Laboratory for Computer Science
Computer Systems Research Division

Request for Comments 201
November, 20 1980

CSS Proposal to DARPA for 1981-82

by David P. Reed

Attached is the section of the 1981-1982 LCS proposal to DARPA that relates directly to the Computer Systems Structure group. It is not particularly detailed, but gives a sense of where ARPA thinks we are going over the next two years.

1. Building Blocks for Distributed Systems

1.1 Introduction and Objectives

The principal objective of this research is to develop and build an effective substrate for the construction of distributed application systems. This substrate includes *subsystems* common to many applications, such as a distributed data storage system, and *tools* useful for the construction and maintenance of applications, such as debugging programs for a distributed environment.

As we discussed in the preceding section, there are certain significant advantages and attractions for developing and using distributed programs, namely: (1) better resource utilization; (2) increased availability; (3) simpler protection; and (4) natural extensibility.

Achieving these advantages requires constructing applications in a completely new way. The application program faces a much more dynamic environment -- including concurrency, failures, changing system hardware configuration, on-line installation of software, and changing protection constraints -- than is normally found in a centralized application. These dynamics make construction of distributed applications qualitatively different than construction of centralized applications.

The construction and maintenance of centralized applications is aided by a wide variety of software tools and subsystems commonly found on any modern time-sharing system such as file systems, debuggers, linker loaders, synchronization mechanisms, protection mechanisms, and command languages. These tools and subsystems, while still helpful, are much less effective in constructing decentralized software. New tools and subsystems will be required for constructing these new applications.

Our approach to decentralized computing is derived from earlier work at MIT on the structure of distributed systems (14), and complements work on distributed system semantics described in the preceding section. Indeed, there is a shared view and strong cross-coupling of the efforts proposed in Sections 4 and 5 of this proposal.

1.2 Background and Technological Need

Our proposed effort under this section falls into four areas -- a decentralized data storage system, a debugging and monitoring system, a naming mechanism to bind decentralized applications together, and a sample decentralized application that uses these building blocks. Relevant background for these four areas is discussed in the following subsections:

1.2.1 Decentralized Data Storage

In many applications where the data involved is naturally decentralized, there is still a strong need for coordination among the nodes. Such coordination is needed, for example, to control concurrent transactions involving data managed by multiple nodes and to orchestrate recovery from failures that impact on such transactions. In addition, aggregates of interconnected advanced nodes (Section 2) exhibit a need for shared data storage servers that provide reliable coordination and longterm storage of information. Although there have been many suggested mechanisms for coordinating synchronization and recovery in a decentralized environment, these mechanisms usually assume that the data accessed by a computation is known in advance -- that is, the entire data configuration is known at the time the application is constructed. For example, the SDD-1 project has developed a technique that requires preanalysis of all transactions that operate on a distributed data base.

The difficulty of such approaches may be illustrated by an example. Suppose that a distributed application involves interconnection of two autonomous inventory control systems, belonging to independent organizations A and B. Ordering an item to be moved from A to B requires a transaction to act at both A and B. Yet if A and B are autonomous, the actions taken at A may be unknown to the implementor of B, and vice versa. Proper synchronization and recovery from failure in such multi-node transactions must be achieved without knowing the implementation details of the autonomous subsystems.

Reed has proposed an approach (1) (2) that works to coordinate synchronization and recovery of data in autonomous subsystems. If each node stores its local data in a data storage system that follows the protocols suggested by Reed, then application transactions on each node that manipulate their own local data can be easily combined into larger, multi-node transactions. Synchronization and recovery of these multi-node transactions is automatically provided by the mechanisms of the individual data storage systems.

Although data is naturally decentralized, the economics of secondary storage devices are such that reliable, low-cost, long-term data storage is still best provided as a shared resource. This naturally leads to the concept of *repository* nodes -- special nodes whose job is to provide reliable, low-cost, long-term storage of data. Such a notion is exemplified by two systems produced at Xerox PARC -- WFS (15), and Juniper (7).

Our approach of coordinating access to distributed data and shared data storage servers is embodied in our proposed SWALLOW distributed data storage system (13). The SWALLOW system is intended to support a wide variety of decentralized applications and is further explained in Section 5.4.

1.2.2 Debugging and Monitoring System

Debugging a decentralized application is qualitatively different from debugging a program in a centralized system. This difference results from at least the following new effects present in decentralized applications:

1. Concurrency: Each piece of a decentralized application operates concurrently, so that debugging tools oriented towards single-stream computations are not very helpful.

