An additional qualifier normally used to address a specific session (e.g. within the DMIS desktop) via the SID dimension. In some cases this category can hold other application-defined values.

The category names can also appear directly in a RoutingID string. For example, a RoutingID such as:

SVC=TIE / TIER = CENTRAL

is the equivalent of:

ENTITY[SVC=TIE] LOG[TIER=CENTRAL]

The RoutingID parser accepts either form, but the implementation always produces the verbose form when converting to display formats.

4.3.1.3. Possible Dimensions

While the messaging subsystem can handle many types of topologies in theory, RoutingIDs in DMIS are designed according to convention that reflects the model of a hierarchy. In DMIS, the following RoutingID keys and may be used in appropriate situations:

[ENTITY] Dimensions:

OP=*name*

The login name of a given operator. This name is only the UserID; the COG ID is not included in this portion because it is normally supplied using the ORG key described separately here.

ORG=*name*

The name of the organization the given entity is associated with. Though this value is arbitrary from the perspective of the messaging subsystem, it is an abstraction of the DMIS COG identifier. When combined with an NAME key, the RoutingID refers to a specific DMIS operator.

When combined with an ITYPE or TIER key instead of an OP, the RoutingID refers to an AppServer running in the part of the hierarchy owned by the corresponding COG.

DEPT=*name*

The name of the organization the given entity is associated with. Though this value is arbitrary from the perspective of the messaging subsystem, in DMIS it usually refers to the COG ID. When combined with a TYPE or TIER key, the RoutingID refers to an AppServer running in the part of the hierarchy owned by the corresponding COG. When combined with an NAME key, the RoutingID refers to a specific DMIS operator.

SVC=*name*

The symbolic name of a specific service (such as the TIE service) running on an AppServer, normally determined by the author of the service. This can be used to address a service directly and bypass other target resolution code, or may be used to locate an “abstract” instance of a given service (such as the nearest MAPPING service).

MOD=*name*

The symbolic name of a specific module running on an AppServer. Modules are functional subsystems that may be loaded when the AppServer boots up. The DMIS Service Manager, which manages service instances and distributes messages to them, is an example of a module. The clustering module is another example.

FUNC=*name*

A sub function of any given SVC or MOD. This dimension has specific, and somewhat arbitrary, meaning to code that uses it. The ThinRPC module, for example, uses **MOD=THINRPC / *FUNC=AUTH*** and **MOD=THINRPC / *FUNC=RPC*** to refer to itself when messages are generated for various specific operations.

[LOG] Dimensions:

ITYPE=[“ DESKTOP”, “APPSVR”, “MSGSVR”, or “THIN”]

A specific type of DMIS installation. DESKTOP refers to the context of a DMIS workstation. APPSVR refers to a DMIS AppServer of any kind. MSGSVR refers to an AppServer that is running no services and is acting as a messaging hub only (this setting is currently unused). THIN refers to a web client or a called using the *ThinRPC* module.

TIER=[“EMBEDDED”, “INTERMEDIATE”, or “CENTRAL”]

The logical tier in the hierarchy on which an AppServer resides. The CENTRAL AppServer is the tier of AppServers, operated by the DMIS group itself, serving in the capacity of the central hub. The EMBEDDED AppServer run at the workstation level (and is thus the equivalent of the “LSM” in previous versions of DMIS). INTERMEDIATE AppServers are any servers in between EMBEDDED and CENTRAL AppServers in the logical hierarchy. INTERMEDIATE AppServers are isolated and addressed using additional RoutingID dimensions.

[PHYS] Dimensions:

IID=*id*

Refers to a specific DMIS installation on a specific machine. This dimension is most commonly used as a common “pivot” between a DESKTOP instance and an

EMBEDDED AppServer, which normally share an IID value since they run from the same DMIS installation.

MID=*id*

Refers to a specific physical machine that has a role on the DMIS network. This can be used to route messages to DMIS installations on a given machine.

[CTX] Dimensions:

SID=*id*

The IID key refers to a specific DMIS installation on a specific machine. This can be used to route messages to a DMIS installation regardless of its other parameters.

Other Dimensions:

DESC=*name*

This is an arbitrary dimension used to describe the purpose of the RoutingID or to provide human-readable information. It does not affect the *logical value* of the RoutingID, meaning the two RoutingIDs that are otherwise identical will return TRUE when compared using their `equals()` methods.

