A DECLARATIVE LANGUAGE FOR DATA AND TELEX EXCHANGE CONFIGURATION

J. Grompone, F. Brum, M. Casamayor

Interfase Ltda., Uruguay

INTRODUCTION

The traditional method used for the configuration of a telecommunications exchange, irrespective of its purpose, is to have a set of programs and procedures -in most cases not particularly friendly-which are used for the painstaking job of assembling the tables required for internal use. This involves training technicians obliged to work with codes, and quite frequently the equipment is discontinued before users have learned to handle the configuration techniques properly.

Following the design of a series of telex exchanges (1, 4) in Uruguay, it was considered necessary to modify this traditional approach and to adopt a method more in accordance with modern programming techniques. The idea was to design and implement a high-level language, LEP, which would provide a simple and abstract means of completing the exchange (2).

The Uruguayan National Telecommunications Administration has been using this language in the public telex network since 1986 and in the data network since 1989 [*].

The LEP language is a simple, declarative language designed to define the routing and customizing rules of a telex exchange, but the concepts employed can easily be applied to telephony or data transmission (these possibilities are at present under study). The language is in accordance with CCITT Series Z.300 Recommendations wherever relevant (3).

The user writes a LEP text using a text editor -possibly in the exchange console- and then follows the normal procedures: compilation, simulation of the configuration programmed, and installation of the latter in the exchange on a resident basis.

The compilation programs perform the static error detection tasks and generate the necessary tables. They also supply auxiliary information which help with the simulation and correction of errors involving logical inconsistencies.

This paper discusses the general abstraction used in this language based on the present packet switching experience. The lexical, syntactic and semantic details are not described here and the examples are rather informal. (6) [**]. LEP implementation appears under (5).

---

[*] The project was carried out in association with CONTROLES S.A. of Montevideo, as part of a joint supply contract with the Administracion Nacional de Telecomunicaciones (ANTEL) of Uruguay.

ABSTRACT VIEW OF AN EXCHANGE

From an abstract point of view an exchange is composed of three basic elements:

- Physical description: processors, terminals, files, counters, etc.

- Attribute description: protocols, parameters, user identification, etc.

- Logical description: numbering plan, routing plan, permissions, etc.

The main point of this paper is to propose an abstract view of an exchange in order to design a high level declarative man-machine language. This language must be capable of declaring the three groups of basic elements.

A declarative language uses statements or declarations in order to produce a structure or program. This program can be compiled, in the usual meaning, in order to obtain a collection of tables which are the real exchange description.

The physical description is made up of representation clauses, each one of which links a named abstract entity with a real physical object. For instance, it declares that COUNTER is located at a certain address and saved in a specific file.

The attribute description is a typical list of basic declarations. Each basic declaration states that one parameter has a value, that certain facility is possible, that a certain protocol is used.

The logical description is the most interesting aspect of the problem. In this description a set of statements links together named abstract entities which have been defined earlier.

The first two sets of declarations are practically contradictions free. The logical description, on the other hand, must be checked to make sure that it is free of contradictions. This is one of the most important points in the scope of a high level description language, a compiler and utility programs.

[**] Readers wishing to have additional information may request a MS-DOS compatible diskette with the syntax table, the LEP compiler, a routing simulation and a user's manual for a telex exchange by writing to: "Juan Grompone, INTERFASE LTDA., Zabala 1378, Montevideo, Uruguay" or by Telex to: 26597 INTERFA UY or by Fax to: (598) (2) 962965 or Unix Network to: grompone@incouy.edu.ar.

BASIC ABSTRACTIONS

Each of the physical communication paths is called a line. Subscribers, trunks, consoles, auxiliary devices or any other similar concept may be considered a line.

Lines are accessed by codes. A code is a string of decimal digits. CCITT recommendations F.69 and X.121 establish the international numbering plan for telex and data public networks. The code is completed with national rules. In a private network the choice of numbering plan can be (almost) free.

In order to personalize an exchange it is necessary to specify at least the following points:

- the physical attributes of each line: protocol used, parameters, time outs, counters, etc.

