The Cedar Project

Jim Morris

March 31, 1980

Background and Goals

Xerox plans to develop and market office systems of increasing power and internal complexity. It will be necessary to develop technologies for the solution of a broad range of office problems. Such systems must be composed of separable, tractable components with standard interfaces, yet be perceived by their users as customized and integrated.

To facilitate these plans, the Palo Alto Research Center's Computer Science Laboratory produces prototype systems that provide interactive, personal computing services. Some of these systems are short-lived experiments to test novel ideas, and some are solid systems that are used by many people. It is important that some of these systems receive use beyond the walls of CSL and PARC; this has proved to be a powerful means of technology transfer as well as a way for us to learn more about large, distributed systems. Besides office prototypes we also develop supporting systems such as compilers, operating system components and program verifiers that are of general interest to computer scientists.

At any given time there are around sixty professionals in CSL. Although they are all sophisticated programmers, they have a wide range of goals and styles: system programming, mathematics, hardware design, artificial intelligence, etc. Most of them are permanent employees who can be expected to invest effort in learning how to use a programming system, but an appreciable fraction are visiting scientists or computer science students who can be productive only if they can assimilate the basic system quickly.

The Cedar project is an effort by CSL to produce a new programming environment. The main goal of this new environment is to increase programmer productivity, specifically the productivity of the programmers in CSL over the next several years. This improvement will come from three main sources: a language system that takes more responsibility for certain programming tasks, programming tools that make program development and debugging faster, and a package library that allows programmers to build upon one another's work. To a large extent these facilities are made possible by the increased capabilities of a new series of computers (D-machines) we have designed and built for our research activities. Given the precedent of the Alto system, it is plausible that the D-machines and Cedar will be used by other groups someday.

Acceptance of this specific goal and its immediacy has made us conservative in our designs. In the main, we have restricted ourselves to those ideas which can be understood and put to use by the intended users in a timely fashion. While it is our business as Computer Science

researchers to strive for new and revolutionary ideas, they are not required for Cedar. Indeed, employing the users of Cedar as guinea pigs for such ideas would tend to *decrease* their programming productivity in the time frame of interest.

History

A few years ago CSL began the process of changing its basic hardware from the Alto to D-machines, a larger, Mesa-oriented machine, and the Dorado, a very powerful machine. As these machines began to materialize several people saw the opportunity to make a large stride in our approach to software. Specifically, the increased capacity of the Dorado made it possible to contemplate using much higher level programming languages and more powerful programming tools on a personal computer.

In early 1978 a group began discussing various design alternatives and developed a general consensus on what features a new programming environment should have. The general idea was that we wanted to combine the best features of the Lisp/Smalltalk environments automatic storage management, delayed binding, fast turn-around, and an integrated environment with those of the Mesa language static checking, explicit interfaces, and efficient use of machine resources.

Later that year a larger group met and explored the possibilities of undertaking the effort starting from a Lisp base or a Mesa base. Eventually the choice was made to start with Mesa.

In 1979 a large, distributed design effort was begun on many different topics: user facilities, display packages, language changes, communications, management of large programs, filing, etc. This effort was coordinated by a steering committee and involved a substantial fraction of the laboratory personnel. An external design review was held in September. The organization of the implementation efforts began in the late summer of 1979 and programming started in the Fall.

The Cedar project makes heavy use of certain core software developed by the Systems Development Department (SDD) of the Office Products Division. As starting points for Cedar we are using their implementations of the microcode for D-machines, the Mesa language, and the Pilot operating system. The interim Mesa environment uses the Tools environment heavily. Finally, Ed Satterthwaite, a member of the Mesa group in SDD, has been a key contributor to the Cedar project from the beginning.

Core Projects

The core of the Cedar project consists of four levels rising from the raw hardware of the D

machines to the system used by the system programmers: the microcode, the Kernel operating system, the Mesa language, and User Facilities.

Moving to D-machines

A significant improvement in our programming environment comes from changing our hardware from Altos to D-machines. Not only are these machines larger than the Alto, they support a fully general virtual memory, are better suited to running Mesa, and allow us to use the Pilot operating system. Pilot brings with it a carefully designed virtual memory, a good file system organization, and an improved process mechanism. Making this move will require work to adapt SDD's microcode and operating system to our needs.

The microcode for the D-machines, like the hardware itself, is not actually part of the Cedar project, but is so crucial that we include it for completeness. It implements the Mesa instruction set, performs low-level I/O functions, and implements certain time-critical operations associated with automatic storage allocation.

The Kernel operating system provides the basic facilities of an operating system: processes, virtual memory, and a local file system. It is essentially the Pilot operating system as amended for Cedar's needs. The Kernel does not address the management directories or distributed files very extensively.

