DGDBM: PROGRAMMING SUPPORT FOR DISTRIBUTED TRANSACTIONS OVER DIRECT ACCESS FILES

*María Consuelo FRANKY*

Eliana RANGEL

Depto. de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976
Santafé de Bogotá - COLOMBIA
Fax: 57.1.284.18.90
(e-mail: cfranky@uniandes.edu.co)

**ABSTRACT:**

This paper describes a set of facilities for programming distributed transactions over direct access files which are accessed by primary key. The files are located on several computers communicated by a network. Each site has the set of GNU dbm (Gdbm) routines for local file management [Nelson 93]. Above this platform we have built an interface and a set of services for distributed transaction programming. The resultant programming environment, "DGDBM", offers transparency in relation to data distribution, giving a centralized vision to the programmer. It assures the functions of management of distributed transactions like as failure recovery and concurrency control. DGDBM is an useful support for distributed application programming over direct access files in UNIX networks and it is available as an API (application programming interface) for the C programmer. This paper describes the services offered by DGDBM to the programmer, the architecture of the system, the adopted solutions for distributed transaction management, the general aspects of design and implementation and the perspectives and planned extensions for this project.

**KEY WORDS:** Distributed Systems. Distributed file systems. Distributed data bases. Distributed Transaction Systems. Tools for the C programming of distributed transactions.

1. INTRODUCTION

Nowadays it is unquestionable that communication networks represent an advance which is becoming deeply rooted in the different environments of organizations. The establishment of national and international networks promotes the development of distributed applications, mainly in organizations with an evident distributed nature having geographically spread out departments and sections which require, in order to function properly, the combination of their different data files [Umar 93].

However, the programming of distributed applications over a set of data files located on several computers, that are communicated by a network, may be a very complex task without the proper aids. In the DGDBM project [Rangel 94] we have elaborated a library of functions that the programmer may include and invoke in his application in order
to obtain transparency to the distribution of data files, resulting in a programming effort similar to the one required in a single centralized file system. The resulting programming environment,"DGDBM", takes advantage of facilities offered by the GNU dbm [Nelson 93] routines for direct access file management: in the project we suppose a communication network where each site has the UNIX system together with the GNU dbm routines.

Moreover, DGDBM assures the functions of management of distributed transactions over multiple files that are located in several computers of the network: the programmer may invoke in his application some special functions to delimit the beginning and the ending of a distributed transaction which includes several requests for reads and writes on a set of files. DGDBM manages the simultaneous execution of multiple distributed transactions that are requested in different sites, assuring their atomicity and recovery in the presence of failures and their control concurrence in the access to common data (including the treatment of distributed deadlocks [Mullender 89]).

DGDBM is available as an API 1 for the C programmer with an interface very similar to that offered by the GNU dbm routines [Nelson 93] for direct access files which are accessed by primary key. It constitutes an useful support for distributed application programming, specially in contexts where the use of distributed databases management systems is inadequate or very costly: in fact, DGDBM uses in its implementation free software tools like the GNU dbm routines and its final version will run on LINUX system[Linux 94] which is a free version of UNIX.

The implementation of DGDBM is based in the use of another API called JOYCE+[Franky 90]: JOYCE+ gives a set of services for programming distributed systems following a model that is independent of specific network protocols. Also, we have applied some technical solutions for management of distributed transactions, that were implemented in the PERSEO project [Franky 92].

The following Section 2 presents the interface of services that is offered by DGDBM to the programmer of distributed applications over multiple files which are located at several sites. Section 3 shows the architecture of the DGDBM system in terms of different classes of processes, with their related modules in charge of executing distributed transactions that are requested by the applications. Section 4 describes the adopted solutions for distributed transaction management: these solutions determine the modules which coordinate and execute distributed transactions. The Section 5 describes the design and implementation of DGDBM in terms of the JOYCE+ model and the structure of communications between the processes of the system. Lastly, section 6 outlines the perspectives and future stages of the DGDBM project.

2. SERVICES INTERFACE OFFERED TO THE PROGRAMMER OF DISTRIBUTED TRANSACTIONS OVER DIRECT ACCESS FILES.

The programmer of distributed applications over a set of direct access files finds in DGDBM a library of C functions which offer transparency in relation to file distribution and to distributed transaction management.