- the attributes of each user: public and private (password) identification, class of service, facilities permitted, etc.

- the line structure: the way in which the lines are grouped.

- routing plan: a collection of rules for the association of codes with destination lines.

- implemented services: actions activated by a selection code. Information menus, information mailboxes management, monthly account information, etc.

- access rules: permission or prohibition to access a line from another line, from user, etc.

- representation clauses: declaration that links a hardware object with an abstract object.

This paper proposes an abstract view of an exchange in order to personalize it with a high level man-machine language.

The abstract concept of line is simple. A line is each of the N integers between 0 and N-1. N is the total number of lines. Later on the data case will be discussed.

Lines are put together in line groups. A line group is a named set of lines in a certain order, but not necessarily consecutive in the natural order. Each group is associated with an identifier which represents the set. All the group lines together include all the lines with neither absences nor repetitions. The line groups are disjointed, ordered and complete partitions of the set of integers between 0 and N-1.

Groups may be assembled in a more complex structure called float set. A float set is a named set of line groups in a certain order, each one associated with a traffic quota distribution.

LINE AND USER ATTRIBUTES

The declaration of attributes, be they of physical terminals or user data, is a problem which may be set apart from the other exchange configuration declarations. In our design of the telex, these declarations used to be in the same program but, in the case of data, the convenience of a separate program arose. In the case of a X.28 terminal, it is necessary to define, at least, the following attributes:

- 22 parameter values of initial profile
- Accepted Facilities
- 8 time-out values
- Physical protocol characteristics

It is also necessary to define, in the case of one dedicated user, its identification, telephone number, etc.

All this shows that declaration of attributes is extensive but seldom modified. For this reason it is desirable to set aside the configuration problem.

This simple problem of tables declaration may be performed pursuant to Recommendation Z.315 using clauses such as the following:

```
ITT        T1=3, N2=10, K=7, LEVEL_2=LAPB,
           D_BIT=no, M_BIT=yes ;
```

In this example ITT is assumed to be the name of an X.75 line. The clause defines the parameters T1, N2 and K (names used in the X.75 recommendation) and defines LAPB as the frame level protocol. It also declares that D bit procedure is not supported and M bit procedure is supported.

The named declaration of parameters allows parameters to be omitted (possibly because there is a default value) and the declaration order to be freely chosen.

This text may be compiled, stored, and by using a suitable table generator, the elements necessary for the exchange can be constructed.

ACCESS AND ROUTING

The problems of access and routing are the most interesting points of an exchange configuration, and it is over this step where it is necessary to be cautious.

In an exchange with N lines the problem arises in describing, for each selection code, which is the ordered list of lines working as a destination. If there are different possible alternatives, the list will have more than one element.

An abstraction level is gained when circuits are discovered to be assembled in entries as groups (equivalent circuits which may be selected in sequence), floating groups (sets of groups which may be selected in sequence with a settled weight). For a circuits exchange it is enough to define the groups by numbering their lines and the floating groups by numbering a list.

In accordance with Z.315 Recommendation a typical group declaration is as follows:

```
LONDON     Akey
           ( D'70 .. D'79)
           ( D'20 .. D'35) ;
```

The group is used for telex with type A signaling and keyboard selection codes. It is identified with the name LONDON. In this clause, D' specifies that a number in base 10 follows. It has 26 lines. The first line is 70th line in the exchange. The line following

the 79th is the 20th. The group ends in line 35th.

In the case of a data exchange an additional abstraction level is required. A packet switching terminal (X.25 or X.75) has many logical channels and cannot be identified with a single line. The natural way to see a packet switching terminal is as a set of lines, one identified with each channel. At this point it is necessary to distinguish two kind of "lines": physical and logical lines.

A physical line is a communication path. A logical line is an abstraction. Each physical line has different logical channels (maybe one in a X.28). A logical line is either a real communication circuit or a pair of entities: a physical packet communication path and a logical channel.

With this abstraction, circuit and virtual circuit are unified under the physical line entity.

