

Request for Comments #51

4 May 70

ULLA

Proposal for a Network Interchange Language

Introduction

In this paper an attempt is made to specify a high level programming language for computer networks, and more specifically the ARPA network. The main concept introduced is the one of an abstract Network Machine, which is consistent with the idea of a HOST asking a service from the computer network considered as an overall computing facility. The dialogue is always between a HOST and the Network Machine which language is always the same, though its configuration may vary according to the real remote HOST.

From a programming language point of view, this concept is similar to the UNCOL proposed in 1958 [STR058] but never implemented, however; the application to a computer network implies a realtime interaction between programs. Also, the possibility for the user to use NIL either in a standard mode or in an extended mode where he defines himself his own entities should give to NIL a maximum of flexibility.

1. Basic concepts introduced in NIL

1.1 Aim of NIL

The two main objectives of NIL are:

- a) to describe the environment in which a program is executed (its complement); this involves the description of:
 - data formats and data structures
 - exchanges with input and output devices and characteristics expected from them
 - interface with operating system
- b) to express the front end part of an interactive system:

The data flow through an interactive system generally decreases as the data reaches the kernel of the system: it is assumed that in many interactive systems a separable module exists or

can be defined which involves a great amount of data exchanged with the user, and much less exchanged with the rest of the system. This module is called Front-End. It is important that the response time of the system is affected as little as possible by additional transmission delays. Also, it is desirable to keep the data rates as low as possible on the network.

It is assumed that the transfer of a Front End does not imply to solve the

1.2 NIL subcategorization

As pointed out by S. Volansky [] it's convenient to divide languages in several sublanguages corresponding to their main functions. NIL is thus subdivided in:

- a control sublanguage
- an operation sublanguage
- a data declaration sublanguage
- an environment sublanguage

1.2.1 Control sublanguage

The control sublanguage states WHEN a computation is done: It describes the flow of control or ordering of the computations. With some information contained from the other sections of the language, it also states WHERE the computations are to be executed.

As a computer network introduces loose connections between several systems, the control language of the Network Machine should be able in an elaborate

version, of assigning computations to available processors, taking into account the time delays and resource allocation problems involved. It is

1.2.2 Operation sublanguage

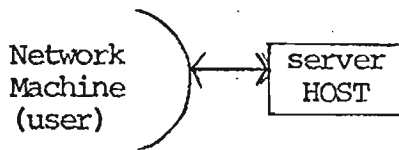
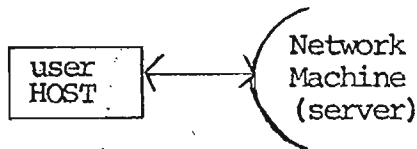
The operation sublanguage describes the operations to be performed on the data without indication of the sequencing between operations, it answers to the question of HOW an operation is performed. The operations are sub-

The data definition sublanguage is as well used for the purpose of documenting the set of standard beads.

The instruction set of the Network Machine stands at a high level permitting global transformations of data structures.

The environment of the Network Machine is determined by the subset of the environment of the server's HOST which is used by the program in execution; the system HOST-Network Machine can take two main configurations shown in figure 1.3.2.

- a) The Network Machine stands for the user of a program provided by the HOST (server HOST)
- b) The HOST machine is the user of a program provided by the Network Machine.



The server machine assigns its hardware environment to the user machine. This choice is made so that programs can be remotely used without being modified; it is up to the user of a remote program to adapt himself and his own environment.

Thus, when the Network Machine is server, it defines the Data Definition and Environment

figure 1.3.2

sections.

1.4 Implementation

The data and environment definition sublanguages should be able to describe as well environment and data in HOSTs as data in Network Machine. At the limit it should enable two programs written in different languages to communicate,

as long as the data representation they use are expressible in the data description sublanguage.

In each HOST will be implemented a "generator" which will accept rules describing the HOST data structures and environment and will generate an adequate translator to translate them in Network Machine format, as shown in the figure 1.4.1.