The suppositions made by DGDBM that must be known by the programmer are the following:

a) Each referenced file by the user application is located at a single site of the network.

b) The application always refers to a file through his name and there is an APPLICATION CATALOGUE, replicated over all sites, that contains the association of each name file to the logical site where it is located. It is also supposed that a name file is unique in the network.

1 Application Programming Interface
c) There is a SITES CATALOGUE, replicated over all sites, that contains the association of logical sites (numbered as 0, 1, 2, ...) to physical sites (i.e. network nodes).

d) The structure of a GNU dbm file register is composed of a primary key and a contents: these two components are character strings of any length.

e) The access to a file register is through its primary key. There is also the option of accessing in a sequential manner all the registers of a file in order of physical storage: these are characteristics derived of the GNU dbm routines [Nelson 92] which manage a hashing structure for each file.

Extending the functions offered by GNU dbm, DGDBM has a total of 10 functions that the programmer may invoke in his application (which must be written in ANSI C language/Kernighan 88):

- **begin_transaction**: with the invocation of this function, the application marks the beginning of a distributed transaction that will contain requests for reads and writes over files and also local operations (i.e. these last manage the interaction with the user and manipulate the data resulting from the requests).

- **end_transaction**: with the invocation of this function the application marks the end of a distributed transaction. The system will assure the atomicity of the transaction [Mullender 89] completing its effects on the data involved or annulling these effects in the event of problems such as crashes of participating sites or concurrence conflicts.

- **dgdbm_open, dgdbm_close, dgdbm_purge**: with these functions the application requests the open, the close or the elimination of a particular file. If the open refers to a new file, the parameters must specify the logical site that will contain the file, its unique name, its block size and others parameters; with this information the system will update the APPLICATION CATALOGUE which is replicated over all sites and contains location information of the files. The catalogue must also be updated when the elimination of a file is requested.

- **dgdbm_fetch, dgdbm_store, dgdbm_delete**: with these functions the application requests the read, the write or the elimination of a record of a particular file by specifying the value key of the register, the file name and a local variable associated to the register contents (this last parameter is not necessary for the dgdbm_delete function). The parameters of these functions are the same of the associated GNU dbm routines which suppose a file register structure composed by two character strings of any length, corresponding to the key and the contents. (the contents conversion to another more detailed structured is a responsibility let to the programmer).

- **dgdbm_firstkey, dgdbm_nextkey**: these functions allow the access of all records of a particular file by physical storage order.

- **dgdbm_reorganize**: this function allows to shrink the space used by a particular file.

All the functions return an error code: if any function returns a value of 0 indicating some problem (for example, a nonexistent file in a dgdbm_open) the system will roll back the current transaction when it executes the end_transaction function. In another hand, an application can contain multiple not nested transactions.

Summarizing, the application written by the programmer deals with the local interaction with the user, while the management of all the distributed transactional features of the system is done by the functions provided by the DGDBM environment. For guarantying data consistency, it is expected that all accesses to the files, including the
local ones, are made through the services interface of DGDBM described here.

3. ARCHITECTURE OF THE DGDBM SYSTEM

A distributed application on multiple files that invokes the services of DGDBM, gives rise in execution to a CLIENT process which is going to interact with SERVER processes provided by the system.

Conceptually, a CLIENT process is composed by two modules:
- the application itself: it interacts with the user and invokes the DGDBM distributed services through the provided functions (it can include several distributed transactions)
- coordinator module: It synchronizes the execution of the distributed services requested by the application. In order to achieve this, it sends each DGDBM file request to the SERVER process of the site where the file is located (i.e. the site location is derived from the APPLICATION CATALOGUE). Likewise, it synchronizes the SERVER processes so as to assure the consistent execution of the distributed transaction required by the application.

In terms of implementation, the CLIENT process results from the compilation of the application written by the programmer, together with the DGBDM library.

Each SERVER process is responsible of local effects of the requests coming from CLIENT processes (each CLIENT is related in any time to a distributed transaction under execution). For this, the SERVER uses a technique based in locks in order to control the concurrence of multiple transactions asking its services.

The transaction operations which affect the local files of a SERVER constitute a sub transaction. In order to administrate the sub transactions with a good performance, the SERVER creates a SLAVE process for each sub transaction. For each transaction request coming to a SERVER, it verifies that the lock requirement is satisfied and then it sends the request to the SLAVE process associated to the transaction. The SLAVE transforms the request into local invocations to GNU dbm routines. A SLAVE will end its execution uniquely when the associated CLIENT application ends itself.

