SETL Newsletter # 146

J.T. Schwartz
March 7, 1975

Adaptation of GYVE/SETLG to Distributed Networks of Computers (Preliminary Proposal)

1. Introduction.

In this newsletter we shall suggest adaptions to SETLG and GYVE which may be appropriate if these languages are to be adapted to run on distributed networks of computers. The word 'network' that has just been used actually covers several different physical possibilities:

a. Several equally fast computers might be able to communicate through a memory which they share at least in part. The peripheral processor / central processor setup on the CDC 6600 would approach this ideal if the processors were closer in speed. Note that one or both of the linked computers may or may not be able to excuse instructions resident in the common part of memory.

b. The memories of two computers might be linked by a cable, and either might be able to start a transfer between the memories, which would then proceed autononously, perhaps delivering an interrupt when finished.

c. Two computers may be linked by a cable, but the cooperation of both may be needed if data is to be transfered between them, especially if it is to be transmitted at an acceptable rate.

The more the 'network' with which one deals resembles case (a), the more it is reasonable to use the existing semantic structure of GYVE without any modifications at all. But we have the more difficult case (c) particularly in mind, and will therefore suggest some restrictions which model typical network behavior but require only very minimal changes to existing GYVE sematics.

Note in all of this that a central question to be faced is the degree to which implementation-level considerations, including handshaking conventions, transmission-error conventions etc., and also issues arising from the fact that communications will always consume at least small amounts of processor time, ought to be made explicit. In the system to be proposed, all these issues will remain implicit, and the computer time required for communication will be absorbed into an invisible 'overhead'.

2. The Notion of 'Machine' or 'Address Space'; Instance Copying; Channel Instances.

In the network model we shall use, the notion 'machine' represents one or more processors, sharing a physical memory, but able to communicate with other 'machines' only over a limited-bandwidth physical communications channel. These 'machines' are the nodes of our network model. If there exists a communication channel between two machines, we say that they are 'adjacent' or 'communicating'. A machine in our model is characterised by its address space, and from an abstract point of view is synonymous with this space. The full global address space of a network-oriented GYVE or SETLG system is therefore subdivided into several address subspaces, each corresponding to one machine. Pointers may be passed anywhere in the system, but each pointer references an object in some specific address space. Accordingly, part of each pointer is an *address space identification field*.

Each process is resident in some particular machine, and can only reference global objects resident in that machine. The only type of instance which is allowed to communicate, except by signals to some port, between machines is a *channel instance*. Instances of this system mode are created at IPL time, and cannot be created subsequently (in this regard, channels are like devices). There will exist one or more channel instances

for each pair of adjacent machines. A channel instance may be oriented (in which case it has an input side and an output side), or bilateral). The semantic rules for a channel instance are as follows:

A channel instance with pointer cp has the entry

(1) cp. copy(inp)set(oup) account(a) result(r)[port(p) priority(i) message(m)]

Here, *inp* is a pointer to a complete global object, of which object a copy with all internal (base) data is to be transmitted, and to which *oup* will point after the transmit operation is complete. The object to be transmitted must be a GYVE packet(possibly consisting of only one global object) available in the memory space of the machine at the input side of the channel instance (this may be either side, if the channel instance is bilateral). After receipt, the packet must be UNPACKed (in the GYVE sense) to expose its contents. A new copy of the transmitted object will be formed in the machine at the opposite side of the channel. The input instance *inp* is not available for [illegible], i.e., is forcibly 'quiesced' in the GYVE sense, during the transmission operation (1).

The transmit-copy operation initiated by the instruction (1) may take substantial time to complete; timeout parameter [illegible] such an instruction is therefore allowed. A transmit operation ties up a channel instance only while it is actually in progress; when it times out, the channel instance becomes available for use by another process.

In our modified, network oriented GYVE each account will belong to some machine, and will give the right to form objects in the memory of that machine. In (1), the parameter *oup* will point to an account belonging to the machine in which the object referenced by *oup* will be formed.

If the optional *port* parameter p is specified in (1), the process executing (1) can be interrupted by a message arriving at p (rather than merely suspended by a timeout) at any time during the copy operation. If such an interruption occurs it will destroy any partial copy which the operation (1) may have formed.

In addition to *copy* operations of the form (1), GYVE ports and SIGNAL operations are allowed to function as interprocess communication mechanisms. If *cp* references a channel instance, and *oup* is a pointer to a port on the 'far' side of *cp*, then a process on the 'near' side of *cp* can execute the statement

(2) cp. signal(oup) message(m) [result(r) priority(pr)],

thereby transmitting an ordinary interrupt message, over the channel, to *oup*. Since it will sometimes be necessary to transmit pointers rather than integer messages between machines, we also propose to generalise the GYVE port notion slightly, introducing a class of *pointer-ports* whose semantic behavior is identical with that of ports except that they receive pointers and not images through their message argument. At IPL time, the top level process which is started in each machine will be passed one pointer port for itself and one additional for each of the machines to which it has a channel (adjacent machines); using these, extra pointers can be created and passed as necessary to support whatever inter-machine communication regimen is to built up.

If the object *inp* is destroyed during the execution of the instruction

(3) cp. copy(inp) set(oup) account(a) result(r),

then we will restore precisely the state which would have been reached if *inp* had been found to be destroyed when this instruction started. If in particular a DESTROYED(L) clause is attached to the instruction (3) then transfer to the label L will be forced.

Note that the account *a* 'charged' for the object *oup* only when the operation (3) completes successfully. Thus no problem arises if (3) fails because *inp* is destroyed.

Note that in the approach which has been described in the preceeding pages communications costs are treated as an 'invisible overhead', which at the implementation level may subtract some capacity from the computers servicing the processes resident at either end of a channel. Then implementation level parameters attached to each node of a network can govern the total part of the node's computing power which is available for communication purposes. This ratio may in turn impact the usable communication bandwidth of the channels attaching to a given node.