Rtools: Tools for Software Management in a Distributed Computing Environment

Helen E. Harrison, Stephen P. Schaefer, Terry S. Yoo

MCNC
Post Office Box 12889
Research Triangle Park, NC 27709

ABSTRACT

We have implemented a software management tool-set which addresses the problems of compilation and distribution of software across multiple architectures in a mixed-vendor distributed environment. This package, which we call *rtools, is* made up of: a central database, a compilation tool, and a distribution tool. We describe the system in detail and look at possible implementations of *rtools* in other environments. The system is currently managing software which spans three operating system versions on two architectures and twenty-two hosts.

Published in Nemeth, Evi and Sam Leffler, *eds*. *USENIX '88* (Proceedings of the 1988 Summer USENIX Technical Conference and Exhibition, San Francisco, 20-24 June 1988) 1988, Berkeley CA: Usenix Association, pp. 85-94.

1. Introduction

Rtools, an automated software management system, relieves the tedious and sometimes difficult chores of recompiling and updating large quantities of software in a heterogeneous distributed environment. An environment that includes a mix of architectures, a large quantity of source code to maintain, and a large number of machines presents problems which are different from those in an environment with a smaller number of machines. There are many approaches to automating software maintenance. We have tried several of them, having greater success with those approaches that did not require modification of the individual software subsystems. In implementing the *rtools* tool-set we strove to improve the overall administration of software and facilitate the handling of object and executable files, while maintaining the integrity of the existing system.

A First Approach Using Makefiles

A first cut at automating the recompile and redistribute process used *rnake*(1)[1]. Makefiles were modified to accommodate compilation for multiple architectures. They also contained targets to distribute executable files to all appropriate machines. A standard template for the enchanced makefile structure was used to aid programmers in managing the large amounts of additional information introduced by the problems of remote compilation and distribution.

To aid in remote installation, we originally wrote *rinstall,* a utility whose operation was much like *install*(1). *Rinstall* was a front end to *rdist,* the Berkeley Unix™ file distribution utility. In addition to the usual facilities of *install, rinstall* allowed one to specify a list of machines upon which to install a file.

Problems

The makefiles used in this approach were complex and difficult to maintain. We had to rewrite every makefile to include the ability to compile and install programs on machines which may have been neither operating-system nor binary compatible with the source machine. For instance, the old makefile for the well known public domain program *shar* was over a hundred lines and 2K characters long for a program with only two C source files and two manual pages. At that size, it supported only two operating systems. The effort required in the recent acquisition of UltrixDn as a supported system exposed many shortcomings in using makefiles as the control point for file creation and distribution.

Furthermore, *rinstall* proved insufficient for our needs. It worked in a linear manner, taking one file and installing it on several machines. *Rinstall* did not take full advantage of the power available in the Berkeley *rdist* utility. It did not check modification dates, nor did it transfer multiple files over the same connection.

What was adequate for an environment of four or five machines running a single implementation of Unix, is not sufficient for our growing environment of over thirty hosts of differing architectures and five versions of Unix.

The Solution

Our objective was to unite our existing tools into a usable package that used the strengths of those tools to the best advantage.

UNIX is a registered trademark of AT&T.
ULTRIX is a registered trademark of the Digital Equipment Corporation.

We recognized that *make* was never intended to be used to control the management of software at the level we required. Its strengths lie in its ability to efficiently and correctly create binary files. To alleviate the inadequacies of *make* for our task, we separated the problem into two distinct tasks: creating binaries and installing them.

Likewise *rdist is* generally adequate for putting several objects on several machines. However, *rdist* has some of the same maintenance requirements as *make.* In particular, it requires that a separate control file be created for every distribution directive. In order to circumvent the problems of maintaining large numbers of control files, we specify a central point of distribution control.

The result is *rtools:* a system built around *make* and *rdist,* which separates compilation and distribution. With *rtools* makefiles need only create executables; other means are used to distribute and install them. *Rtools* creates and installs binaries based on information in a central database, hiding the details of remote compilation and installation for other architectures from the user and from the makefiles.

Design Overview of Rtools

Splitting the functional lines of the file creation process and the file distribution process is a conceptual step above the previous method of handling both creation and distribution in the makefiles of each source code subsystem. The major elements associated with this system are: a database which maintains necessary software management information, *rcreate* which handles the creation/compilation of object/executable files, and *rput* which deals with the distribution of those same files (see figure 1).