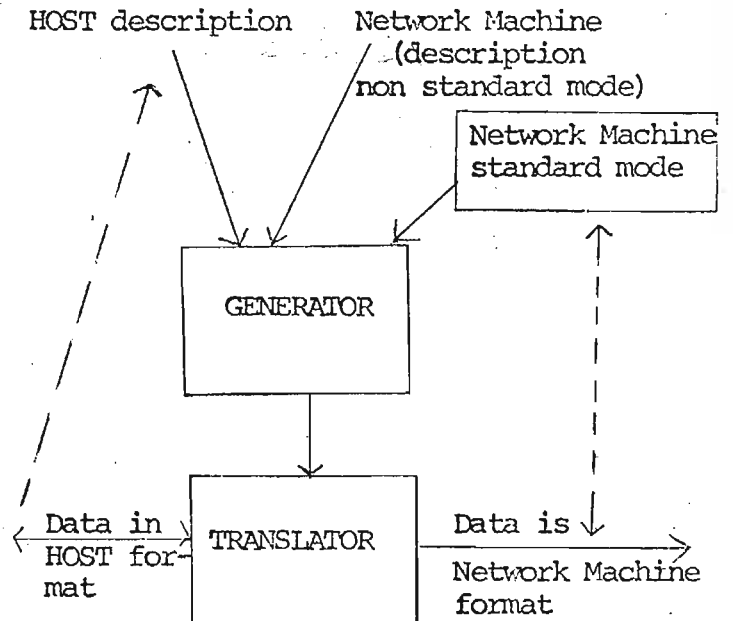
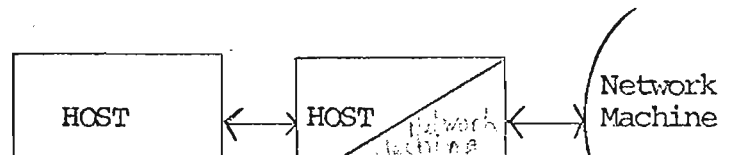


figure 1.4.1

Once the network machine standards will be settled it seems valuable to think about emulating the translator using a microprogrammed unit which would be either added to the Host or rather to the TMU



2. Data definition sublanguage

2.1 Fields

All communications with the Network Machine are done using strings of bits: these strings of bits, also referred to as messages are parsed by the receiving HOST to reconstruct inside its memory the data structures in its own memory and code.

Bits are grouped into significant fields: a field is a group of bits having definite contents. It may contain:

- a) an element of data (data field)
- b) some bit pattern specifying environment parameters
- c) a pointer
- d) the identification of some other fields.

The method to describe the formats of beads is derived from the method of description of a binary message suggested in RFC #31:

- a) each field is declared with its name and length in number of bits.
- b) commonly used fixed values of a field that correspond to a special meaning, may be given names.
- c) legal ways of concatenating fields are initiated by rules; when only certain fixed values of a field are allowed, they may be either specified by their value or by the corresponding name.

2.2 Data beads.

Data fields (type (a)) are concatenated to form data beads: a bead is an indivisible atomic unit of data used as building element of any data structure to be transmitted between HOSTs and Network Machine. A bead is the smallest unit of data that can be referenced.

The legal ways of forming a bead by concatenation of several fields are indicated in a construction rule. Beads have a fixed length and an unambiguous structure. In real machine beads are usually defined as an integer number of contiguous registers. This constraint does not apply here, though it may turn out to be more efficient to favor HOSTs with, for instance, 32 bit words, and 4 bytes per word, which are the most common word structure on the ARPA network.

Data beads may be considered as the operands of the language in which fields of type (b) and (c) would be operators.

2.3 Control fields

The way data beads are linked one to the other and the environment in which they operate are specified by additional control fields which cannot be referenced and are operators on or identifiers of the following string of beads, or linkage between individual beads.

The scope of a control field may as well be all the beads or substructures of a structure if it is specified at the level

of the head of the structure. To be more precise, two kinds of structures of beads must be defined: homogeneous and heterogeneous structures.

A structure is defined as homogeneous if both a unique type of bead and a fixed parameter environment for the whole structure is specified at the head of the structure.