An example of channel declaration is:

```
TRANSPAC  X.75

     D'7  PERMANENT_ONE    D'4, D'5
          INCOME_ONE       (D'10 .. D'15 )
          BIDERECTIONAL    (D'30 .. D'35 )
          OUTCOME_ONE      (D'50 .. D'69 )

     D'8  INCOME_TWO       (D'10 .. D'15 )
          OUTCOME_TWO      (D'50 .. D'69 ) ;
```

This example declares a X.75 group named TRANSPAC which has two physical lines, the 7th and the 8th. In the first line 2 channels are declared for permanent circuit use, 6 channels for incoming traffic, 6 for bidirectional traffic and 20 channels for outgoing traffic. In the second physical line 6 and 20 channels are declared for incoming and outgoing traffic.

If we assume that channel 4 of physical line 7 is the 100th logical line, then, this clause declares 60 logical lines. The last one is the 159th. logical line in the exchange.

Now the problem of routing is to associate a list of logical terminals to be passed over in an orderly manner to a code or set of codes.

```
716*            ARPAC / INTERDATA / WUI ;
7482?????       LOCAL ;
[200..299]*     TRANSPAC ;
```

The declaration 716* means that every code beginning with 716 and followed by any number (including none) of digits matches the clause. The ? can be considered as one digit. 7482????? matches only a nine digit code beginning with 7482. A code range can be specified as [200..299]. This expression means any three digit code between 200 and 299.

The first clause declares (in words) that Bolivian data subscribers are accessed, as first choice, though the Argentinian ARPAC group. If all those lines fail to establish a circuit, then Brazilian INTERDATA group is the second choice. If all those fail, WUI is the last choice.

The second clause (in words) directs all Uruguayan traffic to LOCAL. The third clause declares (in words) that all European data subscribers are accessed by TRANSPAC.

In some numbering plans it is necessary to recode the called address. To recode is to transform a selection code into another selection code for it to be sent to an outgoing link. These operations are frequently performed in international communications.

A format is a named recode procedure. It can be declared as:

```
AS_IT_IS        *          *          ;
TRUNCATE_3      ???*       *          ;
ADD_A_TWO       *          2*         ;
SIXTEEN         *          16         ;
```

As above * means any number of digits and ? means one digit. The first clause declares (in words) that AS_IT_IS does not modify the code. The second clause eliminates the leading three digit; the third adds the digit 2 and the last one replaces any code by 16.

In a routing declaration, recoding formats can be explicitly declared. For example:

```
722*        ARPAC TRUNCATE_3 / INTERDATA
            / WUI ;
```

This clause modifies a previous clause. It declares (in words) that called addresses must omit country prefix when they are accessed via ARPAC. In the other alternatives they are not recoded in order to preserve country identity.

The problem of accesses is also a problem of lists declaration. For each group the permitted or forbidden list of destinations is declared. An example of a forbidden declaration is:

```
ARPAC      ARPAC, INTERDATA, WUI, TRANSPAC ;
```

This clause declares that it is forbidden for ARPAC lines to access ARPAC, INTERDATA, WUI o TRANSPAC. It is not permitted (in words) for ARPAC subscribers to transit trough the exchange.

As a conclusion, the simple rules of lists declaration make it possible for the problems of a declarative language to be solved.

SERVICES

Though the conceptual problem of a service is very simple, it is not an easy one to solve from an abstract point of view. The fundamental problem is how to describe the possible services without entering into the hardware exchange details. Our first design assumed that, if desired, every service could be activated by an order from the console. Thus the declaration of a service presents the following aspect:

```
TEXT_QBF  { console command } ;
```

where the console command depends on the exchange. With this strategy, the service declaration falls outside the scope of the language and must be parsed, validated and executed by a different, and hardware dependent, automaton.

The opposite alternative consists of abstracting every possible service. In this case, it must include every possible service

as a primitive object in the declarative language.

A third possible alternative is to propose an abstract view of the services. In order to be able to declare mailboxes, information menus, billing counters, and other useful services it is necessary to declare, at least, named data areas and possible actions over those areas. Some of the basic actions are:

- Read a data area.
- Write a data area.
- Delete the data of an area.