In essence, *rcreate* and *rput* are simple programs that drive the individual system elements. Each calls tools that extract information from the database and pass that information to the intermediate tools which in tum process the database information into directives for standard Unix facilities. From there, the creation or distribution process is completed using well known Unix tools, drawing on the information provided by the *rtools* system.

Throughout this project we utilize existing facilities where possible, taking advantage of the Unix filter paradigm and the tools available with the Unix 4.3BSD release, avoiding the replacement of any

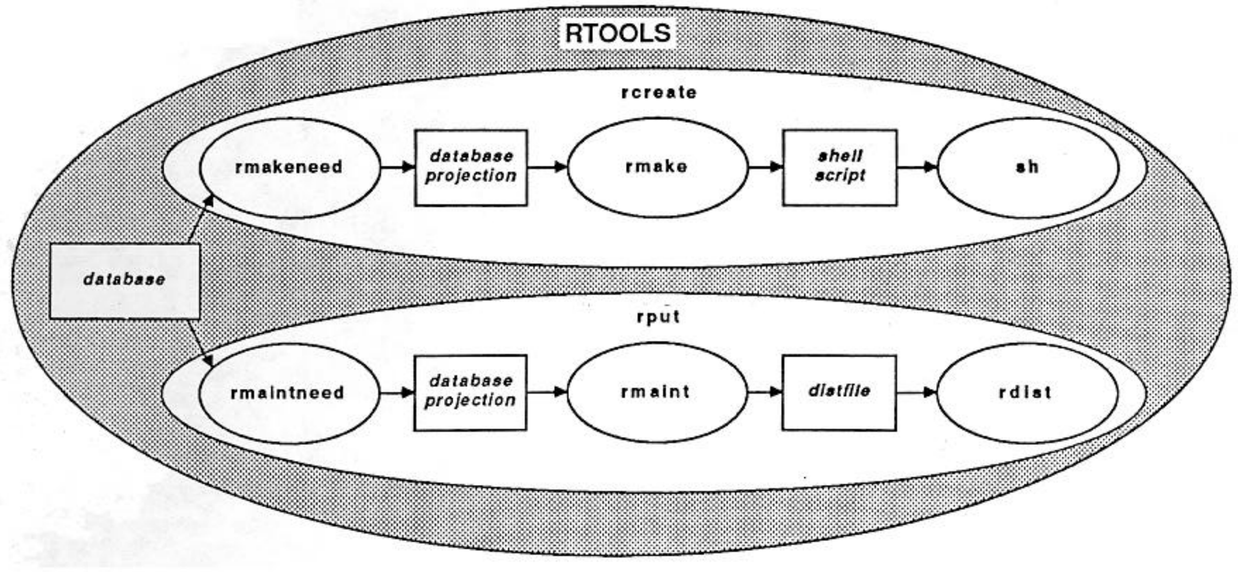

Figure 1 - A representation of the *Rtools* design

functionality that could be integrated from system utilities such as *make* or *rdist.* The tool-set uses standard input and standard output to interface the individual utilities and depends heavily on the Boume Shell for internal cohesiveness.

2. Shadow Trees

For storage of the distributable files, we create "object trees" which mimic the organization of the source code directory /usr/src (see figure 2). There is one object tree for each different architecture or operating system. We called the directories where object files are stored, "Shadow Trees", because their structure is a projection of the source code tree.

Each object tree, or "shadow tree", is a subdirectory of /usr/obj. The correspondence of the trees detemmines exactly where the executable for a particular architecture may be found, given the location of its source code. For instance, the source code for a program called *batch*, might reside in the file /usr/src/local/std/batch/batch.c. The resulting object file, compiled for a Convex C-1 computer would be the file: /usr/obj/convex/local/std/batch/batch.o.

3. Control Database

All the information for remote compilation and distribution formerly kept in makefiles is now centrally located in a single control database. A reason for using a single central database is that some information is common to both the compilation and the distribution process. Furthermore, by removing that information from the makefiles *rtools* is able to abstract the distributed nature of the environment away from the compilation tools.*

In hindsight, this is a simple concept. It is also easy to implement but will be difficult to maintain as the database grows. Until proper database support tools are devised, some aspects of maintaining the information will remain awkward.