2.3.2 FLAG field

When set, the MODIFY field is immediately followed by an 8 bit FLAG field indicating which of the IDENTIFICATION and several possible PARAMETER fields are present; when set to one each individual bit means the following:

bit number

0	IDENTIFICATION field present
1	first parameter field present
2	second parameter field present
---	---
6	sixth parameter field present

Example: the parameter field of the standard bead BEAMMVT will contain the following fields.

- a 2 bit field indicating the type of movement generated

00	do not display	move the beam
01	display final point	point.
10	display vector	vector
11	unused	

- a 4 bit field indicating beam intensity, by a number from 0 to 15, 0 meaning a null intensity, and 15 the maximum possible intensity.

- a 1 bit field for blinking

0	off
1	on

- a 1 bit field for light pen sensitivity

0	off
1	on

2.4 Metalanguage definition

A COBOL - report like meta language is used in the examples because of its readability, as well in the beads as in the structure definitions.

Symbol	Meaning
+	concatanation
{ }	choice
[]	optional choice
{ } _{l<n<u}	repetition
	l lower bound on the number of identical items; if omitted l is assumed to be 0.
	u upper bound on the number of identical items; if omitted:

- : label for further use within the same rule
- = assignment
- = > conditional alternative
- () grouping
- ' ' indicates a special value given to the following field name

- ⊕ plus
- ⊖ minus

2.5 Proposed standard beads

2.5.1 Alphanumeric beads

Character: CHAR

A character is composed of one eight bit field (which has the same name). Many special patterns, corresponding to currently used special characters are defined; they are indicated in table 2.5.1, as well as some subsets of CHAR. The basic character code is declared as standard ASCII

standard EBCDIC

or by the name

CODE

followed by the 128 characters in this code corresponding to the 128 ASCII characters. If no code declaration is specified, the ASCII code is assumed by default.

Number representation

Normally the kernel of a program stays in the server's HOST and the user's HOST should have no arithmetic operations to perform on the data. In this case, the principles involved in the arithmetic unit conception of a HOST do not need to be described. But the format of fixed and floating point numbers has to be described.

- in the case when user and server HOST's have the same number representation, for instance the standard representation, the transmission of data in their number representation reduces the data flow between them.

- if the server HOST has a different number representation than the standard representation, depending on the data transmitted, there are two alternatives:

These properties will be expressed by special field definition:

$$\text{EXP} \leftarrow \text{SBYTE} \oplus '40\text{H}' \text{SBYTE}$$
$$\text{MANT} \leftarrow \text{SIGN} \oplus \{\text{BYTE}\}^3$$

and a floating point number is defined as:

$$\text{FLP} = \text{MANT} \cdot 2^{\text{EXP}}$$

1. Special Characters

Transmission Control Characters

SOH
STX
ETX
EOT
ENQ
ACK
DLE
NAK
SYN
ETB
ESC

Printer Control Characters

horizontal tabulation	HT	+	'0X1'	CHAR
vertical tabulation	VT	+	'0BX'	CHAR
new line	NL	+	'0AX'	CHAR
end of message	EOM	+	'08X'	CHAR

Teletype Control Characters

Carriage return	CR	+	'0DX'	CHAR
shift out	SO	+	'0EX'	CHAR
shift in	SI	+	'0FX'	CHAR
	BS	+		

Device Control Characters

DC1
DC2
DC3
DC4

Table 2.3.1

2. Subsets of Characters

Numeric characters		{ 1 2 . . . 9 }	CHAR
NUM	←		
Printable characters		{ NUM ALPH = = }	CHAR
PRCHAR	←		
Intermediate characters		{ 'characters' in column 2 }	CHAR
ITCHAR	←		
Final Characters			
FIN CHAR	←	CHAR ⊖ ITCHAR	
Transmission Control Characters*		{ 'NUL' . . . DEL }	CHAR
TRACHAR	←		
Derra Control Characters			Teletype control character
DCCHAR	←	{ 'DC1' DC2 }	CHAR TYCCHAR { CR SO }

Alphabetic characters

ALPH ← { A
.
.
Z } CHAR

Printer Control Characters

PCCHAR ← { HT
VT
NL
EOM } CHAR

Table 2.3.1: Special ASCII characters and groups of ASCII characters.