2. Autonomy: The decentralized applications are often built incorporating prexisting application subsystems. These subsystems may present an opaque interface to the application programmer who is debugging his application.

3. Failure Recovery and Time-outs: Failures of components are often detected by time-out mechanisms. Debugging facilities, such as the use of breakpoints, may interfere with the real-time behavior of a system, provoking time-outs where none would otherwise occur. Debugging facilities must also be capable of exercising the failure recovery mechanisms of the application, hence they must be able to simulate failures.

Our approach to debugging and monitoring decentralized applications is based on interposing a debugging and monitoring system (DAMS) on the communication mechanism that links the nodes of a distributed system. This debugging and monitoring system will be able to observe, to modify, to delay, to delete, and to introduce messages between the nodes participating in the application.

The DAMS must be able to control the apparent progress of time in the nodes of the system, so that time-outs do not occur as a result of debugging. It must also be able to simulate a variety of failures and operating conditions such as lost packets, variable network delays and node response times. Finally, the DAMS must present information about the system at a variety of levels of abstraction -- the debugging user may wish to see information in packets as bits, or grouped together as high-level messages. Similarly, interactions among nodes, such as remote procedure calls, might be viewed as multiple messages, or as single abstract computation steps, depending on the level of desired detail.

Although there has been much work to date on debugging systems, very little of it is applicable to decentralized debugging. The inspiration for our work comes from two sources -- the METRIC system developed at Xerox PARC (9) and the virtual machine emulator developed at IBM San Jose Research Laboratory (6). Strictly speaking, neither of these is a debugging system. Instead they are systems that support the monitoring of concurrent system processes.

1.2.3 Naming Mechanisms

The naming mechanisms of any computing system are an important part of the "glue" that holds applications together. For example, directories provide mechanisms for grouping parts of an application together, while the name resolution mechanism of a linker/loader provides means for inter-module communication.

For decentralized applications, existing naming facilities are much more primitive. Typically, the only system-wide standard names refer to the ports on the network (such as the ARPANET host names). Such names are relatively static over time, and are assigned and managed by a central administrator.

We believe that distributed system applications need a much richer name environment, with the ability to assign and rebind names to application-level entities such as processes, services, data objects, and users. This function can be provided by a standard naming mechanism that is itself a decentralized system. Application builders should be able to assign names and rebind them dynamically without interacting with central management. The naming mechanism must itself be extremely reliable in the face of expected component failures.

The naming facilities in traditional operating systems provide a wide variety of functions. Names are used to link the parts of systems together -- files, program modules, etc. The naming systems in traditional operating systems typically provide facilities that allow some or all of the following (non-exhaustive) set of features:

1. Autonomous Name Definition: multiple independent users can invent and use names.

2. Dynamic Name Definition: names can be (re)defined while the system is running.

3. Multiple Naming Contexts: user-dependent name definitions are allowed so that substitutions made by one user do not affect all users of the naming system.

4. Translation of Long-term Names to Efficient Short-term Names: such as the conversion from the name of a module to its virtual memory address by a linking loader.

These issues are further discussed in Saltzer (3) in the context of traditional operating systems. Each of these features has an analog in distributed systems, but the distributed environment is different enough that the traditional mechanisms do not generalize.

Comprehensive programming systems tools such as program libraries (for example, the Multics library system (3) or the CLU library system (4)) use rather sophisticated naming facilities in their implementation. Building such a program library function in a distributed system will require careful attention to the naming mechanisms in order to provide the autonomy and reliability possible in distributed systems.

Natural implementations of naming mechanisms in a distributed system often include "name server" nodes that maintain the name-object mappings. But the server is only part of a larger system. The patterns of name use in the larger system will determine the functionality of the server.

We would like to develop naming mechanisms for use by distributed applications that are at least as flexible and powerful in the distributed system context as are the naming mechanisms of Multics in the context of a single mainframe. Just as the naming mechanisms of Multics simplify the construction and testing of applications, the naming mechanisms of the distributed system should be the fundamental building blocks for distributed applications.

1.2.4 Decentralized Application Support

To date there have been very few decentralized applications built. Those that do exist have relatively simple functionality, e.g., message systems, airline reservation systems, and banking transaction systems. Construction of these systems has typically involved substantial effort because they had to be engineered from the ground up.

We believe that with the proper tools, we should be able to construct more ambitious decentralized applications with greater ease. Such applications might include distributed administrative information systems and command and control systems that are more than straightforward enhancements of message systems. To that end we would like to build a small prototype of a decentralized application that tests and exercises the tools that we develop.

