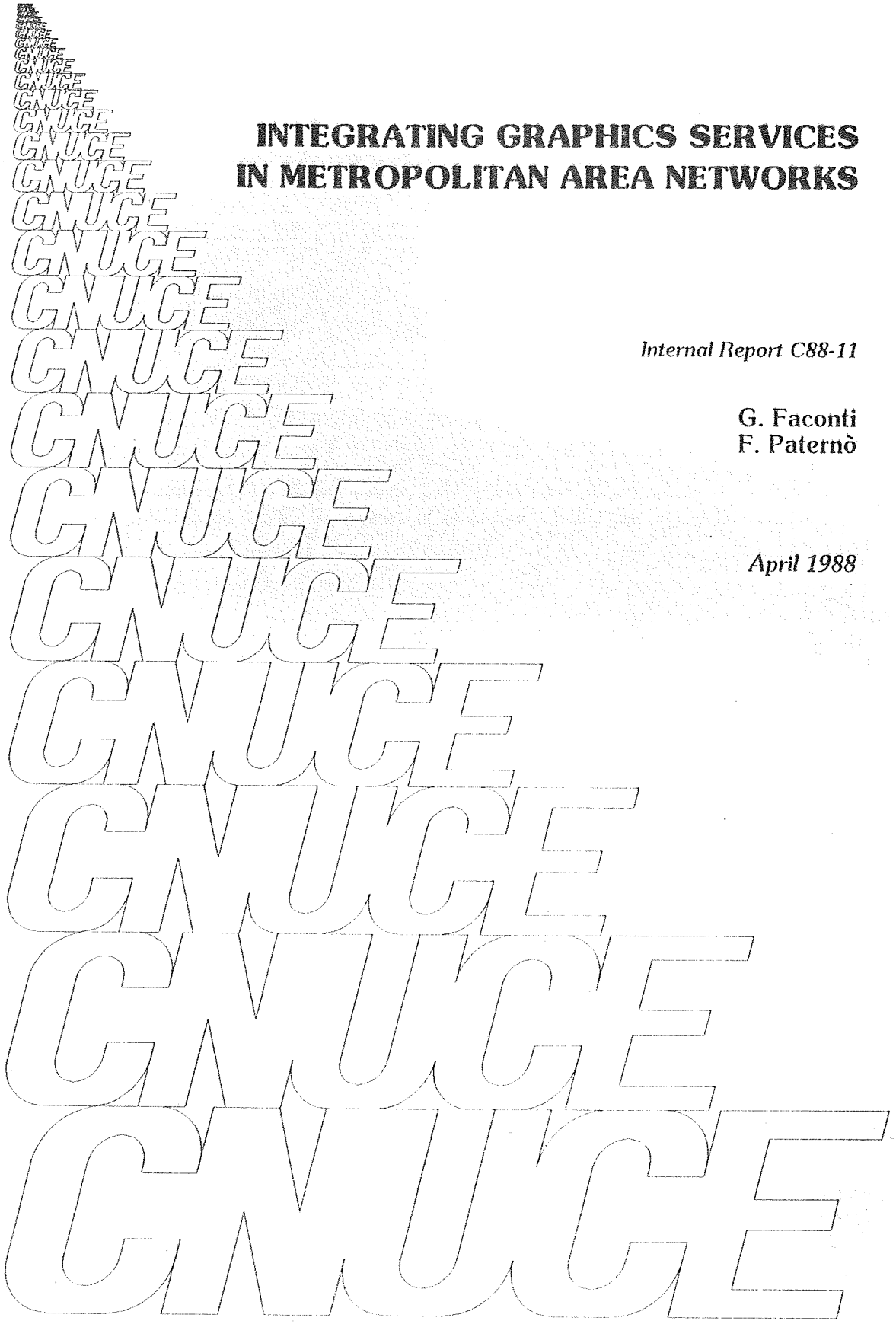
INTEGRATING GRAPHICS SERVICES IN METROPOLITAN AREA NETWORKS



Internal Report C88-11

G. Faconti
F. Paternò

April 1988

Integrating Graphics Services in Metropolitan Area Networks

G.Faconti, F.Paterno'

C.N.R., Istituto CNUCE, Via S.Maria 36, 56100 Pisa, Italy

ABSTRACT

This paper presents the key issues and the results which have been stressed during a prototype implementation of a distributed graphics system suitable for use in a scientific and technical environment. The primary objective of the system is to provide the user with flexible tools to develop graphics applications on interactive bases with minimal programming. This goal is achieved through a greatly improved computing environment implemented over a metropolitan area network and integrating professional workstations, graphics server machines, and large mainframes. The professional workstations, which run the Unix operating system, provide powerful user interfaces and object-oriented graphics.The server machines allow for the sharing of graphical resources, while the mainframes perform the heavy computational tasks. The system allows for the dynamic acquisition and release of any resource in the network through interactive, menu-driven commands.