*see USACII standards

2.5.2 Graphic Beads

As proposed in RFC #5 by J. Rulifson, the screen of any graphical display is taken to be a square; the coordinates of points are normalized from $-1/2$ to $+1/2$ on both axes. The position of the first point of a structure is determined by the deflection from the origin which is the rest point of the beam; following points are determined by their deflections (AX,AY) from the last beam position.

Thus, only two data fields need to be defined:

DEFLECTION which is a 12 bit field: the deflection is defined by a number between -1 and $+1$ with the precision usual to the server.

ANGLE which is a 15 bit field defining an angle from 0 to 2π in radians between the horizontal axis and an axis passing through the origin. The first bit of it indicates if the angle must be taken clockwise or counterclockwise.

The data beads are:

MOVE

Depending on the parameters which are set when this bead appears, MOVE may specify:

- an invisible movement of the beam; in this case the beam intensity is null
- a new point: in this case the beam intensity is on only when the beam has reached the new point.
- a vector: in this case the beam intensity is set to a certain non zero value

MOVE ← {DEFLECTION}²

Arc of circle: ARC

An arc of circle is defined by its center, followed by its starting point and the angle of its ending axis.

ARC ← {DEFLECTION}⁴+ANGLE

2.7.2 Standard structure type.

Two basic standard structure types are chosen

VECTOR to represent sequence of data beads (strings, arrays, tables...)

PLEX to represent any kind of directed graph, tree, ring...)

VECTOR (C;N₁,...N_C) ← VECTORHDR + VECTORBODY

VECTORBODY ← (=C+1:{defined bead}^{N_i} + [VECTORBODY]

VECTORHDR ← 'VECTOR' IDENTIFICATION + C + N₁ + N₂ + ... + N_C
C is the number of parts (columns) in the vector, each part having N_C elements.

It is also probably interesting to define a compressed vector COMPVECTOR in which sequence of the identical elements are transmitted as 1 element + a special bead + the number of identical elements in sequence.

PLEX (M)

The first bit of a pointer field indicates if the pointer points to a terminal element or not. If it is the case, forward pointer fields are not added to the data element.

M is the number of data elements in the structure.

2.8 Objects

2.8.1 object definition.

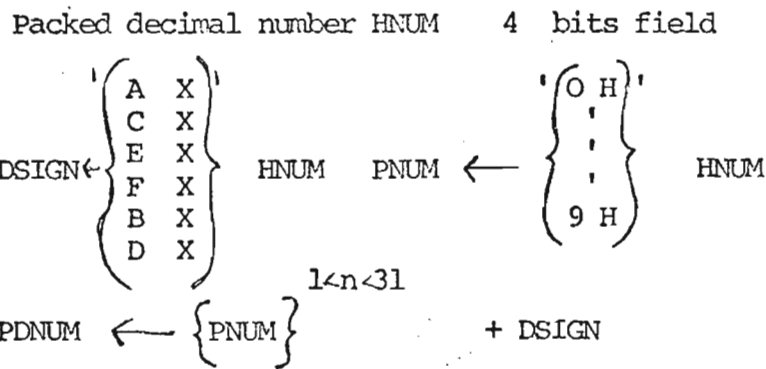
An object is defined by a semantic rule including, on the right hand side {
a name to identify the object
a set of parameters of the object definition.
operands: name of the beads used as data elements
on the left hand side {
operators: parameter fields
structure of the data beads.

i.e. The definition of a new object called SQUARE is: SQUARE ←
(A,L;Aθ) ROT(A,ANGLEθ) (VECTOR(1,4) (BEAM'OFF'+
MOVE (A) + BEAM 'ON' + MOVE (0,L) + MOVE (L,0) +
MOVE (0,εL) + MOVE (εL,0))

Where ROT refers to a transformation defined in the data manipulation language, and VECTOR is defined as a standard structure.

- Binary card image

B CARD ← VECTOR (1;120) {CHAR}¹²⁰



- Decimal number (unpacked or zoned

$1 \leq n \leq 31$

$$\text{DNUM} \leftarrow \{ \text{NUM} \} + \text{D SIGN} + \text{PNUM}$$