These key/value pairs are combined to form a complete RoutingID that DMIS recognizes and the messaging subsystem can route effectively. The following table shows typical RoutingIDs that might be encountered for entities within a mythical COG with ID=100:

RoutingID	Interpretation
LOG[TIER=CENTRAL]	Any server on the CENTRAL Server bank
LOG[TIER=INTERMEDIATE / ORG=100]	LAN-based AppServer for COG=100
ENTITY[SVC=TIE] LOG[TIER=INTERMEDIATE/ORG=100]	TIE service running on LAN server for COG=100
ENTITY[OP=user1 / ORG=100]	Operator ‘user1’ in COG=100
PHYS[IID=4896686]	A specific DMIS installation
ENTITY [OP=user1 / ORG=100] PHYS[IID=4896686]	Operator ‘user1’ in COG=100 when logged into a specific DMIS installation
ENTITY[SVC=TIE]	Any known instance of the TIE service

RoutingIDs are fed into *message headers* as applications and services generate messages for consumption by other entities. The SourceID and DestinationID fields of the *Message* class designation where a given message originated and/or where it is headed, respectively. Both fields are specified in the form of RoutingIDs. When message is *sent*, the messaging subsystem determines how to best get the message from point to point based on current conditions, or queues the message accordingly until a path can be resolved.

4.3.1.4. RoutingID Assignment

An operator, AppServer, other entity normally formulates and declares its own identity, passing it to the messaging layout – which in turn notifies other entities within the network that the new entity is addressable.

4.3.2. Message Routing Strategies

In order to route messages to their destinations according to RoutingIDs, each entity in the network must have some knowledge of the network layout. The messaging subsystem operates using a peer-oriented approach and determines the most effective path between nodes.

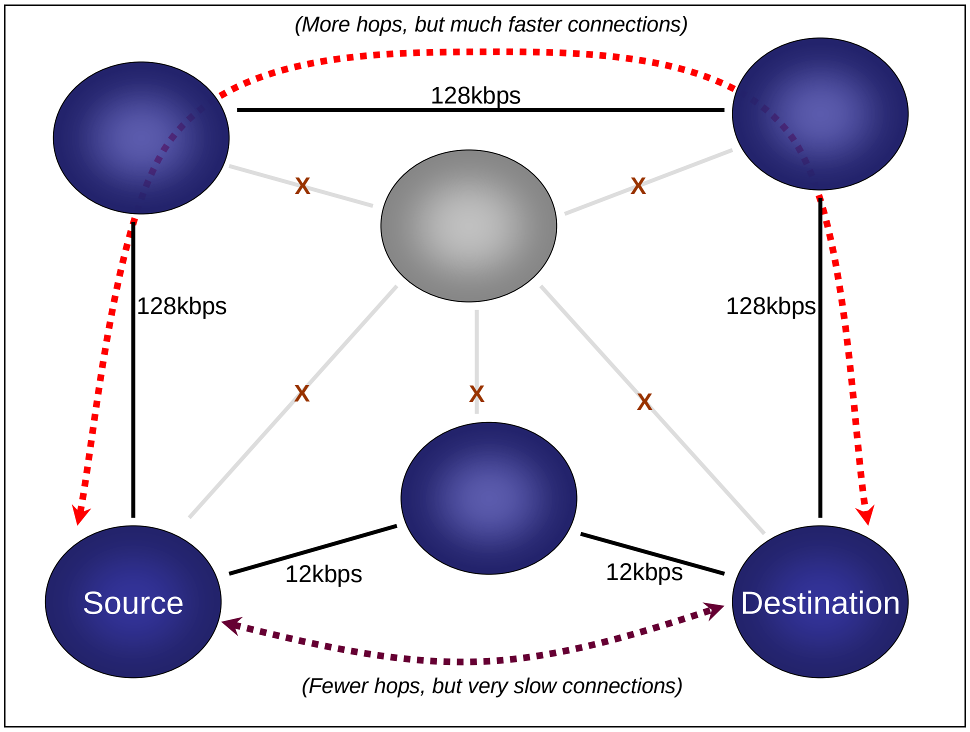

(More hops, but much faster connections)
128kbps
X
X
128kbps
128kbps
X
X
X
12kbps
12kbps
Source
Destination
(Fewer hops, but very slow connections)

Figure 17: Intelligent message routing within a network

Routing preferences are based on a set of criteria that are considered in order of relative importance:

1. Whether all nodes in the route are currently connected,
2. The minimum link speed between any two nodes in the proposed path,
3. The average link speed between nodes in the proposed path, and
4. The shortest path between source and destination.

The priority is essentially (1) delivering the messaging, followed by (2) delivering it over the fastest route. In some cases, such as when the DMIS desktop is offline, the best route cannot be determined because there are no points of entry into the DMIS network. In this case, the computation is *deferred* until such resolution is possible (usually when the node is reconnected to the network).

Messages remain at the *source* until a full, clear path exists between *source* and *destination*. That is, messages do not travel to possible intermediaries until a usable route is determined and those nodes are part of the proposed path. This generally produces the most efficient pathway, reduces the time that messages are spent in the hands of other parties, and prevents messages from undue "bouncing" around the network before their final delivery. Once the message reaches an intermediate entity, though, that intermediate node may decide to alter the proposed route based upon changing conditions (e.g. a link has been severed since the original node attempted to route the message).