Introduction

A join project between CNR-Istituto CNUCE and Olivetti Italia is being under development in order to define a prototype implementation of a metropolitan area network interconnecting several research institutes and university faculties.

A major goal of the project is to provide for integrated graphics services available throughout the network and accessible with minimal programming from any workstation.

Within this framework, the project aims to achieve a major advance in user productivity through a greatly improved computing environment made of networked, multiprogrammed professional workstations that may also access the processing capabilities of a powerful mainframe computer.

To achieve this goal three major components have been considered: a high speed metropolitan area network, a distributed graphics system, and an object-oriented programming language.

This paper is mainly focusing on the architecture of the distributed graphics system and on a set of tools provided on top of the system itself.

The Metropolitan Area Network

Recent technological advances make it possible to conveniently connect most computers to each other through *local area networks* (LANs). The performance of LANs and suitable software packages often lead to the definition of *distributed systems*, whose advantages over the collection of single computers are well known.

Stressing this concept, it is conceivable to build a network interconnecting computers in a wider area, such as a town, than actually allowed by LANs by using the same technology and components.

With this goal in mind, requirements are quite obvious and can be summarized saying that the resulting *metropolitan area network* (MAN) must behave as far as possible like a LAN, even spanning over longer distances and even interconnecting LANs.

The Architecture of MAN

The above requirements suggest the overall architecture of the MAN to be based on a two level structure made by a metropolitan backbone network, to which single LANs are connected (figure 1).

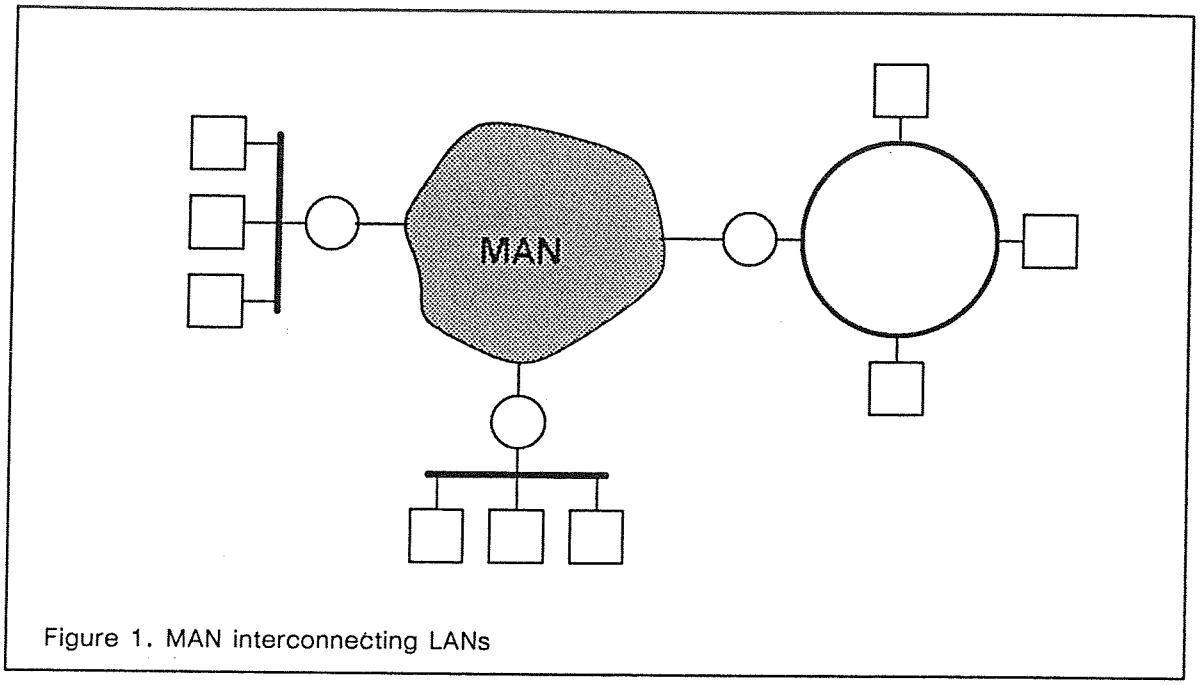


Figure 1. MAN interconnecting LANs

Communications across the boundary LAN/MAN are managed by suitable *bridges* or *gateways*; every host on each connected LAN is then automatically in MAN.