For the management of the APPLICATION CATALOGUE, which is replicated over all the sites, there exists a FILER process who is in charge of the consistent update of the catalogue copies. This process controls the concurrence of transactions intending the update of the catalogue primary copy (located at the logical site 0) and it also assures the update of the secondary copies (located in the others sites).

Figure 1 shows the architecture of the DGDBM system in terms of CLIENTS, SERVERS, SLAVES and FILER processes and its related modules. It also illustrates where the DGDBM and GDBM interface are used:

a) The application module, component of a CLIENT process, uses the DGDBM interface for invoking distributed.
b) A SLAVE process uses the GDBM interface for accessing local GNU dbm files.
c) A SERVER process uses the GDBM interface for accessing the local copy of the APPLICATION CATALOGUE.
d) The FILER process uses the GDBM interface for accessing its primary copy of the APPLICATION CATALOGUE.

The figure 1 shows, for simplicity reasons, the CLIENT and SERVER processes in separated sites. Actually, the
In the case of simultaneous multiple transactions, there will be a CLIENT process in the origin site of each transaction and a SLAVE process in each site with files intervening in the transaction. On the other hand, there is a SERVER process in each site having files of the system. A CLIENT process requires general communication with all SERVER processes. If a transaction requires the creation or deletion of a file, its CLIENT process will also communicate with the FILER process in order to obtain the consistent update of the APPLICATION CATALOGUE copies.

4. COORDINATION AND EXECUTION OF DISTRIBUTED TRANSACTIONS

The Coordinator module, component of the CLIENT process, is in charge of coordinating the execution of each distributed transaction requested by the application. A transaction is delimited by the application through the begin_Transaction and end_Transaction functions, and it contains requests over the application files (invoked through the DGDBM functions), together with C local instructions (i.e. interaction with the user, management of the data resulting from a query, etc.).

The execution of a distributed transaction in the DGDBM environment is ruled by the Two Phase Commit Protocol ([Ceri 84], [Coulouris 88], [Mullender 89], [Umar 93]) and requires the synchronization between the CLIENT
process of the site where the transaction originates and the SERVER and SLAVE processes of the sites where the involved files by the transaction reside. If the transaction creates or deletes a file, the synchronization must also cover the FILER process which manages the consistent update of the APPLICATION CATALOGUE.

The following are the main functions performed by the DGDBM system in order to support distributed transactions:

- **Atomicity:**
  In order to assure that all effects of a transaction or not one will be made, the synchronization of the processes which participate in a transaction is governed by the Two Phase Commit Protocol [Mullender 89] in the following manner:
  * First, the transaction is divided in sub transactions, one for each participant site.
  * The CLIENT process of the transaction, associated to the origin site, communicates the component actions of each sub transaction to the SERVER processes of the participant sites. Each SERVER process communicates this information to a local SLAVE process which will be associated to the transaction for performing the actions over the local files.
  * The CLIENT process of the transaction consults the decision of each SERVER process in order to commit or rollback the transaction: if all participant SERVER processes are in accord to complete the transaction effects, the CLIENT process will send a commit order to these SERVER processes; in the opposite case, the CLIENT process will send a rollback order.
  * In this manner, each participant SERVER process is responsible of the atomic execution of the sub transaction over the local files (through the SLAVE process) while the CLIENT process coordinates and assures the transaction atomicity over all sites.

- **Recovery:**
  In DGDBM system, the atomicity of each transaction is assured even in the presence of failures: for this purpose, the CLIENT process manages persistent information about the transaction state with three principal values: *initial state*, if any decision has not yet taken about the transaction, *commit state*, if the commit decision has been taken after a positive consultation to all participant sites, and *abort state* if the rollback decision has been taken because one or several participant sites don't want to commit (or because its answer doesn't arrive after a time delay). Depending of this state information, the actions that the CLIENT process must undertake after a crash failure, are the following:
  * to send the commit order to all participant sites, if the transaction state is commit
  * to send the rollback order to all participant sites, if the transaction state is initial or abort (in these two cases the final state of the transaction is abort).