In some services the following may be added to the above actions:

- Create a data area.
- Delete a data area.

Associated with a named data area, a set of representation clauses must declare its physical attributes: memory address or file used, length, kind of information used or stored, etc.

From the logical point of view a service is declared when the routing and access rules for the different kind of basic operations are given. With this approach the problem is very similar to the general routing case.

REPRESENTATION CLAUSES

A representation clause is a declaration that links a named entity with a physical object. Among the physical objects the following must be considered:

- The communication links
- The memory
- The disk

The clauses must be capable of declaration where the hardware objects are located among the available processors, pages of memory, disk units, etc.

Representation clauses are strongly hardware dependent. Examples of declarations are:

```
processor D'6        (D'0..D'7) ;
memory    H'50000    TABLE_1    ;
file      C:/MAIN    MENU       ;
```

The first example declares that physical lines 0 through 7 belongs to communications processor number 6. The second example declares that TABLE_1 (whatever this identifier means) is located at hexadecimal address 50000. The third example declares that identifier MENU is located at disk C: with path /MAIN.

Representation clauses provide a simple mean of multiprocessor declaration. This is important in exchanges of distributed architecture.

LANGUAGE ENVIRONMENT

The man-machine language must have a user friendly environment. The environment consists of text editors, compilers and utility programs. The main purpose of utility programs is to detect logical inconsistency errors and misunderstood specifications.

Undoubtedly, a compiler can perform a good analysis of consistencies of the declarations. In this way, it is possible to detect new configuration errors before they are installed in a real exchange. But it is also necessary to have a tool in order to verify some cases of interest in an interactive way.

In the first LEP design the user had a simulator to consult tables and perform routings. The program simulates the exchange routing actions as subscribers see it. Each of the lines or groups can be set busy or free, by name or as a number interval. Calls can be simulated and the program returns to the user final destination, as well as clearing calls and computation time used by routing automaton.

This tool is useful but does not detect subtle errors. At this time a simulator is being developed which will allow interactions such as the following:

- Indicate every destination of a code
- Indicate every code acceded to a destination

Interactions of this kind are similar to those which can be performed with an expert system.

CONCLUSIONS

The proposed abstraction for circuit exchanges has proved to be easily adaptable to data exchanges.

It is possible to repeat here what has been said earlier (5). A better implementation efficiency is obtained using a declarative language approach in relation to traditional dedicated programs. For example, to adapt the telex declarative language for the data exchange required less than one man-month of programming effort and this was actually performed successfully in a few calendar days.

The general conclusion of three years of work with man-machine declarative languages shows that, from the telecommunications engineering point of view, it is a very fertile field of experimentation. In the future better concepts abstraction and language design can be achieved.

Systems design, systems generation and systems administration are well known concepts for mainframe computers. With the evolution of man-machine languages the same ideas can be transferred to communications system management.

In order to keep the different fields of work apart, an abstraction effort must be done. This process is not a simple division but a real function structuration. So design, implementation and management will become separate fields with the accepted benefits of a structured organization.

REFERENCES

1. Grompone J.A. and Mace N., 1981, "Una central telex con procesadores triplicados". Anales de la VIII Conferencia Latinoamericana de Informatica, E2-E15.

2. Shaw M., 1984, "Abstraction techniques in modern programming languages". IEEE

Software, 1.

3. CCITT VIIth Plenary Assembly (Malaga-Torremolinos, 1984), Vol. VI, Fascicle VI.13 (Man-machine language (MML)), Recommendations Z.301-Z.341.

4. Grompone J.A. and Simini F., 1985, "Communications technology development: a case study in Uruguay". Communications Indonesia '85.

5. Grompone J.A., Brum F., Casamayor M., Simini F., Bogota, 1987, "El lenguaje declarativo como herramienta de implementacion: un caso de estudio en telecomunicaciones". Anales de la XIII Conferencia Latinoamericana de Informatica.

6. Grompone J.A., 1989, "A declarative language for the configuration of exchanges". ITU Telecommunication Journal, 1, 33-38.