A high-level language for use with multi-computer networks

by HARVEY Z. KRIOFF

University of Illinois
Chicago, Illinois

INTRODUCTION

Two basic trends can be observed in the modern evolution of computer systems. They are the development of computer systems dedicated to a single task or user (minicomputers) where the sophistication of large computer systems is being applied to smaller units, and the trend of very large systems that locate the user remotely from the computer and share resources between more and more locations. It is to the latter case that this paper is directed. This trend reaches its culmination in the design of distributed computer systems, where many individual computer components are located remotely from each other, and they are used to jointly perform computer operations in the solution of a single problem. Systems such as these are being developed in increasing numbers, although they are yet only a small fraction of the total number of computer systems. Examples of such systems range from those that are National and International in scope (the United States' APRANET,1 Canada's CANUNET,2 the Soviet Union's ASUS system3 and the European Computer Network Project4), the statewide systems (the North Carolina Educational Computer System5 and the MERIT Computer Network6), to single site systems (The Lawrence Radiation Laboratory Network7 and Data Ring Oriented Computer Networks8). These systems and others have been designed to solve problems in the areas of research, education, governmental planning, airline reservations and commercial time-sharing. Taken together they demonstrate a capability for computer utilization that places more usable computer power in the user's control than he has ever had before. The challenge is to make effective use of this new tool.

Development of this new mode of computer usage has followed the same set of priorities that has prevented effective utilization of previous systems. A large body of information has been collected on the hardware technology of network systems,9 but little effort has been expended on the development of software systems that allow the average user to make effective use of the network. A systematic examination of the requirements and a design for a language that uses the full facilities of a number of computer networks is needed.

NETWORK LANGUAGE REQUIREMENTS

After the language design had begun, it developed that the concept of what a computer network was, and how it was to be used, was not well understood. An effort was therefore begun to classify computer networks and their operations. This classification scheme indicated that a language for computer networks would have to be constantly changing because networks evolve from one form to another. It is this dynamic behavior that makes the design of a single language for different types of networks a high priority requirement.

Types of computer networks

A classification scheme was developed based on the resource availability within the computer network and the network's ability to recover from component failure. The scheme consists of six (6) different modes of network operation with decreasing dependability and cost for the later modes.

1. Multiple Job Threads: This consists of a system where all components are duplicated and calculations are compared at critical points in the job. If a component should fail, no time would be lost. Example: NASA Space Flight Monitoring
2. Multiple Logic Threads: This is identical with the previous mode except peripherals on the separate systems can be shared.
3. Multiple Status Threads: In this mode only one set of components performs each task. However, other systems maintain records of system status at various points in the job. If a component should fail all work since the last status check must be re-executed. Example: Remote Checkpoint-Restart
4. Single Job Thread: In this mode one computer system controls the sequencing of operations that may be performed on other systems. If a system failure occurs in the Master computer, the job is aborted.
5. Load Sharing: In this mode each job is performed using only a single computer system. Jobs are transferred to the system with the lightest load, if it has all...
necessary resources. If the system fails you may lose all record of the job. Example: HASP to HASP job transfer.

6. Star: In this mode only one computer system is available to the user. It is the normal time-sharing mode with a single computer. If the system is down you can't select another computer.

All presently operating networks fall within one of the above modes of operation, and additional modes can be developed for future networks. Operation within the first two and last two modes require no additional language features that are not needed on a single (non-network) computer system. Thus, the language to be described will be directed toward utilizing the capabilities of networks operating in modes 3 and 4. In order to evaluate its usefulness, implementation will be performed on both the ARPANET and MERIT Networks. Both networks can operate in modes 3 and 4, and possess unique features that make implementation of a single network language a valid test of language transferability.

Computer network operations

In designing the language we recognized three (3) types of operations that the network user would require.

1. Data Base Access: Four operations are necessary for this type of usage. The user can copy the entire data base from one computer system to another, or he could access specific elements in the data base, or he could update the data base, or he could inquire about the status of the data base. Decisions on whether to copy the data base or access it element by element depend on data transfer speeds and the amount of data needed, therefore they must be determined for each job.