In DGDBM system the CLIENT process is not permanent because it executes a particular application. For this reason, when the CLIENT process takes a decision of committing or rolling back a transaction, it communicates this decision to its local SERVER process, which is permanent, and after that, the CLIENT process could initiate another transaction or finish its execution. In this way, it is the SERVER process and not the CLIENT process who is responsible of undertaking the recovery actions after a crash failure for a transaction originated in its site.

In the order hand, for assuring local atomicity even in the presence of failures, each SLAVE process manages persistent information about the state of a sub transaction and about the intentional operations associated to this sub transaction (i.e. the operations that the transaction must complete over the local files in the case of a commit decision). The principal states of a sub transaction indicate if the SLAVE process has just started to
participate in the transaction (initial state), has been given its accord for committing (precommit state), has received the commit order (commit state) or has received the rollback order (abort state). With this information, the SLAVE process will be able to complete or to rollback the sub transaction effects after a crash failure, except in the case of the precommit state: this is a state of uncertainty [Mullender 89] and the SLAVE process must consult the transaction state to the origin transaction site or to the other participant sites in order to take a definitive consistent decision.

Seriability and isolation:
In order to preserve the consistency of the system when several transactions are executed concurrently, their final effects must be the same of the effects of their sequential execution [Ceri 84]. This seriability property is obtained in the DGDBM system through a lock mechanism: each SERVER process controls the concurrent access to the local files among several transactions through a TABLE OF LOCKS with two possible modes: shared and exclusive. In order to identify a transaction in a unique and orderly way with respect to other transactions of the system, each transaction has a stamp composed of two data: the number of the logical site which is its origin and a consecutive number (i.e. logical hour) for the transactions originated in that site; a transaction is "older" than another if its logical hour is the lowest or if the two logical hours are equal but its site number is the lowest [Lamport 78].

The consistent use of locks by each SERVER process assures distributed transactions that satisfy the properties of seriability and isolation (this last one means that no transaction must observe the intermediate effects of another transaction in course of execution).

When a SERVER process can assign the adequate lock to a transaction request which has arrived to the site, the request is communicated to the associated SLAVE process. In the case of concurrency conflict of the request with other transactions, the SERVER process will put the request in a WAITING LIST. When a SLAVE process ends the committing or the rolling back of its sub transaction, the SERVER process liberates the local locks of the sub transaction and, after that, it newly examines the kept requests in the WAITING LIST.

Deadlocks prevention:
The use of locks for concurrency control among transactions may produce local or distributed deadlocks in which no transaction could finish. In the DGDBM system it is applied a prevention technique for deadlocks in order to solve such situations. The selected technique is "Wound or Wait" [Maekawa 87] which is applied by each SERVER process when there is a concurrency conflict between a transaction request with another transaction that has the requested lock: in that case, a younger transaction makes a rolling back and transfers the lock to an older transaction which request this same lock (the priority between transactions is established through the total order derived from its stamps). The stamp of a rolled back transaction is conserved for the next execution of a transaction in the same application in order to assure that a transaction that must be reexecuted several times because of deadlocks prevention can finally attain success in its execution when it becomes the older transaction of the system. The Wound or Wait prevention technique for deadlocks is simple for being applied by each SERVER process and it doesn't imply the overhead of building a centralized graph of waiting for locks like as in the detection techniques [Maekawa 87]: however, all prevention techniques can produce unnecessary roll backs of transactions.

Transparency with respect to data distribution:
The application programmer in the DGDBM environment obtains transparency with respect to file location thanks to the APPLICATION CATALOGUE which contains information about each file location given its unique name. This catalogue is replicated over all sites in order to obtain its efficient access by the
transactions.

Consistent update of the APPLICATION CATALOGUE:

In the DGDBM system the update of APPLICATION CATALOGUE copies is governed by the "Primary copy - secondary copies" protocol [Mullender 89]. Corresponding to this protocol, there exists a primary copy of the APPLICATION CATALOGUE in a unique site of the system (the logical site 0), where is assigned the FILER process, and all other copies (i.e. one for each other site of the system) are considered as secondary. The primary copy is always up to date because any change is first made over the primary copy and then over the secondary copies.

The FILER process attends update requests over the APPLICATION CATALOGUE that are derived from the file creation or deletion operations included in some transactions. For the concurrency control between several transactions attempting to update the APPLICATION CATALOGUE, the FILER process assigns an exclusive lock to the first transaction who requests to modify this catalogue and builds an intention list with the intended modifications; when this transaction makes a global commit, the FILER process modifies the catalogue, sends the intention list to the SERVER processes and liberates the lock. Each SERVER process will modify its APPLICATION CATALOGUE copy when it receives the intention list from the FILER process but, in the case of crash failure, the SERVER recovery will begin by asking to the FILER process for the updates made during its isolation.