4.3.3. Routing Tables

Each node in the DMIS network stores information regarding the current layout and state of the network in its own private *routing table*. This routing table is the network topology as it is perceived by that entity, in addition to the current set of connections within that grid.

The routing table is updated dynamically as activity occurs on the DMIS network such as connections, disconnections, and new nodes coming online. This gives each node a near-real-time view of the world. The routing updates are sent in a word-of-mouth style from node to node throughout the network until they reach all relevant nodes. The transmission logic for these updates is integrated with the overall messaging protocol into a single, unified information exchange protocol.

Because the amount of data to be captured for the entire DMIS network could potentially be huge, filters are applied so that sub networks (such as that used in a LAN deployment) can only have direct knowledge of entities on the logical "continent" in which they live. Only the CENTRAL Server bank has a view of the entire network at once, communicating small pieces to subnets as appropriate for them.

The following diagram depicts a routing table held by the CENTRAL Server, which has a view of the entire network. In this example, the network is small and consists of two COGs (ID=1000 and ID=2000) plus a smattering of direct connections to the CENTRAL Server:

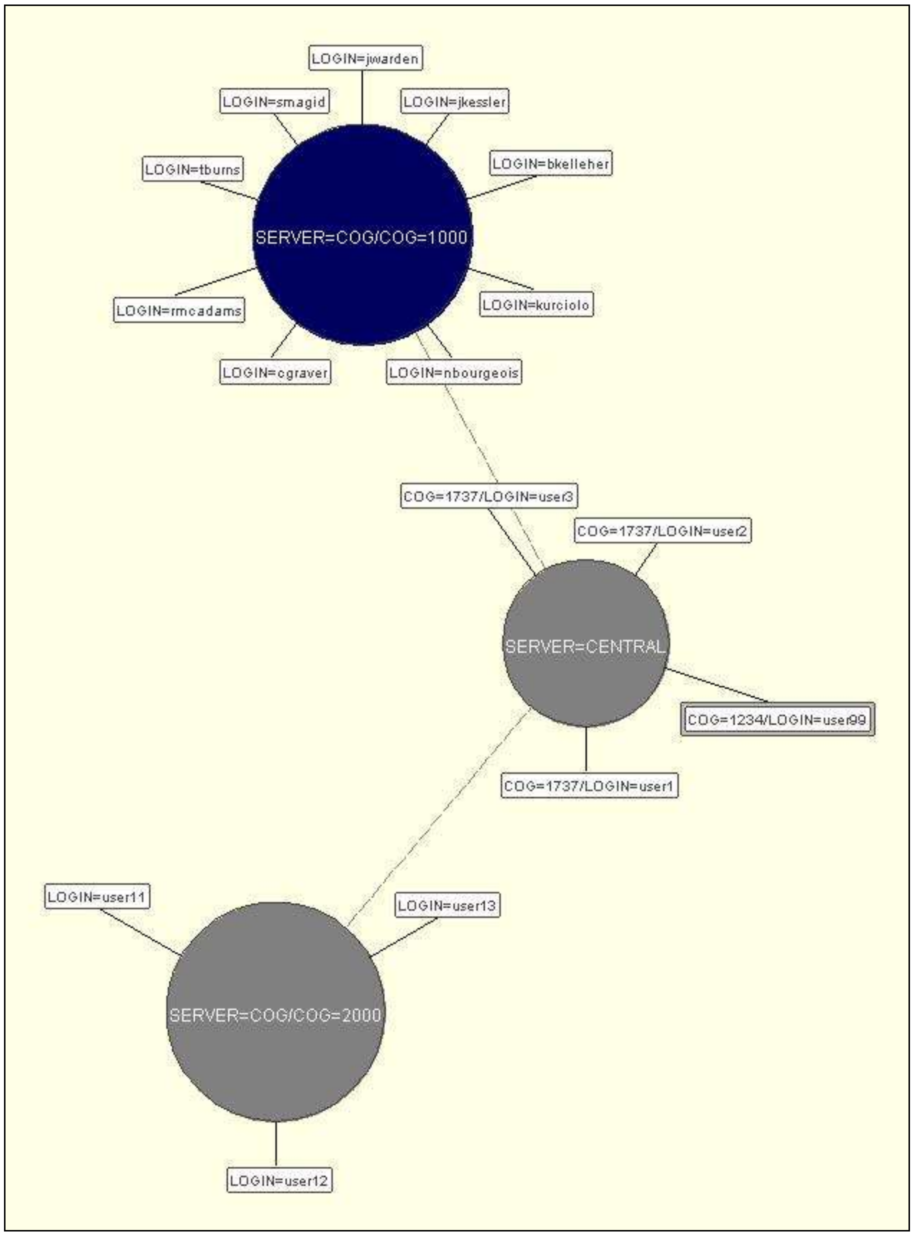


Figure 18: Visualization of routing table data

Note that this is the *logical* layout of the network. Because the messaging layer has intelligent routing, the *physical* connection points may be different.