1.3 Accomplishments

In 1977 and 1978, we developed the mechanisms for coordinating accesses to data in decentralized systems. During 1979 and 1980, we began work on the concepts of the SWALLOW distributed data storage system and on the specification and construction of an initial prototype implementation on several interconnected ALTOs.

During the course of this work we focused our attention on the performance of the shared data storage servers (repositories) of SWALLOW. Our efforts during 1980 resulted in a design of the server that uses *append-only* random-access storage devices for its stable storage. Such storage devices can be naturally implemented by write-once optical disk storage technology. By the end of 1980 we expect to have completed construction of a prototype repository and to have begun using and enhancing it.

1.4 Proposed Effort and Milestones

Our proposed effort for 1981 and 1982 is presented in the following four subsections and represents the four proposed areas of work that we have already discussed.

1.4.1 The SWALLOW Distributed Data Storage System

By the beginning of 1981, we expect to have a working prototype of a SWALLOW repository. This protype will support the basic protocols and will be implemented on an ALTO (16) using magnetic disk storage. In 1981, we will concentrate on implementing the *brokers* of the SWALLOW system; i.e. the software at each node that manages both the local data stored on the node and the remote data stored on remote repository nodes. We intend to construct the brokers on the advanced nodes (Section 2), and perhaps on other architectures, should that seem desirable. Our ultimate intention is to integrate the SWALLOW system with the Extended CLU language (8) (proposed in Section 4) on the advanced nodes.

During 1982, we plan to continue work on the SWALLOW repository, producing a second prototype on the advanced nodes. To that end, we would like to explore construction of a repository that uses random-access, write-once optical disk memory in order to demonstrate both our design and the optical disk technology.

Protection of information in the SWALLOW system is important to that system's purpose and usability. Accordingly, during 1981 we plan to explore techniques and possibly the use of hardware modules toward that end.

1.4.2 Debugging and Monitoring System

During 1981, we plan to develop a system for debugging and monitoring decentralized applications. As noted in Section 5.2.2, this work will concentrate on issues such as timing, monitoring and controlling message passing, thereby providing several levels of detail to the user of the debugging and monitoring systems.

The initial phase of the work consists of the construction of a debugger that can observe and control the message passing activity of a distributed application running on several nodes. Where feasible, we intend to coordinate this work with the work on the ECLU language proposed in Section 4, so that the messages seen by the debugger are viewed on the level of ECLU rather then as packets that are simple strings of bits. Each node implementing part of an application would cooperate with the node containing the debugger, so that all messages are forwarded through the debugger. The debugger then can display the messages, forward them to their destinations with controlled delay, and relate the activity caused by forwarded messages to the causing messages.

The result of the first phase will be a debugging system that allows detailed observation and testing of a distributed application. It will often be the case that the initial system provides information and control that is far too detailed. The second phase of our investigation will involve development of techniques to hide unwanted detail. Our approach will be to look for common patterns in the dynamic behavior of systems. For example, the debugger may be able to recognize retransmission of a high-level set of requests. The user then would not have to deduce that a set of messages are duplicates of previously seen ones. The debugger could then analyze the difference between the first execution and the one resulting from retransmission, presenting the user with the "difference" information.

1.4.3 Naming Mechanisms for Decentralized Applications

During 1981, we plan to investigate naming mechanisms for naming the constituent parts of decentralized applications. We plan to construct a useful naming system based on this research.

This activity closely related to the work proposed in Section 4. The parts of a distributed application, i.e. its modules, its nodes, and its permanent data, will use a common naming mechanism. We will first explore the impact of the autonomy and the dynamics of distributed applications on the semantics of the underlying naming facilities. The result of this exploration will be in the form of a design for a naming mechanism to be used within decentralized applications.

Our next step is the construction of software that implements the design. We intend to coordinate the construction of this software with the research proposed in Section 4. We expect that construction of specialized name server nodes will be necessary to support the naming mechanism in order to reduce dependence on individual autonomous nodes.

1.4.4 Demonstration Application

During 1982, we would like to demonstrate the utility of these system building blocks in the context of a sample decentralized application. The sort of application that we have in mind must have a significant requirement for local autonomy and local tailoring of software. Examples of such applications are:

1. Command and control systems that include interaction of both computers and human operators i.e., something more than a message system. For example, if each such site maintains its own operations data base, an operation at one site may need to perform coordinated actions at multiple sites as part of this command.

2. A scheduling system for meetings, personal calendars, rooms, and so forth. By this we mean a system that assumes much of the burden carried by secretaries in arranging schedules, but where secretaries and principals are still in the loop to handle exceptions. This system calls for a high degree of autonomy and tailoring.