By this structure, already existing LANs can remain unchanged and the MAN is not choked by local traffic. Moreover a single connection on the MAN can be used by many hosts, thus reducing costs of connection and increasing the total number of possibly connected hosts.

Because of its physical structure, this MAN has been called MIN, standing for Metropolitan Interconnection Network [1].

For the sake of reliability, simplicity and feasibility in the target environment, MIN is a BUS network (figure 2) and its physical specifications, bit rate, packet format and addressing structure comply with

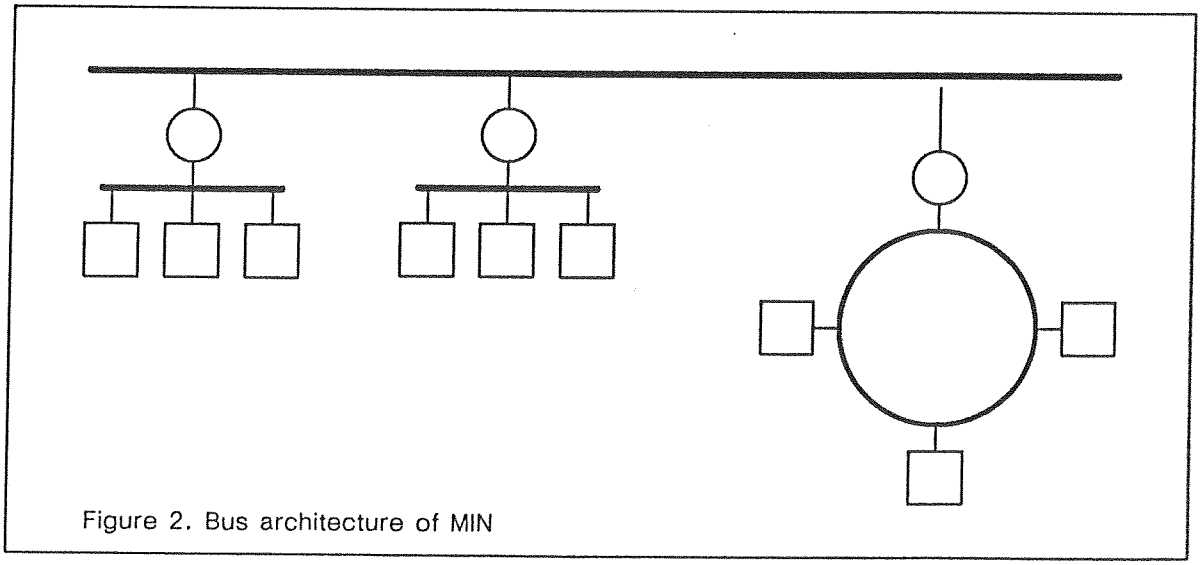


Figure 2. Bus architecture of MIN

Ethernet [2] specifications. However, a different channel access protocol is used to guarantee good performances even when operating on longer distances than supported by the CSMA/CD protocol [3, 4].

Network communication and service protocols

The TCP/IP communication protocols are adopted by the MIN because, even if they are not official standards, they are adopted by the *internet model* defined by the *Defence Advanced Research Projects Agency* (DARPA) [5] and thus their implementations are widely available in most computers.

The distribution software package usually includes services such as: *telnet* (virtual terminal), *ftp* (file transfer), and *electronic mail*.

On the contrary, work is done to extend services. In particular a *remote procedure call* (RPC) mechanism has been built on top of TCP/IP.

The International Standards Organization (ISO) has defined a layered model, called the *Open Systems Interconnection model* (OSI) [6], which is actually the International Standard for the exchange of data among systems.

As the components of the DARPA Internet Model are easily related to the OSI Layers, a strong concern exist to progressively migrate from the TCP/IP to the OSI protocols.

This migration will allow a natural implementation on LANs, and thus on MIN, of the services defined for use by the *Application Layer* of OSI.