5. DESIGN AND IMPLEMENTATION OF DGDBM API IN TERMS OF THE JOYCE+ MODEL OF DISTRIBUTED SYSTEMS

JOYCE+ is primarily a model of multi-site distributed systems that is independent of specific network protocols ([Franky 90] [Arenas 91]). In this model, a distributed system is a set of processes, named agents, that cooperate in a common task through messages interchange. The communication is made through abstract objects named "channels" which allow the unidirectional transmission of one message each time (these channels are in fact the abstraction of the network and of its communication protocols). For each channel there exists an destination agent and one or several source agents. The association between an agent and a channel is made through another abstract object named "port" which is local to the agent and has an input or output role depending of the channel use by the agent.

In the JOYCE+ model there are classes of agents that supposes a common behavior for all the agents of a same class. Each agent executes in a logical site of the system. The model is dynamic in the meaning that any agent can create a new channel and can activate a new agent of a particular class assigning it to a particular logical site. In its activation, an agent can receive as parameters the references to existent channels for establishing communication with others agents.

The principal advantage of JOYCE+ model is its simplicity and the network abstraction that is offered to the programmer of distributed systems: the communication primitives for sending or receiving a message allow the asynchronous communication [Sloman 87] between agents by only naming a local port from the point of view of each agent (the channels are not named and there isn't any dependent aspects on the network nor on the operational system). The association of the logical sites to specific nodes of a network is established in a SITES CATALOGUE without affecting the agents specification; several logical sites can be associated to a same node.

JOYCE+ model is nowadays offered to the C programmer as an API [Vargas 93] with functions for channel
creation, agent activation, sending and receiving messages and polling for the reception of several possible messages allowing to specify what to do in "time-out" case (that is, when the agent doesn't receive any message after a maximum time delay).

The implementation of the DGDBM API was made in terms of the JOYCE+ API: in this way, each DGDBM function is defined in terms of the JOYCE+ functions. The JOYCE+ API version that was utilized is the UNIX version; this version is itself implemented over several communication and synchronization mechanisms that are common in UNIX environment: sockets (for remote communication), message queues (for local communication), FIFO pipes, signals, demon processes, etc. ([Comer 93], [Stevens 90]). In the design and implementation of the DGDBM API we were concentrated in resolving the synchronization problems for supporting distributed transactions but we were not aware of the use and associated problems of UNIX communication mechanisms: this was the fundamental advantage derived from the JOYCE+ model utilization.

Figure 2 shows the principal communication relations among CLIENTS, SERVERS, SLAVES and FILER agents corresponding to multiple transactions. These relations are expressed in the JOYCE+ model showing the channels, agents and ports elements (for simplicity, the communication relations are represented in a partial manner).²

²The agents are represented through ovals, the channels through arrows and the ports through diamonds.
The initiation of the DGDBM system in terms of the JOYCE+ model is made through the activation of an INITIAL agent in any logical site: this agent first creates all the system channels and then it activates a SERVER agent in each site of the system and the FILER agent in the associated site to the primary copy of the APPLICATION CATALOGUE. In these activations the agents assign to its ports the channels references which will allow the communication between SERVERS, SERVERS with the FILER agent, and SERVERS with the potential CLIENT agents.

When an application that uses the DGDBM API initiates its execution, it first finds out the references to the system channels in a file generated by the local SERVER agent; after that, the application assigns these references to its ports and becomes a CLIENT agent for the system.

Each SERVER agent is always active attending the CLIENTS requests over its local files. When a new CLIENT agent requests the services of a SERVER agent, this last activates a SLAVE agent which is assigned to the CLIENT. The SLAVE agent will be in charge of making the transaction actions requested by the CLIENT agent over the local GNU dbm files.

The FILER agent is in charge of the consistent update of the copies of the APPLICATION CATALOGUE when the transactions request operations for creating or deleting files.