Consolidation of Programming Languages

For several years, programmers at PARC and elsewhere have tended to fall into two groups: systems programmers who use compiled, machine-oriented languages (BCPL or Mesa) and programmers who use higher level languages (Lisp, Smalltalk). Locally, the first group produces systems that run on the Alto and receive widespread use by non-programmers. The second group produces prototypes which contain many new ideas, but cannot be used widely because they do not run on Altos or because they run too slowly for a non-author's patience. Although this diversity in programming systems has its virtues, it has some drawbacks:

> The communication of ideas between these two groups is impeded by issues such as the syntax of the respective programming languages.
>
> Maintenance of multiple languages in a changing environment is not cost-effective.
>
> It is hard to produce an integrated system if the only common denominator is the hardware or file structure.

Aside from its technical objectives, Cedar is an attempt to improve this communication without compromising the essential styles of programming and system development of these two groups. There are many factors, technical and non-technical, affecting this objective,

and we have not fully come to grips with all of them.

Having made the decision to start with Mesa and perceiving our design goal to be roughly the least upper bound of Mesa and Lisp, we singled out automatic storage management, delayed binding of type information, and the ability to manipulate programs as data structures as the most important additions. An abstract machine definition is being developed to define a convenient interface for debugging and program manipulation and a primer for Cedar Mesa is being written.

An integrated system of tools

A striking advantage of Lisp and Smalltalk over Mesa is that they have integrated, very powerful programming environments; where Mesa is only a compiler, linker, debugger, and a handful of run-time routines. Lisp provides quick interaction while Mesa provides more thorough static checking at a more stately pace. We hope to create an environment for Mesa programming that provides tools in the spirit of InterLisp and Smalltalk. Of course, there are intrinsic trade-offs between flexiblity and static checking; we seek compromises satisfactory on the D-machines. At a minimum we hope to learn what is intrinsic and what is merely stylistic. The Cedar User Facilities will consist of editing and debugging tools incorporated into an integrated framework based upon some general purpose display and window packages.

Allied Projects

There are some other basic system building activities in Cedar, but they are more open-ended than the core projects, and likely to outlive the Cedar project *per se*.

System Models

This effort is directed at the problem of programming in the large: how to manage a large, evolving system with multiple versions. A new language (resembling a subset of Mesa) has been developed to describe how configurations of modules are put together. An implementation that operates by generating command files and configurations is currently underway.

Data Bases

The immediate goal is to provide programmers with a package that permits the rapid storage and retrieval of tuples in a data base. The system uses Juniper as its filing medium. An initial release for this system is planned for the summer of 1980. Some applications will then be tried; system models may be one of them.

Filing Facilities

The long range goal is to develop a single, integrated file system which spans personal work stations and file servers in a graceful way. Its design includes an attempt to use the work stations' disks as caches rather than as permanent repositories of information. Its only short term objective is to provide a local directory system for the initial release of Cedar.

Problems, Short-falls, and Opportunities

Pickled values

Since Cedar is a one-language environment we have the opportunity to extend the Mesa language type system to facilitate inter-machine communication as carried out via the Ethernet, file storage, and data bases. Earlier work on remote procedure calls addressed itself briefly to the question, but there has not been much thinking about it recently. We need a scheme that permits a Mesa value to be converted to a permanent (pickled) form suitable for transmission to another machine or data base and subsequent incorporation into a running Mesa program.

Persistent Environment

It is nice to be able to save an arbitrary state of a running environment in non-volatile memory from day to day. InterLisp, Smalltalk, and JaM all provide this facility; so there is no lack of models for what we want. Providing a persistent environment calls for the language's run-time system to have a robustness not provided by the current Mesa system. One needs to worry about smashed structures, space leaks, and fragmentation. These problems have been addressed on a application-by-application basis in the past, but now the system must take more responsibility. The compacting garbage collector is crucial in this regard.

Performance Analysis Tools

We need better tools for understanding the performance of running programs. The general philosophy of getting the functionality right and then improving the performance is not credible if the second step cannot be accomplished in a straightforward way.

Sharing of programs and packages

One of the major arguments put forth for static checking, explicit interfaces, modularity, etc. is that they make the sharing and maintenance of programs easier. Also, removing concern for storage management should make many interfaces simpler and more robust. At the moment, however, we do not have a good collection of packages. A major problem we have not yet solved is the management of the information about packages documentation, bug

reports, plans, etc.

The work underway so far on specific packages is in the area of displays, data bases, and caching. There is a long list of other useful packages to be developed (message transmission, forms, history lists, direct access to disk, dictionary service, teleconferencing, audio, direct access to network, string processing, small data base manager, ...).

Remote procedure calls

The ability to communicate between computers in a simple and natural way is the goal of the remote procedure call mechanism. It enables experimentation with various distributed computing regimes. This ability will be addressed by the group working on the kernel.