The Distributed Graphics System

The Process Based Model of Graphics System

The graphics environment is modeled upon the concepts of set of functionalities and processes within a set implementing totally or partially these functionalities.

Seven set of functionalities have been identified as follows:

- *Control Functions*
- *Communication Functions*
- *Application Functions*
- *Viewing Functions*
- *Multiple Viewing Support Functions*
- *Interaction Functions*
- *Resource Sharing Functions*

Unlike many systems modeled upon the concept of layering, our concept of set of functionalities does not imply any hierarchical ordering of processes within a specific instance of the system. Rather instances of the system are build by applying to processes an ordering function which is characteristic to the application problem.

Figure 3 shows a traditional configuration of graphics system made by composition of processes. An application process (APn) is linked with a multiple viewing process (MVP1) which allows for multiple viewing (i.e. the concept of multiple workstations). MVP1 is linked to two viewing processes (VP1 and VP2) which provide output capabilities and to an interaction process (IP1) which provides for input capabilities. VP1 and IP1 are also linked together to implement a workstation of category OUTIN.

The capabilities of the process based model can be fully implemented over a high speed computer network interconnecting nodes of different capabilities which can host one or more specialized processes.

Previous investigations on distributed graphics computing focused their attention to the access to remote resources [7, 8, 9]. Here, the problem is to define a set of operations which can be autonomously processed in a remote system whose access is granted by consolidated network services as the handling of messages and the transfer of files.