3. A distributed program library system to support the development of decentralized applications, by enabling sharing of subsystems.

During 1981, we would like to select an application and invite DARPA's help toward that end. By 1982, we expect to begin construction of a prototype. Our major effort in 1982 will be oriented around the construction of this prototype.

1.5 Relevance to DARPA/DOD and Technology Transfer

Within the DOD, decentralized systems have a natural application due to the decentralized nature of the organization and its data and computing requirements. New hardware advances such as the development of powerful advanced computing nodes, high performance local area networks, and longhaul packet networks have all enhanced the opportunity and need for such decentralization.

Yet the capability to construct decentralized systems has not keep pace with these hardware advances. Since there are many applications within the DOD that require or will require decentralized solutions, it is important that tools and subsystems be developed that aid in the construction of such applications.

The SWALLOW distributed data storage system, for example, can be a key component in any application where access to decentralized information is important. Similarly, the debugging systems that we plan to construct are expected to have wide applicability in the construction of many different kinds of decentralized systems.

Transfer of this technology to the DOD is likely to occur in two ways. First, the systems that we construct will be usable on the advanced nodes. Second, the techniques that we will develop are expected to be useful in a wide variety of application environments.

1. Reed, D.P., "Naming and Synchronization in a decentralized computer system," Ph.D. dissertation, MIT Dept. of Electrical Engineering and Computer Science, September 1978. Available as MIT/LCS/TR-205.

2. Reed, D.P. "Implementing atomic activity on decentralized data," Presented at ACM Search Symposium on Operating Systems Principles, 10-12 December 1979, Pacific Grove, California.

3. Saltzer, J. "Name binding in computer systems," in Computer System Engineering, Draft notes for MIT Subject 6.033, MIT, Cambridge, Ma. revised July 1979.

4. Liskov, B. et al., "CLU reference manual," MIT/LCS/TR-225, MIT Laboratory for Computer Science, Cambridge, Ma., October 1979.

5. Bernstein, P.A., Shipman, D.W., Rothnie, J.B., and Goodman, N. "The concurrency control mechanism of SDD-1: A system for distributed databases (The general case)," Technical Report CCA-77-09, Computer Corporation of America, Cambridge, Ma., December 15, 1977.

6. Canon, M.D., Fritz, D.H., Howard, J.H., Howell, T.D., Mitona, M.F., and Rodriguez-Rosell, J. "A virtual machine emulator for performance evaluation," Communications ACM 23, 2 (February 1980), 71-80.

7. Israel, J., Mitchell, J., and Sturgis, H. "Separating data from function in a distributed file system," Proceedings of the Second International Symposium on Operating Systems, France, October 1978.

8. Liskov, B. "Primitives for distributed computing," Proceedings of the Seventh ACM Symposium on Operating Systems Principles, Pacific Grove, California, December 1979, 33-42.

9. McDaniel, G. "METRIC: A kernel instrumentation system for distributed environments," Proceedings of the Sixth ACM Symposium on Operating Systems Principles, Purdue University, November 1977, 93-100.

10. National Bureau of Standards. Data Encryption Standard, Federal Information Processsing Standard Publication 46, 1977.

11. Reed, D.P., "Implementing atomic actions on decentralized data," Proceedings of the Seventh ACM Symposium on Operating Systems Principles, Pacific Grove, Ca., December 1979, 163. Submitted to Communications ACM.

12. Reed, D.P. "Naming and synchronization in a decentralized computer system," Ph.D. dissertation and MIT/LCS/TR-205, MIT, Laboratory for Computer Science, Cambridge, Ma., September 1978.

13. Reed, D.P. and Svobodova, L. "SWALLOW: A distributed data storage system for a local network," submitted to International Workshop on Local Networks, sponsored by IBM Zurich Research Laboratory, August 1980.

14. Svobodova, L. et al. "Distributed computer systems: Structure and

semantics," MIT/LCS/TR-215, MIT, Laboratory for Computer Science, Cambridge, Ma., March 1979.

15. Swinehart, D., McDaniel, G., and Boggs, D. "WFS: A simple shared file system for a distributed environment," Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 9-17.

16. Thacker, C.P., et al. "Alto: A personal computer," to appear in Computer Structures: Readings and Examples, Sieworek, Bell and Newell (Eds.), McGraw-Hill, 1979.

17. Ward S. and Terman, C. "An approach to personal computing," invited paper, COMPCON Spring 80, IEEE Computer Society International Conference, San Francisco, February 1980, 460-465.