UNO: USENET News on Optical Disk

A. Garibbo,
L. Regoli,
G. Succi

University of Genoa
Italy

1. Introduction

The size of a WORM optical disk is greater than a Gbyte and it is likely to grow fast within few years; moreover storing and retrieving USENET news is becoming tedious and difficult: at present time a user has easy access to news if he knows exactly which ones he wants to consult; besides reading daily news takes little time using standard read-news tools.

Troubles arise when one wants to find some news only knowing few features because the help he has is merely a hierarchical organization of the news supplied by the USENET system: actually, such a tree-shaped framework seems to be quite unsuitable as long as:

i) the structure is not strongly enforced

ii) quite different leaves lie in the same directory

Owing to the high rate of news traffic, lots of space is needed, and usually each local network connected with USENET either devotes too much space to archiving or it needs frequent backup on tape.

UNO – USENET News on Optical disk – attempts to solve this kind of problems, since more than four years of full news, at the present rate, can be archived on a WORM disk.

All facilities provided by standard readnews tools are enclosed in UNO; moreover it supports an incremental knowledge driven search, which allows interactive data retrieving without either knowing exactly the wanted news or having to deal with all the news of a USENET directory.

UNO provides easy interaction through a smart query language, remote query and intelligent programmable selection of relevant news.

UNO was developed on a workstation named Arianna, based on a National 32032 processor, which runs UNIX System V.3. A WORM disk is fully integrated in the global file system; UNO is designed in C++ according to object oriented programming and software engineering criteria.

2. Object Abstraction in UNO

As written above, UNO is designed in object oriented programming style, which means:

2.1. Fully modularity

Four are the big independent modules that build up the skeleton of UNO:

- database
- remote query system
- user interface
- archiving mechanism

They interact with one another via message passing, e.g. calling member functions and storing data on files. In the pictures below the data flow through the system is shown
As can be easily understood such a modularity allows substitutions and improvements in each module without having to be concerned with the global system consistency.

2.1.1. Data abstraction

Data abstraction implies easy definition of modules' global architecture and clean code where errors can be easily detected, the use of C++ programming language, in such a context, sounds as a natural choice: C++ indeed supports single and static inheritance rules often useful, e.g. in ordering access lists.

3. Database Structure

Two different levels are identified in our database, as it is shown in the picture above:

- Query parser
- Core

3.1. Query parser

Its aim is to provide the user a simple interface with database: its input is an understandable request while its output is a query easy to be processed by the database core. In other words, the query parser translates the query into a core input format through an interaction with the user kernel interface.

The structure of the core query is really simple, as can be argued by its class definition and by the following example:

```cpp
class query {
    char *arg[max-key];
    :
    :
    public:
    :
    void set_subject_id(char *c);
    :
    void set_author (char *c);
    :
    void set_keyword (char *c);
    :
    istream& operator >> (istream& s);
    ostream& operator >> (ostream& s);
};
```
Example: mapping of request of news with author Ritchie and subject "pic"

<table>
<thead>
<tr>
<th>Query.arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
</tr>
<tr>
<td>country</td>
</tr>
<tr>
<td>author</td>
</tr>
<tr>
<td>subject</td>
</tr>
</tbody>
</table>

Once the core query is compiled, a pattern matching with the news header is fired.

### 3.2. Core structure

![Diagram of database management structure](image)

News headers and news are stored in different files to supply a more flexible structure. Several tables – one for each key-field – are used to build a virtual associative memory with which the query parser interacts.

![Diagram of key selector and tables](image)

Because of their size, those tables are dynamically loaded in RAM only when it is needed, during query processing.

Tables data structure follows with an example.

```c
class couple {
    friend class table;
    char *c;
    header ptr;
    couple *next, *previous;
};

class table {
    :
    couple *head, *index;
    virtual list *read_table (char *expression);
};
```

Example: table for "NAME" field
<table>
<thead>
<tr>
<th>NAME</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernigan</td>
<td>123428.hd</td>
</tr>
<tr>
<td>Sinatra</td>
<td>8875.hd</td>
</tr>
<tr>
<td>Ritchie</td>
<td>2012.hd</td>
</tr>
</tbody>
</table>

Tables are sorted in increasing order of specificity, for example, the field "author" is more specific than the field "country" (several authors come from the same country!).

