M.I.T. LABORATORY FOR COMPUTER SCIENCE
Computer Systems Research Division

July 20, 1977
Request for Comments No. ~~345~~ 146

THE CONCEPT OF UTILITY IN A MODEL OF A DISTRIBUTED SYSTEM

by Liba Svobodova

The attached report was conceived during the fall 1976 when we were trying to formulate an informal model of a distributed system to serve as a framework for our research. Since then we have moved on to more specific topics, thus this memo is somewhat out of date. It is being published now mostly so that the documentation of the early stages of our research on distributed systems will be as complete as possible.

In RFC 128, Jerry Saltzer projected two different images of a "distributed system":

1) A system that looks to its users and behaves at its interfaces as though it were a monolithic central system, but is actually engineered from physically or geographically distributed pieces.
2) A system whose external image is "this computer is self-contained, but is prepared to be a cooperating member of a set of similarly cooperating computers".

In the first image, a system, coherent by definition, is partitioned internally into loosely coupled functions that can be physically and geographically separated. The other approach starts with self-contained systems that are prepared to cooperate in a manner that produces an image of a coherent system on a certain defined level.*

To the two discussed images of the nature of the system that should serve as a vehicle for our research, I would like to add a third one, with the necessary acknowledgement that these interpretations are not mutually exclusive. I want to argue that we should not completely reject the notion of a computational "utility" as a special type of node (or a set of nodes) in a distributed system. Thus, the third image is a system where totally self-contained computers can be, if desired, connected to a large utility. This may look like another version of "distributed Multics", but I believe that this model has more general application and appeal.

The utility <u>extends</u> the functional capabilities of individual computers connected to it. We may talk about an <u>information</u> utility that stores, maintains, and retrieves shareable information, or a <u>processing</u> utility that provides services requiring special hardware (and related software) that cannot be justified to be a part of individual member systems. The function of the information utility is to supply information (programs or data) to requesting system members, whereas the function of the processing utility is to process information (programs or data) supplied by the system members.

(*) While the latter approach implies integration rather than distribution, the suggested name "Research in Integrated Systems" (RFC 128) does not fully capture the essence of this approach. A "system" is integrated by definition: a system is "a set of arrangements of things so related as to form a unity or organic whole" (Webster's Dictionary). What we are really addressing is research in "integration of computer systems".

The purpose of the information utility is to manage information used or usable by all or a majority of system members. This may include software tools such as language processors and text processing and formatting programs, libraries of scientific programs, or general information data files such as a Library of Congress index, encyclopedic data, or general information specific to the application for which the system members agree to cooperate. The information utility is an important support to members with a relatively small amount of information storage. On the other hand, individual members may decide (if permitted) to maintain copies of frequently used utility programs and data files in their own storage. However, the utility is still important as a backup and as the source of new versions of programs and data stored by the utility.

It can be argued that to handle the use of information in the described way, it is not necessary to have special utility nodes. The utility functions described above could be distributed over the individual system members. However, to get hold of information that belongs to a specific member of the system, this member must be:

1) operating
2) connected to the system
3) willing to cooperate.

While the member may be willing to share some specific information at any time, it may want (and, to maintain true autonomy, must be allowed) to disconnect itself completely from the system, possibly for the purpose of doing some sensitive work. But if the information has been placed into a utility-like repository, it can be used by other members without having to wait for the owner to start cooperating. Further, techniques for achieving robustness of a shared distributed data base assume a degree of trust among the distinct nodes supporting the data base that may not be achievable if all nodes are allowed to retain full autonomy.

The purpose of the processing utility is to manage sharing of computational resources needed but not owned by individual system members. Included in the processing utility might be special processors such as an FFT module, or a super fast processor for computation bound applications (for example, numerical solutions of complex models), or special I/O devices (fast line printers, graph plotters). Also, the processing utility may provide services that do require special software, but not hardware. Such

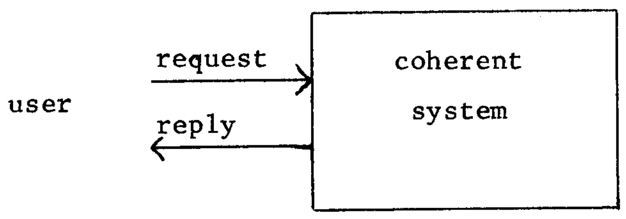

Figure 1: User's view of his self-contained system extended by the utility.