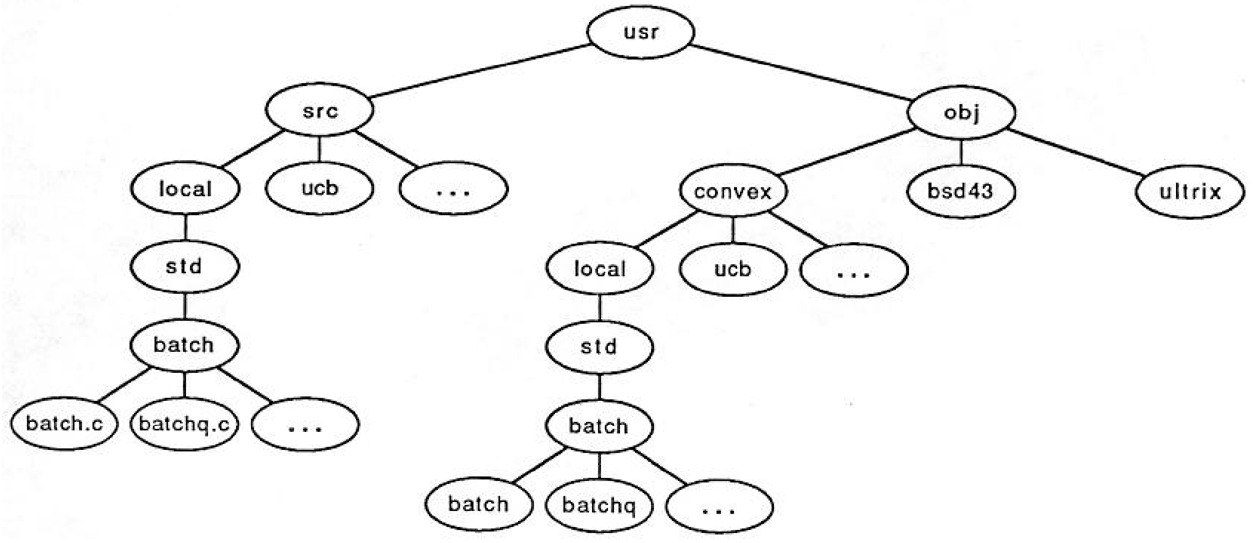

Figure 2 - *Rtools* Directory Structure

* The Tektronics "Utek Build Environment" makes similar use of a central database for its source management, but with a different goal. Their system provides a soft~vare development environment, while *rtools is* designed for software maintenance.[2]

Records

There is one record for each distributed file (see table 1). Each record in the control database is divided into ten fields. The data in the fields control the selection and utilization of individual records. Each record can be selectively accessed through one of two data extraction tools. They mask away information unnecessary for some particular step in the distribution or installation process.

Fields

Each record contains the following fields:

- Host List
  This field contains a *host list,* describing the systems for which the current record applies.
- Installation Class
  Currently, the installation class is one of "bin", "lib" or "man". Other classes may be appropriate as the database develops.
- Target Architecture
  The architecture or operating system for which this file is to be prepared.
- Shadow Directories
  Distributed objects/executables reside in a directory subtree appropriate to their target architecture. By its nature, this field denotes the path for both the source directory and the object directory for the file described by the current record.
- *Make* Target
  This field contains the target that directs *make to* create the file locally, in preparation for distribution.
- Local File Name
  Usually identical to the *make* target
- Remote Destination Name
  This field is the fully qualified (e.g. /usr/local/gnu/bin) path name of the destination directory and filename on the remote host.
- Owner, Group, and Mode Fields
  These three fields specify the eventual owner, group designation and file permission mode on the remote machines.

Field	Used In	Example
Host List	file creation & distribution	natasha boris
Target Architecture	file creation & distribution	convex
Installation Class	file creation & distribution	bin
Shadow Directory	file creation & distribution	local/std/batch
Make Target	file creation only	batch
Local File Name	file distribution only	batch
Remote Distribution Name	file distribution only	/usr/local/std/bin/batch
Owner	file distribution only	bin
Group	file distribution only	bin
Mode	file distribution only	755

Table 1 - A database record

4. Rcreate: The Binary Creation Tool

Rcreate, the *rtools* compilation utility, must accomplish three tasks. First, it must acquire information about the current location of source files and the final destination of the executables. Through *rmakeneed* a data access program, the database provides all the necessary information. Second, it must compile software for remote machines, accommodating different operating systems and different architectures. The final requirement is that it place all object and executable files within the appropriate shadow tree.