The query processing mechanism acts as follows:

- Table for the most specific field set by the user is loaded in RAM.
- A first pattern matching is performed on this table to create a list of news including the required ones.
- The previously loaded table is removed from RAM.
- Pattern matching, on the other fields specified by the user, in decreasing order of specificity, is performed on news header files pointed to by header field of class couple instances belonging to the previously made list.

The relational database is designed with a smart splitting of data between hard and optical disk. Headers and news are saved on the optical disk, while tables and several other suitable information are stored in the hard disk both because such data need to be update (a Write Once Read Many disk is used!) and because of the frequent access to them (the access time to the WORM disk is about five times greater than to the hard disk).

When the optical disk is nearly full and it has to be substituted, the tables are automatically saved on it, whereas at installation of an already used WORM disk (e.g. for consultation) the database in the hard disk, copies them from the optical one, if they exist.

4. User Kernel Interface

Our aim was to make the search through the database nice and so provided a series of masks the user has to deal with according to one of two possibles custom files, which can be enclosed in the home directory of the user: .uno_db and .uno_su ; the file called .uno_db must be written following the guidelines of a template provided by the designers, it has a higher priority than the other .uno_su which can be defined at runtime using one of the options of the system main menu.

Example 1: Structure of .uno_db if user wants to select
- author field
- dept field

but not one country one.

    # if you want to select author, write it below
    author

    # if you want to select country, write it below

    # if you want to select dept, write it below
    dept

Example 2: Structure of .uno_su (the situation is the same as the example 1)

    author
depth

Be careful of the fact that this file acts as internal form, so the user is never request to edit it: UNO allows the user to see and to update it runtime.
The screen is structured in windows as can be argued from the following picture.
5. Remote Query System

The remote query system consists of two parts: the first one is the account checker that checks the validity of the user request, updates accounting tables and sends warning messages to behindhand users; the second one is the database interface that processes the requests and sends the answers.

The account checker policy is the following:

- Each user has an initial amount of resources which decrease whenever he requires a transaction, proportionately to the size of the interchanged information packed.
- A credit – up to 5% of the starting quota is granted.
- Warning messages are sent when the resources amount becomes less than 10%, 5%, 2% of the starting one. Moreover a warning message is sent for each request as long as credit is not exhausted.
- All the requests are refused when the credit is exhausted.

Database interface works in three steps: first it translates the requests from the user format in a one suitable for the database, then sends a message to the account checker and it creates a query instance that it submits to the database and finally returns data files to the remote user.

Remote requests can be performed using a supplied mail tool, similar to the user kernel interface, but written in C language instead of C++ to make it fully portable, such a tool can also mail a query, via UNIX system call, to a receiver, whose address is specified in one of the custom files; in our implementation the receiver is

The user can write the query by himself, anyway, taking care of the fact that badly formed requests are rejected.

Example 1: well formed request

: 
  author : Ritchie
  subject : pic
 : 

Example 2: badly formed request

Dear Mr. UNO,
  would you like to send me all
Ritchie's news about "pic"?

Yours faithfully,

LOUIS GHIOUS

6. Archiving mechanism

Archiving of news is performed by a self-activating time driven daemon which works once a day.

News are stored in files by the mail system, so the daemon recognizes them by reading the date field of the file descriptor. If the date is greater than the date of the last activation of the daemon – previously stored in
a global variable – the system processes the news. In this case a pattern matching is performed between
news and system custom file, that contains directives about interesting keys.
If the news results interesting, they are archived into the database, according to the mechanism described in
section 3.
The custom file, which contains selection directives, can be updated runtime by the superuser without editing it by mean of a specific command provided by UNO package.

7. Conclusion

We think UNO is quite remarkable both because of the idea of building a big and user-friendly database
where news that otherwise could be lost can be saved and because it can be further improved e.g. using
sockets to perform remote query in a local network and applying A.I. techniques in the selection task.
Unfortunately it is more than an year that no news arrives in Italy, so at present time our system cannot
offer the perhaps most interesting news.

8. Acknowledgements

We wish to thank Prof. Joy Marino (University of Genoa - DIST) because he gave us the idea of this
project, encouraged us and helped us whenever we needed it.
We also wish to thank professors Antonio Boccalatte and Giorgio Olimpo of University of Genoa.

9. References

   1986
2. Bishop M. "How to Use USENET Effectively". 1986

behindhand file, which