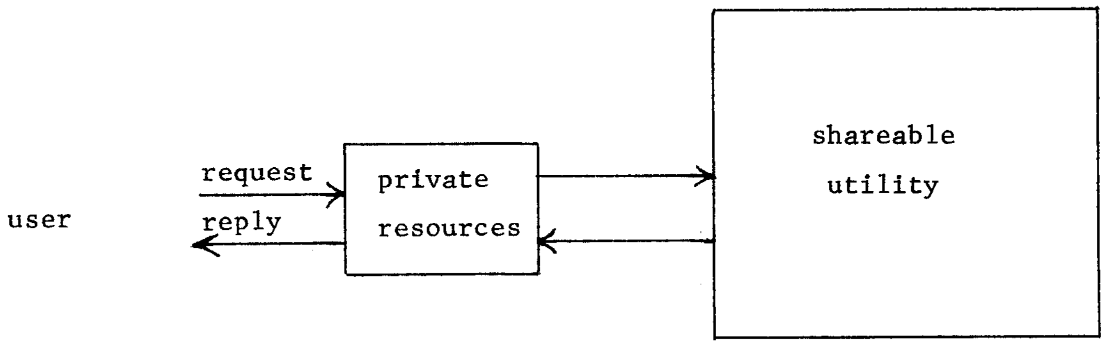

Figure 2: Logical division of resources in the extended system.

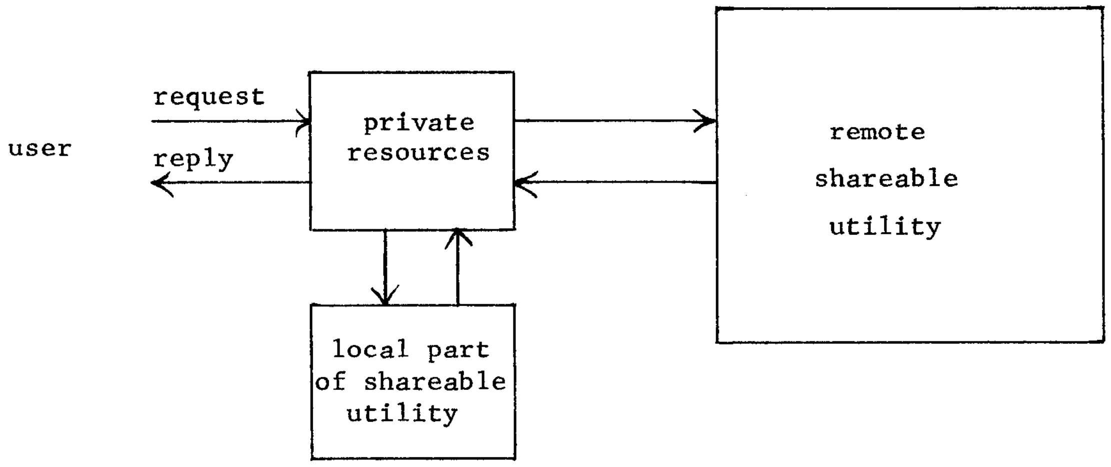

Figure 3: Physical and geographical distribution of the utility.

software could be placed into the information utility, but that means that the members qualified to use it could obtain their own copies. If a program belongs to the processing utility, a system member can only cause an execution of this program, possibly with input data supplied by the requesting member.

There are several ways to view this type of system. The utility was already described as an extension of an otherwise self-contained computer. To the user, this combination represents a coherent system (Figure 1). Logically, however, the functions and services available in this system are divided into those that are private to the system member and those that are a part of independently maintained utility (Figure 2). Further, while the utility is assumed to have the quality of an integral system, it can be both physically and geographically distributed (Figure 3). Multiple copies of files, possibly in different geographical locations, may be necessary to guarantee sufficient bandwidth between the community of the system members and the information utility. The utility has to be robust; this argues for a distributed arrangement. Finally, the physical distribution of the utility can be accompanied by the distribution of control. Consequently, individual parts of the utility could be under the same administration as individual members of the system; the utility would be <u>integrated</u> from contributions (software and hardware) of individual members.

The utility does not necessarily eliminate the need for direct communication between individual members. In general, it should be possible for a member to provide information to other cooperating members without placing it into the repository of the utility. Now we have an image of a two-level* distributed system. One level consists of self-contained cooperating computers, the second level is the modular utility, where individual modules too may be capable of operating as self-contained systems should the need arise (Figure 4). The direct cooperating of system members might be necessary for processing and distribution of dynamically changing information. The utility is more suited as a repository of more static information, but it guarantees high availability.**

(*) This by no means imply hierarchical structuring. The levels as used here are abstractions of two possibly independent, possibly overlapping, or perhaps even inseparable subsystems.