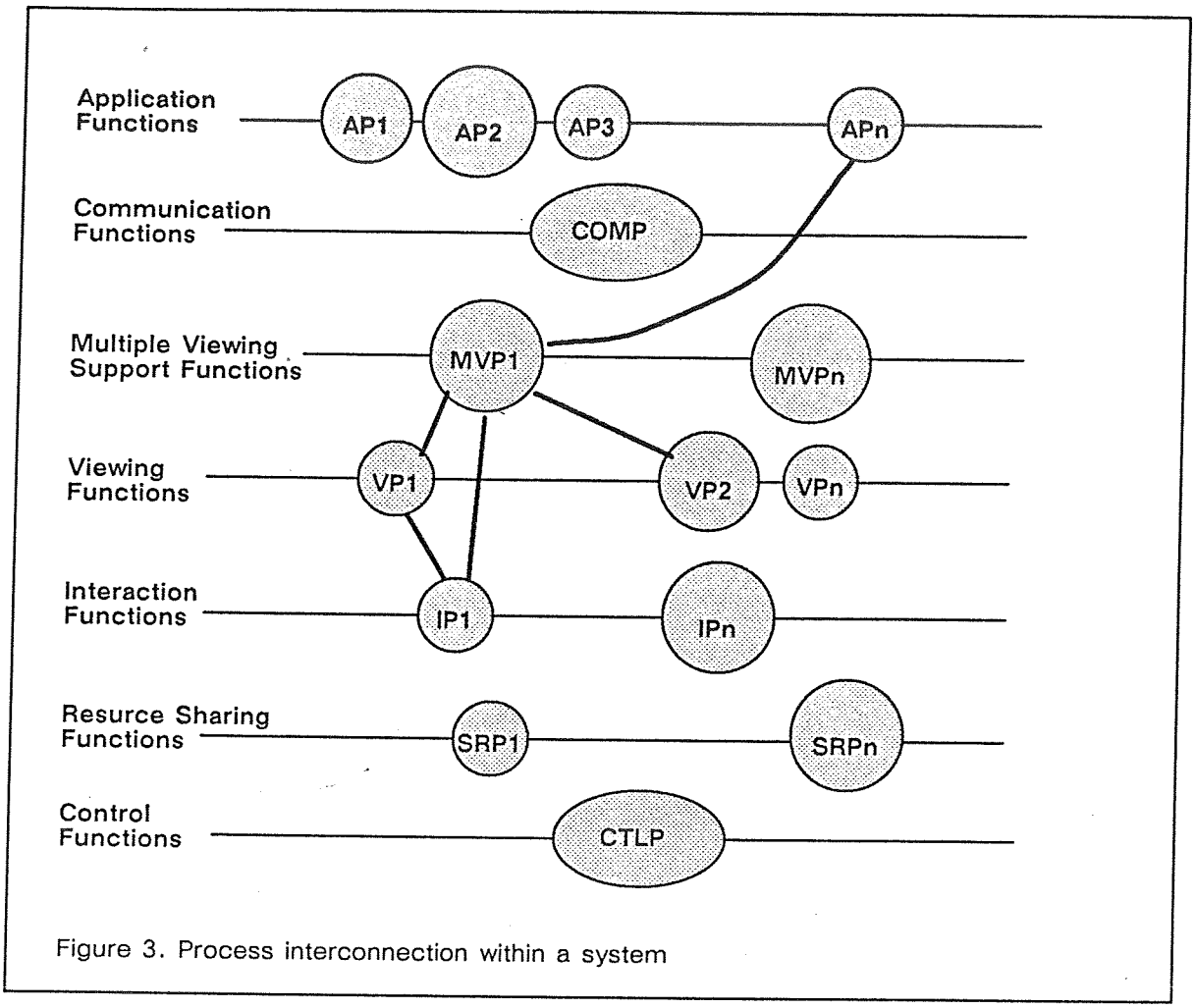


Figure 3. Process interconnection within a system

Actually available technologies, as the local and metropolitan area networks with a bandwidth starting from 10 megabit per seconds, allow a different approach to distributed computing based on the integration of services in a distributed system; that is processes running on different computers become part of a whole integrated system.

The Graphics Resource Manager

The key element which allows to implement a distributed graphics system is a special process implementing the control functions which is able to manage the graphics resources of a specific node and to cooperate with peer entities in order to maintain the consistency of the system across the network.

In principle, any node in the network is candidate to become part of the distributed graphics system. This actually takes place when the node is running a control process, called *graphics resource manager,* which enables graphics processes in this node to be *called* from graphics processes in other nodes.

An instance of graphics resource manager is created either implicitly at boot time or explicitly by user's command. Conversely a node is excluded from the system either at shutdown or by an explicit user's command.

Among the tasks accomplished by the graphics resource manager are the maintenance of the configuration of the graphics system, the notification to local active processes of any event (deletion, error conditions, etc.) occurred in a partner process, and the creation and destruction of processes within a node upon request.

In accomplishing those activities the graphics resource manager is able to dynamically catalogue the actual configuration of graphics resources in the system and to find the nodes required to perform specific services.

The graphics resource manager may create processes either autonomously or on explicit request of some other processes in the system.

Autonomously created processes are generally self-contained processing units and normally exist in a *passive* state. A *passive process* is *activated* when it becomes part of a *communication association*, that is it becomes a partner of another process in the system. On termination of the communication association an active process may be either *passivated* (so that it may respond to further requests), or *destroyed* (as no longer needed), or *locked* (in a way that it may only become part of a subsequent communication association started by a specific process).

Processes created on explicit request are also implicitly *activated* and generally do not survive to a communication association (that is they are *destroyed* on termination of the communication association).

This simple schema of operation allows for the handling of server and shared processes as well as for the dynamic configuration of specific instances of a graphics system.

Communication Processes

Conceptually any two processes in the system may communicate through the exchange of messages. Consequently, a *common service element* is to be identified to provide a uniform access method to the network from all the processes. This service has been identified with the *remote operation service* (ROS) [10] of OSI and has been mapped to the RPC mechanism built on top of TCP/IP.

The RPC service has been chosen as it is applies to a general class of problems and it can be easily extended to support different semantics. In fact, the communication service provider needs only to have some knowledge of the functional nature of the interface it is servicing, but not the semantics of a particular function.

All graphics processes in the system have automatically linked with them an *RPC service provider* when they are created. This enable interprocess communication within a single computer as well as between different computers through a network server process which is in our implementation the *tcp server process*.

Application Processes

An application process is the concrete representation of an application. They are created by user's programming and include the implementation of algorithms for a specific application including modeling functionalities on which no assumptions have been made.

An application processes, called *system monitor* [11] has been developed, which help the user to navigate through the system, and to relieve it from the burdens of system programming. It is a high level tools used to manage and to interact with both the distributed system as a whole and a particular process within an instance of the graphics system.

The System Monitor

The system monitor is composed of a highly interactive user interface and of an RPC runtime support to request services to the graphics resource manager.

The basic set of services available to the user includes:

- *enable/disable* of a node for graphics processing; that is the creation and deletion of an instance of graphics resource manager for this node
- *inquire* of the graphics processes actually running on the network or on a specific node
- *create/destroy* of graphics processes in the network possibly with associated access control logic.