2. Subtask Execution: Tasks may be started in one of four different modes. In the Stimulus-Response mode a task is started in another machine while the Master computer waits for a task completion message. In the Status Check mode the subtask executes to completion while the Master task performs other work. The subtask will wait for a request from the Master to transmit the result. The third mode is an interactive version of status checking where the status check may reveal a request for more data. The final mode allows the subtask to interrupt the master task when execution is completed. Most networks do not have the facilities to execute in mode four, however all other modes are possible. The novice will find mode one easiest, but greater efficiency is possible with modes two and three.

3. Network Configuration: This type of usage is for the experienced network user. It provides access to the network at a lower level, so that special features of a specific network may be used. Programs written using these type commands may not be transferable to other networks. These type commands allow direct connection to a computer system, submission of a RJE job, reservation of a system resource, network status checking, select network options, access to system documentation, and establishment of default options.

The network language must allow the user to perform all these operations by the easiest method. Since the average user will know very little about the network, the system must be supplied with default parameters that will make decisions that the user does not direct. These defaults can be fitted to the network configuration, the individual user or a class of problems.

User operations must be expressible in a compressed form. Operations that the user performs very often should be expressed by a single command. This will prevent programming errors and it will allow for optimization of command protocol. As new operations are required they should be able to be added to the language without affecting already defined commands.

Additional language constraints

Three of the six constraints used to design the network language have already been described. The list of features that were considered necessary for a usable language are:

1. All purely systems requirements should be invisible to the user.

2. The language should be easily modified to adapt to changes in the network configuration.

3. The language should provide easy access to all features of the network at a number of degrees of sophistication.

4. The language should provide a method for obtaining on-line documentation about the available resources.

5. The fact that the system is very flexible should not greatly increase the system's overhead.

6. The language syntax is easy to use, is available for use in a non-network configuration, and it will not require extensive modification to transfer the language to another network.

These requirements are primarily dictated by user needs, rather than those required to operate the hardware/software system. It is the hope that the end result would be a language that the user would use without knowing that he was using a network.

The demand for on-line documentation is particularly important. Most software systems are most effectively used at the location where they were developed. As you get farther from this location, fewer of the special features and options are used because of lack of access to documentation. Since most of the systems to be used will reside on remote computers, it is important that the user...
be able to obtain current documentation while he uses the system. That documentation should reside on the same computer as the code used to execute the system.

Options used to determine which computer system you are communicating with should not have to exist in interpretable code. This generality often leads to heavy overhead that defeats the advantages of a computer network. An example of this was the Data Reconfiguration Service created by Rand as a general data converter that could be used when transferring data from one computer to another on the ARPANET. Because it was written to execute interpretively, data conversion can only be performed at low speed. While high-speed and low-overhead were not conditions of their implementation, an operational language should not produce such restricted conditions of usage.

The final condition that the language be easily used and operate on a single computer, led to the investigation of available extendable languages. Using an already developed language has certain advantages, people will not need to learn a new language to use the network, program development can continue even when the network is not operating, and network transparency is heavily dictated by the already designed language syntax. It was a belief that a network language should not be different in structure than any other language that led to the investigation of a high-level language implementation.

THE NETWORK LANGUAGE

An available language was found that met most of the requirements of our network language. The language is called SPEAKEASY and it consists of a statement interpreter and a series of attachable libraries. One set of these libraries consist of linkable modules called “linkules” that are blocks of executable code that can be read off the disk into core and executed under control of the interpreter. This code, that may be written in a high-level language such as FORTRAN, can be used to perform the necessary protocols and other system operations required by the network. Thus, the user would not be required to know anything other than the word that activates the operation of the code. Each user required operation could be given a different word, where additional data could be provided as arguments of the activating word.