(**) Possible application (?): an airline reservations system (operating as a utility for all airports and ticket offices) combined with airline operations system (direct communication among relevant airports and planes).

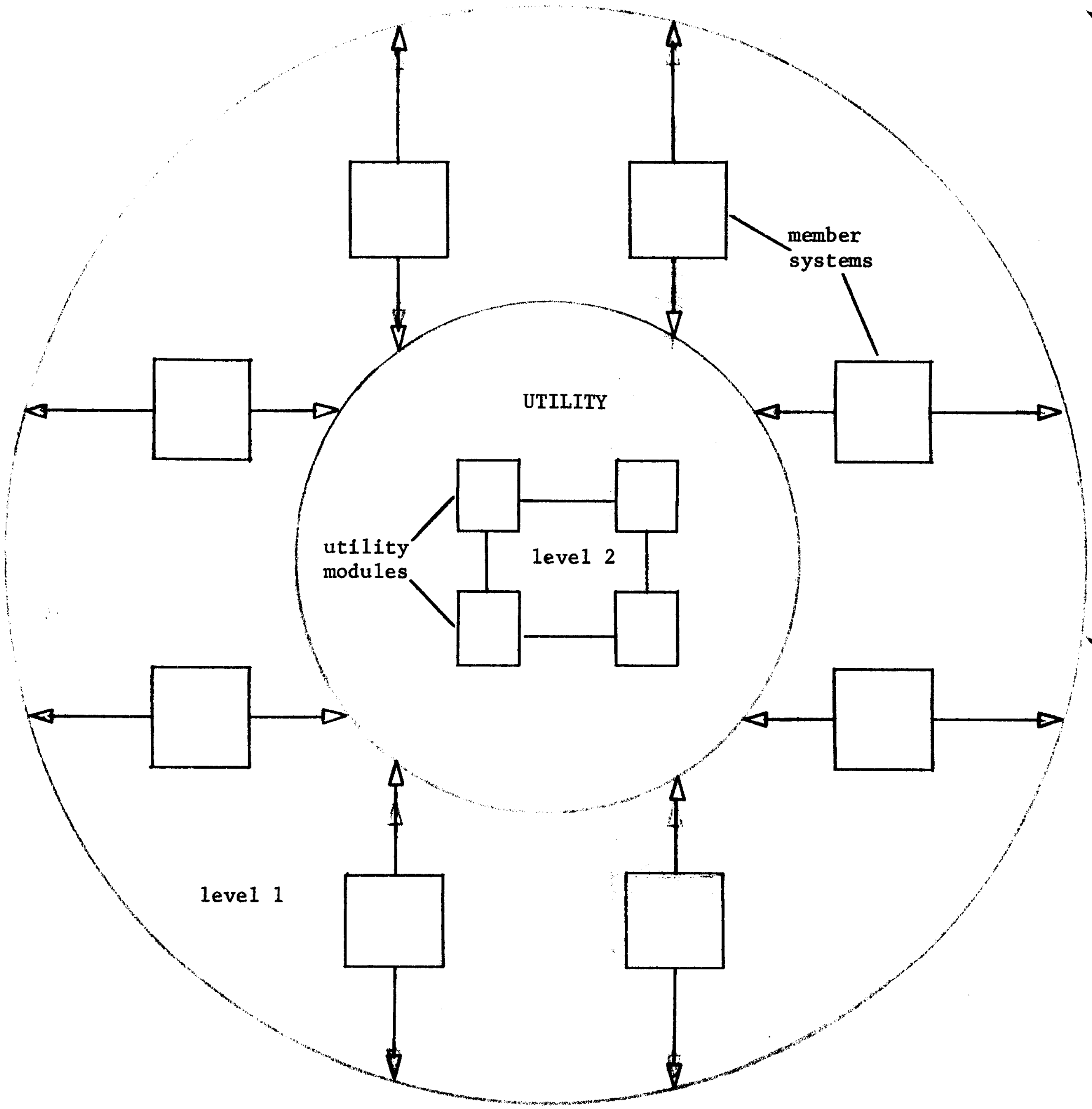

Figure 4: Two-level distributed system:

Level 1: network of cooperating member systems
Level 2: modular utility

In summary, the outlined model of a distributed system consists of two levels with the following characteristics:

- The utility that looks like a single coherent system, but exhibits functional distribution in structure.
- The network of self-contained systems that may agree to cooperate in creating a coherent system at some level but will not lose their autonomy by joining such effort. In addition to direct communications among members, each member can independently use the utility.