In *rtools*, we made no additional enhancements to *make.* Instead we augment the make process with a superstructure of shell directives. *Rcreate* makes use of a our local distributed computing utilities. FREEDOMNET,[3,4] developed at the Research Triangle Institute, is a distributed computing system that transparently provides both remote execution and remote file access across heterogeneous architectures. We use FREEDOMNET not only for maintaining sources in one virtual place, but also for remote compilation on different architectures. Specifically for its second task, *rcreate* uses the FREEDOMNET utility *excr* which makes the process's notion of the root directory (/) relative to the remote machine. *Excr* preserves all of the current environment, including the current working directory, while all of the programs invoked, in particular compilers, execute on the foreign machine.

To accomplish the third task, *rcreate* employs *build*[*], - a *make* utility also developed at RTI.[5] *Build* provides the ability to search multiple source directories specified in the environment variable VPATH. It also maintains a correct VPATH through recursive invocations in subdirectories. It also has the ability to augment the VPATH variable dynamically as scripts within the makefile change directories and again invoke *make/build. Rcreate* uses this mechanism to maintain the separation between the source and object.

Rcreate itself is a pipeline of other tools: *rmakeneed* - the database data extraction utility, *rmake* - the compilation coordinator, and the Boume shell to invoke *build.*.

Rmakeneed

Rmakeneed is a database projection tool. It bridges the information in the database with a script generator and serves as the source for the compilation pipeline. Its function is to extract the information required to create files for machines throughout the distributed environment. The output has the form of directives for the *rmake* utility.

Rmake

It is in the *rmake* step that the distributed nature of the environment is introduced into the make process. *Rmake* takes the database projection from *rmakeneed* and produces a shell script with the proper environment variables, etc. to *excr* to a machine of the appropriate architecture and run *build* for each of the specified *make* targets. It should be stressed that the "make-in"" utility is not affected by this step.

Build

Build is called when the shell script produced by *rmake is* run. The previous steps of creating an *ad hoc* compilation environment allows *build* to proceed in a normal fashion, oblivious to the nature of the distributed environment in which it is working.

[*] Build is a "view path searching version of *make*developed at AT&T Bell Laboratories[6] and reimplemented ar the Research Triangle Institute. It uses the environment variable 'VPATH' which specifies the directory trees to search when looking for makefiles and source files.

5. Rput: The Distribution Tool

Rput is the *rtools* file distribution utility. It runs *rdist* using information from the *rtools* database. *Rput* obtains the information required to produce the distfile from *rmaintneed. It* pipes the result to *rmaint*, which creates the distfile, which is in turn piped to *rdist. Rdist* takes a specification of what files should be put where on a set of hosts. That specification is often in a file called a "distfile". *Rtools* needs only a well defined subset of the extensive capabilities of *rdist;* by generating distfiles *ad hoc*, we avoid maintenance of many hundreds of distfiles containing information mostly redundant with respect to each other or with respect to the contents of the control database.

Rmaintneed

Rmaintneed is a database access tool whose output includes directives for *rmaint.* Like *rmakeneed, rmaintneed* serves as the head of its process pipeline. However, the information extracted by *rmaintneed is* specific to the distribution process.

Rmaint

The labor saving tool of the distribution process is *rmaint. Rmaint* automatically generates distfiles for the *rdist* program, hiding the details of *rdist* syntax from *rmaintneed It* takes distribution specifications on its stdin and puts the distfile on its stdout.

Rdist

The actual distribution of files is performed by *rdist.* This powerful tool relieves much of the burden of verifying modification dates, connecting to remote machines, and handling the security issues surrounding distributed environments.

6. Rtools in Other Environments

We consider the design of the system to be more useful than any particular program comprising it. The shell scripts are almost trivial, but they obviously rely heavily on some powerful tools that were already implemented. A similar system could be implemented for a different environment using other commonly available tools. Here we discuss some of the details of implementing elements of the *rtools* package in other environments.

Distributed File Systems

Perhaps the most striking feature used by the *rtools* package is the FREEDOMNET *excr* utility. We surmise that *Excr* could be emulated in other distributed file systems by mounting directories of the source machine onto a target machine, and then invoking a compiler on the target machine, perhaps with *rsh* (Berkeley remote shell).

In the absence of a remote file system, some elements of the *rtools* design could be implemented using other Unix concepts (e.g. remote daemons for compilation and *tar* or *cpio* for file transfer). One might also examine the traditional alternative of using cross compilers.

Using Make/Build

Build is more subtly important: it allows a minimum of intervention into a working makefile to adapt it to a variety of situations. The *make* delivered with 4.3BSD also recognizes the VPATH variable but without the ability to change when invoked in another directory. Such recursive calls of *make* happen surprisingly often—it appears in over fifty of the software subsystems delivered with source to 4.3BSD. *Nmake*, a fourth generation make utility in the AT&T Toolchest[7], may also cope with this situation.