The ServerDiag utility, whose entry point is *org.cmis.util.diagnostic.DiagnosticFrame*, can generate diagrams like this one in real-time as network conditions change.

4.3.4. Logical vs. Physical Networking

The dynamic nature of routing in the distributed architecture allows *physical* and *logical* connection points to be different. The logical layout is described by the content of RoutingIDs, while physical connections are defined as TCP/IP connections to particular AppServers or other nodes on the network. As long a path exists, direct or indirect, between nodes that are trying to communicate (each with logical RoutingIDs), efficient routing across various physical connection layouts is automatic.

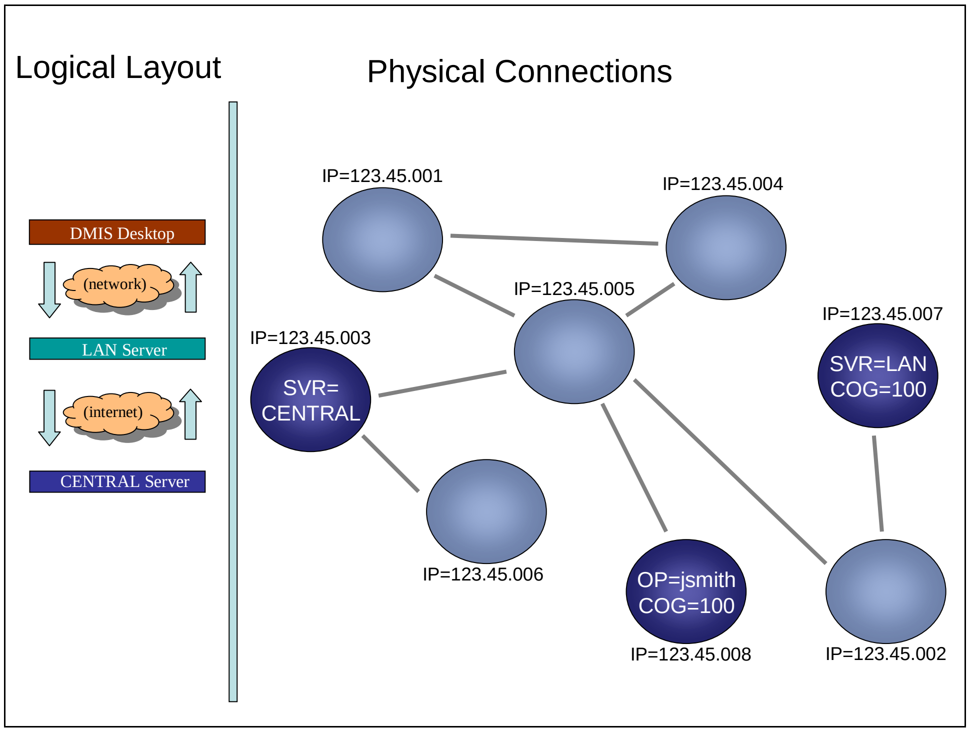


Figure 19: Logical layout vs. physical connections

A side-effect of this feature is that one Desktop could, in theory, connect to the DMIS network just by pointing to another Desktop that is already connected. In this case, the already-connected Desktop serves as an access point for the one that needs a connection. The same is true at the AppServer level. Indeed, any DMIS entity can act as a router to any other DMIS entity assuming that the network of physical connections provides a routing path from one logical place to another.

4.3.5. Efficient Topologies

Assuming equal connection speeds, message routing is the most efficient when there are as few "hops" as possible between the source and the destination of a given message. Therefore, from an efficiency standpoint, physical connections are best defined with the logical topology in mind. Although routing *will work well* in a variety of interconnection scenarios, the bulk of traffic between any two nodes should pass through as few intermediaries as possible to reduce potential latencies.

In practice, the DMIS Desktop will usually connect directly to the corresponding LAN server or the CENTRAL Server if there is no COG-level server. LAN servers also will tend to communicate directly with the CENTRAL Server. Though permutations are possible, this default layout for DMIS is generally efficient.

Note that any entity, such as a DMIS Desktop installation, could have direct connections to multiple AppServers. It may, for example, have connections defined to both its local LAN server *and* the CENTRAL Server, meaning that there may be multiple paths from the Desktop to other entities. In this case the routing code will choose the most appropriate routes based upon context, using the extra paths for enhanced efficiency in some cases – and possibly for enhanced redundancy as well.