The utility concept in a distributed system addresses the important problem of availability, as already mentioned in connection with the information utility. The utility stands for those facilities that will be available to the system members independently of which system members are in the cooperating mode (that is, willing to communicate information to and perform tasks on behalf of other members) at any point of time. Thus far, the utility has been assumed to represent a clearly identifiable and separable compound of hardware and software. Functionally, however, the utility is an abstraction that, when implemented, might be unseparable from what was described in the preceeding paragraph as level 1. The two-level model is offered here as a means of focusing attention on the fact that different parts of a system may be used differently and will have to conform to different rules. Specifically, the level of cooperation supported by system members might be different from (more limited than) what is required from the utility. Similarly, the cooperation rules in the network of self-contained member systems do not have to be uniform. That is, the network could consist of clusters governed by different rules. In this sense, the utility is just a special cluster.

Distribution and Coherence

Two very important notions permeate discussion of what is and how to design a distributed system. One is the "level of distribution", the other is the "level of coherence". By "level of distribution" I do not mean the physical distances, but the type and character of modules that are to be the basic building blocks of the system. Level of coherence then is the level at which the system modules can communicate and cooperate without having to consider their internal differences.* In the discussed model, the modules are collections of hardware and software that are either completely self-contained, or at least highly autonomous (utility modules). The level of coherence desired in the system may be inherently defined by the type of modules used. However, in a system where the modules are self-contained computers, coherence can be achieved by software and the level is therefore variable.

The most primitive level of coherence in a network of self-contained systems is the exchange of messages in the form of bit strings. A slightly more advanced level is where it is also possible to communicate the semantic content of inter-system messages. The next level then deals with the naming and the use of services. Possible schemes of interest are:

a) programs and data from several different system members or from the information utility can be combined to form a single computational job executed using the hardware resources of the requesting member,
b) a system member executes a computational job partly on its own hardware, and partly by using the services of the processing utility.
c) several system members (with appropriate programs and data) can be involved in an execution of a single computational job.

Though the experience has proved that it is possible to achieve some level of coherence in a network of systems that were not originally prepared for it, it is clear that more involved cooperation requires preparation affecting both the software and the hardware design. An interesting question is what features are critical to what level of coherence and what are the permissible variations in the hardware and software of future members of a distributed system. It should be noted that compatibility of all system members on the machine programming level is not a sufficient prerequisite for building a distributed system. In fact, it does not have to be a prerequisite at all, provided that programs and data can be sent to a system member in some (possibly higher level) form that the member understands.

(*) Note that according to this definition, practically any computer system can be claimed to be a distributed system, and unfortunately, this is what frequently happens. However, since "distribution" can be applied in many different ways, what we see as a misuse of the term "distributed system" or "distributed processing" is probably unavoidable. Thus rather than assuming that everybody has the same or at least a similar image of distributed system, in any research concerning distributed systems, it must be stated explicitely what the level of distribution and the level of coherence are.

Now, a preparation for future integration into a coherent distributed system is not just a question of the initial design. The possibility of diversifying member systems is important to supporting the sense of local autonomy. However, unless the owners of computer systems clearly understand what features make future integration possible, local "improvements" may override the initial preparation at any level above the level of hardware compatibility. Though this is more a sociological issue, it does require recognition and support on the technological level.

Protection Considerations

It is widely believed that distributed systems provide a better environment for maintaining privacy and enforcing protection rules. In support of this claim, two factors should be considered:

1) the actual physical separation of independent or loosely coupled processes, and
2) as a possible side effect of such separation, a reduction in the software complexity.

A distributed system facilitates physical separation of processes that might be mutually harmful, yet allows interprocess communication. A distributed system may be arranged such that each physical module supports only cooperating processes with no conflict of interests. Potentially harmful processes do not communicate through shared memory, but through hardware interface; the possibilities of interaction reduce to those supported by the interprocess communications protocol.

The functions of an operating system can be divided into three categories:

- create an effective and convenient environment for the user,
- supervise allocation of the system resources to the system users,
- logically isolate and protect the processes and the proprietary information of individual system users.

Sharing in a computer system takes place on two levels: information sharing, which is an important part of the user environment, and hardware resource sharing (including sharing of copies of programs and data in the main memory) which in general is imposed by economy considerations. Functions of the second and the third class handle mostly this second type of sharing. Distribution reduces the level of hardware resource sharing, and consequently may reduce the complexity of software for resource allocation, scheduling, and protection. Lower software complexity then could make the system provably secure.