Association of processes in the network to become an instance of graphics system may either be determined automatically by the system using an optimization algorithm or explicitly indicated by the user. The first choice belongs to servers sharing resources among users, while the second one is normally used to address proprietary devices.

Viewing Processes

Viewing processes are the means by which an application process may actually obtain the display of a picture on a display surface.

Only one class of workstation processes, implementing an extended concept of the GKS workstation, is actually available on the system. However, different semantics may be in principle be defined.

Workstations, or viewing processes, differ from the GKS concept, in that they are dynamically configurable. The state list is not filled with values found in a description table; rather, the creation of a viewing process involves a negotiation mechanism in order to determine what capabilities are actually needed by the client process. This includes, as an example, the display surface size and the color look-up table.

As the resources available to a viewing process are time dependent, a client processes may select different strategies of negotiation. This does not apply to server processes whose capabilities are uniquely defined at creation time. When a server process, which is a *passive process*, is *activated* there may not exist negotiation at all.

Another major change with respect to GKS, is that a workstation may exist independently from an application program or a graphics kernel and that it may continue to exist in the system either as *passive* or *locked process*, after it has been *closed*, . Application processes main gain access to a viewing process which already existed in the system, and do not need any kernel support, which is necessary only in the case of multiple viewing.

Multiple Viewing Support Processes

Multiple viewing support processes (MVP) provide an application process with the capability of handling multiple viewing processes at the same time. When an application process establishes a communication association with an MVP, an implicit hierarchy is established between processes so that any subsequent access to viewing processes is handled through this MVP. Inversely a communication association between an application process and a viewing process inhibits any use of an MVP for the time it is being active.

Possible conflicts are handled by the graphics resource manager which is the only process owning the rights to create and destroy processes, and to establish communication associations between them.

Within this framework, accesses to workstations from applications are automatically routed through the MVP, when it exists, while accessing an MVP produces an error condition when an association between an application and a workstation is underway.

Actually only one multiple viewing support process is being implemented which is based on the concept of graphics kernel implied by GKS.

Interaction Processes

Interaction processes are responsible for the handling of event generated by an operator from an input device. They may be linked with viewing processes in order to build workstations of category OUTIN.

The model of interaction is based on the asynchronous (event) input mode described in the GKS standard and it is related only with *locator, stroke,* and *pick* logical devices. *Valuator, string*, and *choice* logical devices are actually simulated outside of the graphics system by using respectively basic I/O routines and pop-up menus where applicable.

Interaction processes maintain their own queue of events locally and distribute it on request by clients. This actually may cause erroneous ordering of events when multiple input is allowed from different workstations. This unwanted situation is actually acceptable as no application of this kind are being considered.

At the actual state of the project, however, a complete a consistent input model has not yet being developed and it will certainly be one of the major tasks we will be dealing in the near future together with the integration of resource sharing processes.

Resource Sharing Processes

Resource sharing processes coincide actually with the window manager systems [12, 13, 14] available on the workstations we are using within the testbed environment.

In the actual configuration, viewing and interaction processes are built on top of the window managers. However, we believe that a closer integration of those components is mandatory.

The idea is to develop an extension, at the Xlib interface level of X Window, to implement a GKS-like workstation directly within the window manager.

The testbed environment

The testbed environment used to develop the MIN project and its related services including graphics, is made of two Ethernet LANs interconnected by a proprietary bridge.

Available nodes include an IBM mainframe running VM/370 and several workstations running different version of the Unix operating system as it appears from figure 4.