Using Rdist

Rput depends heavily on *rdist*. In its absence, one might implement a work-alike using *rsh* and *tar*, but there are important considerations that *rdist* addresses properly, such as having the loop over machines outside the loop over files, and the network security issues involved in a utility that will install sensitive programs (and what program isn't?).

We should mention here a local change to *rdist*: when the intended user on the remote host has not given permission for the user at the local host to gain access automatically with an appropriate .rhosts file, our version of *rdist* will prompt for the appropriate password. The behavior of the Berkeley version of *rdist* was to simply deny permission.

We also added options to allow more flexibility in maintaining remote files, namely set owner, group, and mode, do comparisons, and install even if the file has not been modified.

Information Management

We chose a database format that required the least possible work in creating database tools. After more experience with the system, we may decide that more formal and better controlled database tools would be worth the investment, at which point we could turn to a traditional DBMS to manage the data and to produce the input expected by *rmake* and *rmaint.*

7. Conclusions

Rtools is in production use and is currently managing software in an environment that includes 4.3BSD, Ultrix, and Convex 4.1 Unix implementations. We plan to add *rtools* support for Sun Unix 3.5 in the immediate future, and possibly Apple Unix later.

We have a powerful tool with simple syntax after a modest programming effort. It succeeds even with complicated software systems with many machine dependencies and large numbers of components, such as GNU *emacs.* It has automated an error-prone and tedious task, allowing us to support a consistent environment in a more timely and reliable manner. It frees support staff to concentrate their attention elsewhere.

Background

The Microelectronics Center of North Carolina is a non-profit corporation involved in cooperative research and education that unites the resources of five participating universities, the Research Triangle Institute and a $40 million research facility. MCNC research programs focus on integrated circuit design, simulation and testing, as well as semiconductor materials, devices, and fabrication processes. MCNC works with its industrial affiliates to accelerate the transfer of this basic research into commercial application.

The MCNC computer facilities include 2 VAX 8650's, one VAX 11/750 and an assortment of Microvax II's running 4.3 BSD Unix™, 2 Convex C-1's (4.2BSD variant), a MegaOne automated wafer tester (4.2BSD variant), several Vaxstation 2000 and Microvax GPX's running Ultrix 2.0, one Sun-3/260, and other miscellaneous workstations. The machines are networked together on 2 local ethernets: one for the main computing center and one for outlying workstations. We expect the number of workstations to increase greatly in the coming years.

In addition, MCNC created and currently maintains a state-wide microwave network connecting the participating institutions, providing broadband circuits for data communication, and two-way color video channels for interactive teleclasses and conferencing. MCNC maintains gateways for the ARPANET and SURANET, providing internet access for the southeast regional area We support nearly 400 registered internet hosts, including departments at the Research Triangle Institute, NC State University, UNC-Chapel Hill, Duke University, UNC-Charlotte, NC A&T University, and the Triangle Universities Computation Center. (We plan to add departmental access to Wake Forest University, Winston Salem State University, UNC-Asheville, and Eastern Carolina University.) MCNC has a full time system support staff of 6 full-time programmers, 2 operators, and an electronics technician.

References

1. Helen E. Harrison, "Maintaining a Consistent Software Environment," *Proceedings of the Large Installation System Administrators Workshop,* (April, 1987).

2. Alan McIvor, "UTek Build Environment," *Proceedings of the Summer USENIX Conference, pp.* 437 443 (1987).

3. Bob Warren, Tom Truscott, Kent Moat, and Mike Mitchell, "Distributed Computing using RTI's FREEDOMNET in a Heterogeneous UNIX Environment," *Proceedings of the UNIFORUM Conference, pp.* 115-126 (1987 j.

4. Tom Truscott, Bob Warren, and Kent Moat, "A State-Wide UNIX Distributed Computing System," *Proceedings of the Summer USENIX Conference, pp.* 499-513 (1986).

5. Kent Moat, "Design of build: A Path Searching *make,"* Technical Memorandum, Research Triangle Institute,, Research Triangle Park, NC 27709 (1986).

6. Verlyn Erickson and John Pellegrin, "Build - A Software Construction Tool," *AT&T Bell Laboratories Technical Journal,* July-August, 1984.).

7. Glenn S. Fowler, "The Fourth Generation Make," *Proceedings of the Summer USENIX Conference, pp.* 159-174 (1985).