The question of the extent to which the discussed type of distributed system simplifies the protection problems or enlarges the set of solvable protection problems cannot be answered without thorough research, but some possible trends can be outlined.

As the most severe measure, a self-contained member can be guaranteed privacy by physically detaching it from the rest of the system. While a member is connected to the system, its local supervisor is fully responsible for protecting private information from the other members in the system. A member system does not have to allow any foreign programs to execute on its hardware. Private information of member A can be obtained by requesting member B only if the supervisor of member A agrees to and arranges for a transfer of the requested information to member B. Of course, the supervisor of each member system should be verified to ensure that no request from another system member can upset the receiving member.

Now we can envision several different situations. Each self-contained member may be a personal computer, a shared computer which is dedicated solely to one person at a time, a shared uniprogrammed computer, or a multiprogrammed computer. In the first case, all information stored in a computer belongs to the same person, who is the only agent who potentially has direct access to this information. The second case is similar - a user can have his own disk pack, that, when connected to a compatible computer system, turns this system into a personal computer. Any meaningful cooperation with this type of system can be difficult, though, since not only the member system must be operational, but it must be being used by the right user. Here the concept of utility gains in importance. In the uniprogrammed system, the information storage media are shared by several users. Each of the users may potentially gain access to all information stored in his member system if not prevented by the protection mechanism. In a multiprogrammed system, the possibility of protection violation is increased because of coexistence of programs and data belonging to different users in primary memory. However, it can be anticipated that even if individual members are multiprogrammed, their correctness may be verifiable, since they will be much smaller and much less complex than present computer systems that have to serve large communities with diverse needs and interests.

The utility is responsible for protection of its own resources, including information entrusted to it by individual members of the system. A simplifying situation is a utility that does not have to execute programs supplied by individual system members. However, execution of members' programs may be a type of service desired from the utility. An information utility provides services in the form of information retrieval, deposit, and modification. A retrieval may result in a transfer of a very large file where only a few specific data items from this file are actually needed. It may be preferable, and sometimes even necessary, to perform the selection of needed data within the utility. Since the selection algorithm could be user dependent, the utility would have to execute programs supplied by individual members on its own hardware. A processing utility also provides services in the form of executing special programs on special hardware with data supplied by the requesting member. Undoubtedly, there will be situations when it will be desired to run programs developed by individual members on this special hardware.

Thus, the utility as a whole has to be able to run both the trusted utility programs and potentially harmful programs supplied by individual members. Within the utility, however, execution of these two types of programs can be physically separated. Further, the same type of arrangement, protection by physical isolation, may be also desirable within the class of utility functions. The resulting model of a utility is then a collection of functional modules where each module is a distinct physical entity (a processor with its own memory and appropriate software) dedicated to a specific class of functions.

Summary

This memo has outlined several ideas that I feel might be worth investigating. First is the idea of the utility as a facility functionally distinguishable from the member systems. The utility does not have to be a centrally controlled system. In fact, the parts of the utility could be controlled by individual system members. The distinction between utility and member to member cooperation is drawn on the basis of availability. As stated earlier, the utility stands for those facilities that will be available to the system members independently of which system members are in the cooperating mode. It is not clear, however, if this abstract two-level view of distributed processing has a deeper significance, in particular, if it is necessary or at least useful to distinguish these two levels in a design of a distributed system.

The distribution of processing tasks over the nodes of a distributed system is a problem that has not yet received sufficient attention. Several possible types of operation were outlined earlier. In one situation, all necessary programs and data are assembled and processed in one node. In another situation, programs involved in an operation are executed at their local sites. These two situations are of course extremes - a single operation can combine both modes. The factors that effect the feasibility or preferability of a particular mode include performance and security. In the performance category, it is necessary to consider the processing capacity of a node and the time needed to assemble all necessary programs and data, as opposed to the overhead of synchronizing programs being executed at different nodes and the processing capacity of nodes that are requested to serve several member systems simultaneously. A utility program may be too large to be executable by a member system, and so popular that its execution on a utility node will represent a bottleneck. It may be possible to consider a strategy such that part of the processing is done locally by the member system.

Another topic for research is the claim that proper distribution is an effective mechanism for achieving protection. Finally, it has been observed that a functional distribution reduces the design cost as well as the cost of maintenance and growth. This issue should be further investigated, with the goal of developing a design methodology that specifies the required modularization on the hardware level.