Since there are almost no limitations on the number of library members, and each user could be provided with his own attachable library, the language can be easily extended to accommodate new features created by the network. Linkules in SPEAKEASY could be created that communicate with individual computer systems or that perform similar operations in more than one system. Where the latter is implemented, an automatic result would be the creation of a super control language. Since one of the factors that prevent the average user from using more than one computer system is the non-uniformity of the operating systems, the development of a network language will eliminate this problem. Using data stored in other libraries, the network language could supply the needed control syntax to execute a specified task. This operation is not very much different from what the user does when he is supplied control cards by a consultant that he uses until it is outdated by systems changes.

The SPEAKEASY language presently provides the facility to provide on-line documentation about itself based on data in attachable libraries. This would be extended to allow the SPEAKEASY interpreter to read from libraries resident on a remote computer. Therefore, the documentation could be kept up-to-date by the same people responsible for developing the executing code.

Since SPEAKEASY linkules are compiled code, and there may exist separate modules that are only loaded into core when needed, minimal overhead is provided by adding new operations to the system. This is the same technique used to add operations to the present system, therefore no difference should be detectable between resident and remote linkules.

NETWORK MODULE PROTOCOLS

Where a single computer version of SPEAKEASY has an easy task to determine how a command should be executed, a multi-computer version makes more complex decisions. Therefore it is necessary that there be established a well defined pattern of operations to be performed by each linkule.

Linkule order of operations

Each linkule that establishes communication with a remote (slave) computer system should execute each of the following ten operations, so as to maintain synchronization between the Master task and all remote subtasks. No module will be allowed to leave core until the tenth step is performed.

1. Determine System Resources Needed.
2. Establish the Network Connections.
   a) Select the Computer Systems that will be used.
   b) Establish the System Availability.
   c) Perform the Necessary Connect & Logon Procedures.
3. Allocate the Needed Computer System Resources to the Job.
5. Determine what Data Translation Features are to be used.
6. Determine whether Data Bases should be moved.
7. Start the main task in the remote computer executing.
8. Initiate and Synchronize any subtasks.
10. Terminate the remote task and close all related system connections.

In the present single computer version of SPEAKEASY only tasks 3 and 7 are executed, and a more limited form of task 4 is also performed. Where the overhead in opening and closing network connections is greater than the cost of leaving the connections open, the 10th task will not terminate any connection once made, allowing the next linkule to check and find it open.

Information on the systems availability can be obtained by inquiry from the network communications software. Procedures and resource information can be obtained from data stored in local or remote data set, by inquiry from the user, or by inquiry from the system. All allocation procedures will be performed by submitting the appropriate control commands to the standard operating system on the relevant computer.

Since the SPEAKEASY system will have the necessary data about the origination and destination computers for any unit of data being transferred, it can link to a linkule that is designed for that translation only. This linkule could take advantage of any special features or tricks that would speed the translation process, thus reducing overhead during the translation process.

Requirements for network speakeasy

The minimal requirements to execute the Network version of SPEAKEASY is a computer that will support the interpreter. At the present time that means either an IBM 360 or 370 computer operating under either the O/S or MTS operating systems, or a FACOM 60 computer system. The network protocols for only two operations are required. The Master computer must be able to establish a connection to a remote computer and it must be able to initiate execution of a job in the remote computer system.

The remote site should provide either a systems routine or a user created module that will perform the operations requested by the Master computer. This program, called the Network Response Program, is activated whenever a request is made to the remote computer. There may be one very general Network Response Program, or many different ones designed for specific requests. Figure 1 shows the modular structure of the Network Language in both the Master and Slave (Subtask) computing systems.

Because the network features of the language are required to be totally transparent to the user, no examples of network programming are shown. The reader should refer to previously published papers on Speakeasy for examples of programming syntax.

SUMMARY

A network version of the SPEAKEASY system is described that consists of a series of dynamically linked modules that perform the necessary language and system protocols. These protocols are transparent to the user, producing a language that has additional power without additional complexity. The language uses attached libraries to provide the necessary information that will tailor the system to a specific network, and to supply on-line documentation. Since the modules are compiled code, the generality of the system does not produce a large overhead.

ACKNOWLEDGMENT

The author is indebted to Stanley Cohen, developer of the SPEAKEASY system. His assistance and support made this extension possible.

REFERENCES
