The Saguaro Distributed Operating System and Related Projects

Gregory R. Andrews
Richard D. Schlichting

Department of Computer Science
The University of Arizona
Tucson, AZ 85721

July 22, 1986

Our research in distributed systems has concentrated primarily on various aspects of the Saguaro distributed operating system. This has included not only design and development of the operating system itself, but also investigations into related areas. These related projects have included the redesign of the SR distributed programming language, and the development of the MLP system for constructing distributed, mixed language programs. These projects are related to Saguaro in that SR is the language being used for the implementation of Saguaro, while MLP is based on the type system developed for use in Saguaro. Both SR and MLP have been implemented. Prototypes of the Saguaro file system and user interface have also been implemented; implementation of the full Saguaro system is underway on a network of Sun workstations.

To characterize this recent work, titles and abstracts of several representative papers follow.

The Design of the Saguaro Distributed Operating System
G.R. Andrews, R.D. Schlichting, R. Hayes, and T. Purdin
IEEE Transactions on Software Engineering, December 1986, to appear

This paper describes the design of the Saguaro operating system for computers connected by a local-area network. Systems constructed on such an architecture have the potential advantages of concurrency and robustness. In Saguaro, these advantages are made available to the user through several mechanisms. One is *channels*, an interprocess communication and synchronization facility that allows the input and output of different commands to be connected to form general graphs of communicating processes. Two additional mechanisms are provided to support semi-transparent file replication and access: *reproduction sets* and *metafiles*. A reproduction set is a collection of files that the system attempts to keep identical on a "best effort" basis. A metafile is a special file that contains symbolic pathnames of other files; when a metafile is opened, the system selects an available constituent file and opens it instead. The advantages of concurrency and robustness are also realized at the system level by the use of pools of server processes and decentralized allocation protocols. Saguaro also makes extensive use of a type system to describe user data such as files and to specify the types of arguments to commands and procedures. This enables the system to assist in type checking and leads to a user interface in which command-specific templates are available to facilitate command invocation.

Mechanisms to Enhance File Availability in Distributed Systems
R.D. Schlichting, G.R. Andrews, and T. Purdin
The 16th Int'l Symposium on Fault-Tolerant Computing Systems, Vienna, July 1986, 44-49

The design of the file system component of the Saguaro distributed operating system is described. The goal of this file system is to enhance file availability in a way that is easy to use yet inexpensive to implement. The logical file system seen by users forms a single tree and file names are location-transparent. However, any file can be placed at the user's discretion in any of the physical file systems. Also, two mechanisms—reproduction sets and metafiles—are provided to support file replication. Together these mechanisms enable a user to set up collections of replicated files and then access them as if they were normal, unreplicated files. Moreover, a file open is guaranteed to succeed if at least one of the copies is available. A prototype implementation of reproduction sets and metafiles on top of Berkeley Unix has confirmed that these mechanisms are also useful in existing systems, and that they are relatively inexpensive to implement.
(The Berkeley Unix implementation is described in a separate paper that has been submitted to *Software—Practice and Experience*.)

An Overview of the SR Language and Implementation
G.R. Andrews, R.A. Olsson, M. Coffin, I.J.P. Elshoff, K. Nilsen, and T. Purdin
Submitted to *ACM Transactions on Programming Languages and Systems*

SR is a language for programming distributed systems ranging from operating systems to application programs. Based on our experience, the language has evolved considerably during the past year. This paper describes the current version of the language and gives an overview of its implementation. The main language constructs are still resources and operations. Resources encapsulate processes and variables they share; operations provide the primary mechanism for process interaction. One way in which SR has changed is that both resources and processes are now created dynamically. Another change is that the mechanisms for operation invocation—**call** and **send**—and operation implementation—**proc** and **in**—have been extended and integrated. Consequently, all of local and remote procedure call, rendezvous, asynchronous message passing, multicast, and semaphores are supported. We have found this flexibility to be very useful for distributed programming. Moreover, by basing SR on a small number of well-integrated concepts, the language is also relatively simple and has a reasonably efficient implementation.

The Evolution of the SR Language
G.R. Andrews and R.A. Olsson
Distributed Computing, Vol. 1, No. 3, July 1986

As a result of our experience, the SR distributed programming language has evolved. One change is that resources and processes are now dynamic rather than static. Another change is that operations and processes are now integrated in a novel way: all the mechanisms for process interaction—remote and local procedure call, rendezvous, dynamic process creation, and asynchronous message passing—are expressed in similar ways. This paper explains the rationale for these and other changes. We examine the fundamental issues faced by the designers of any distributed programming language and consider the ways in which these issues could be

addressed. Special attention is given to the design objectives of expressiveness, simplicity, and efficiency.

Facilitating Mixed Language Programming in Distributed Systems
R. Hayes and R.D. Schlichting
IEEE Transactions on Software Engineering, to appear

An approach for facilitating mixed language programming in distributed systems is presented. It is based on adding a generic remote procedure call facility to each language, and the use of a type system to describe procedural interfaces, as well as data to be transferred between procedures. This type scheme also specifies a machine independent representation for all data. By defining standard mappings for each programming language, the data conversions required for cross-language calls may be performed, automatically in most cases, by active agents that provide the interface between program components written in different languages. When necessary, explicit control of the conversion is possible. A prototype implementation of a system based on this approach has been constructed on a collection of machines running Berkeley UNIX.