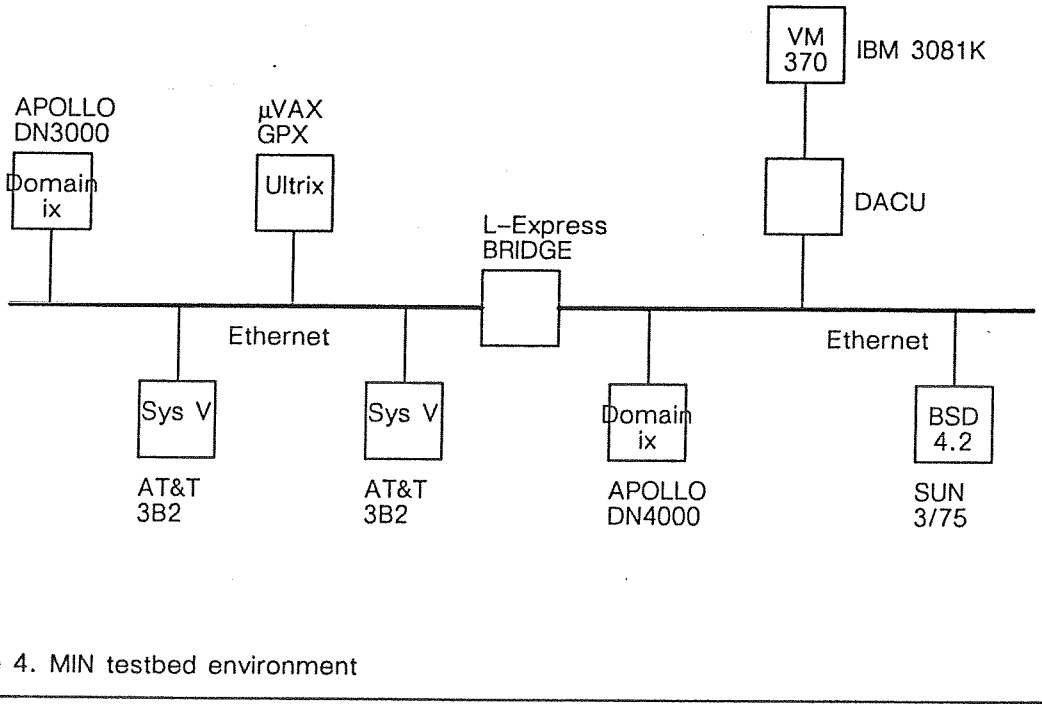


Figure 4. MIN testbed environment

Conclusions

The Metropolitan Interconnection Network Project is underway at CNR-Istituto CNUCE and Olivetti Italia.

At present a MIN prototype is under development; it reaches CNUCE, the Institute of Electronics and Telecommunications of the University of Pisa and the CNR-Institute for Computational Linguistic. In each of these locations, a set of standard services including the virtual terminal, the file transfer, and the electronic mailing is provided. Recently a GKS-based version of a distributed graphics system has been added.

Within the CNUCE testbed environment the extensions of the distributed graphics system discussed in this paper is being developed with the purpose of providing the users with a more advanced and flexible environment accessible through high level tools with minimal programming.

References

[1] P. Coltelli, M. Mannocci, F. Tarini, P. Zini, **A LAN approach to MANs**, Proceedings of EFOC/LAN.87, Basel, June 1987

[2] R.M. Metcalfe, D.R.Boggs, **Ethernet: Distributed Packed Switching for Local Computer Networks**, *Communications of ACM*, Vol. 19, N. 7

[3] F.Borgonovo, L.Fratta, F.Tarini, P.Zini, **L-EXPRESS NET: the communication subnetwork for C-net Project**, *IEEE Transactions on Communications*, July 1985.

[4] L. Rizzo, F. Tarini, P. Zini, **High Performance Bridge for Ethernet-like Networks**, Proceedings of EFOC/LAN.87, Basel, June 1987

[5] V.G.Cerf, R.E.Khan, **A Protocol for Packet Network Interconnection**, *Transaction on Comm.*, Vol. COM-22, N. 5

[6] ISO/IS 7498, **Open Systems Interconnection - Basic Reference Model**

[7] J.Bechlars, **Graphics Workstations and GKS in Networks**, CAMP'85, September 1985.

[8] G.Faconti, R.Bettarini, L.Moltedo, **Extending GKS to a Distributed Environment**, Proceedings of Eurographics'85, September 1985.

[9] Chr.Egelhaaf, G.Schurmann, **GOCS - The GKS-oriented Communication System**, Proceedings of Eurographics'87, August 1987.

[10] Information Processing Systems, **Message Oriented Text Interchange Systems - Remote Operation Service** - Parts 1, 2, ISO/DP 9040

[11] G.Faconti, C.Giuliano, F.Paterno', **An Interactive Approach to Graphics Systems**, CNUCE Internal Report, C1-88

[12] SUN Microsystems Inc., *Programmer's Reference Manual for the SUN Window System*,

[13] Apollo Computer Inc., *DOMAIN System User's Guide*.

[14] R.W. Scheifler, *X Window System Protocol - Version 11*.