6. PERSPECTIVES AND PLANNED EXTENSIONS

Presently, the DGDBM API is available for programming C applications that include distributed transactions on a local network of UNIX computers under TCP/IP protocols [Comer 93]; presently, these computers are Digital MicroVax II with ULTRIX system. DGDBM manages all the aspects of distributed transactions (i.e. concurrency control and failure recovery) in a transparent manner for the programmer.

DGDBM API constitutes an useful support for distributed application programming, specially in contexts where the use of distributed databases management systems results inadequate or very costly. With this perspective, DGDBM uses in its implementation free software tools like the GNU dbm routines and its final version will run on LINUX system [Linux 94] which is a free version of UNIX. To long term, we want to include DGDBM as part of a free platform for developing distributed systems which will be specially available to our regional universities (DGDBM itself will be a free product).

From the point of view of design and implementation, DGDBM constitutes a big example of utilization of the JOYCE+ model for distributed systems. As DGDBM designers, we have appreciated the usefulness of working with a high level model that is independent of specific network protocols and we have better concentrated in expressing the solutions for managing distributed transactions. We have also detected some inefficiency problems of the JOYCE+ API that we will solve in the next future with a better implementation of the local inter processes communication module through UNIX message queues (the final version of JOYCE+ API will also run on LINUX system).

In another hand, we are working in a parallel implementation of DGDBM in the RPC paradigm ([RPC 87], [Kong 90], [Catañeda 94]) with the objective of making a final comparison of the two implementations (i.e. JOYCE+ versus RPC) in the aspects of efficiency and ease of design specification. Also, we intend later to compare DGDBM with similar products available as free software: such is the case of TDBM [Brachman 93] that offers a
similar interface and manages distributed transactions.

In future versions of the DGDBM project we plan to extend its facilities by supporting the following aspects:

- **Additional services that would be typical of a distributed file management system** management of directories and subdirectories, management of file access rights of users and options for defining user groups with the possibility of sharing files between different applications that are associated to different user groups.

- **Improvement of effective concurrence between transactions**: by managing locks to a register level and not to a file level in the operations for reading or updating a unique file register (i.e. dgdbm_fetch, dgdbm_store).

- **Additional functions for the interpretation of file register contents as a structure with fields**: the administrator user of a DGDBM application will be able to define the register structure of each file class that belongs to the application; with this information, the DGDBM API will provide functions for the automatic conversion of retrieved register contents from a character string to the associated structure.

- **Additional functions for making simple SQL queries on a file**: after having the option of interpreting the file register contents as a structure with fields, the DGDBM API will provide functions for retrieving the multiple registers of a file that satisfy certain field conditions. This kind of SQL queries [Groff 91] will be restrained to a file each time, avoiding the optimization and overhead problems of distributed Joins between several files (i.e. seen as relational tables) that are proper of distributed databases management systems [Ceri 84].

- **Management of replicated files**: DGDBM will permit to the user to specify a logical file with copies at several sites and will assure their mutual consistency [Mullender 89]. This corresponds to extending the actual management of the APPLICATION CATALOGUE (which has a primary copy and multiple secondary copies) to all data files.

With these planned extensions for DGDBM project we pretend to build a complete and free software for supporting all programming aspects of distributed transactions on files. The source software well documented will be valuable as part of our free platform for developing distributed systems.

**BIBLIOGRAPHY**


[Brachman 93]: B. Brachman; "TDBM version 1.1", free software available through an anonymous ftp to directory pub/local/src/tdbm.tar.Z from the node cs.ubc.ca, September 1993.


[Linux 94]: information about this free version of UNIX system is available through an anonymous ftp to directory /pub/Linux from the node sunsite.unc.edu, April 1994.


[Nelson 93]: P. A. Nelson; "GNU dbm (GDBM) version 1.7", GNU free software available through an anonymous ftp to directory /pub/gnu from the node prep.ai.mit.edu, September 1993.

[Rangel 94]: E. Rangel; "DGDBM: Ambiente de programación de aplicaciones transaccionales distribuidas sobre archivos de acceso directo", Master thesis, Computer Science Department, University of the Andes, Bogotá-Colombia, Mars 1994.


[Stevens 90]: R. Stevens; "Unix Network programming", Prentice-Hall 1990.


[Vargas 93]: J. F. Vargas; "API para el Desarrollo de Sistemas Distribuidos bajo el modelo JOYCE+ en ambiente UNIX - TCP/IP", Undergraduate thesis, Computer Science Department, University of the Andes, Bogotá-Colombia, 1993.