1985

TILDE Trees in the UNIX Environment

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Ralph E. Droms

Report Number:
85-503

TILDE TREES IN THE UNIX ENVIRONMENT

Douglas Comer
Ralph E. Droms

CSD-TR-503
January 1985

Tilde Trees in the UNIX Environment

Tilde Report CSD-TR-503

January 28, 1985

Douglas Comer
Ralph E. Droms

ABSTRACT

TILDE is a multi-year research project exploring distributed computing in an environment where the primary computing engine consists of a cluster of UNIX-like† systems loosely coupled with high-speed local area networks. The goal of TILDE is to extend UNIX to provide a transparent integration of local and remote facilities, allowing the user access to remote files and services without forcing the user to know about the location of data, processes, or other objects such as servers. One important aspect of TILDE is the Tilde naming scheme. The TILDE approach to naming objects in a distributed system provides a transparent, consistent mechanism for referencing both local and remote objects.

The Tilde naming scheme substitutes a collection of directory hierarchies, known as *Tilde trees*, for the single UNIX directory hierarchy. A program executes in an environment composed of a *forest* of Tilde trees, which can be selected from the collection of Tilde trees known to the distributed computing system. In the prototype, each of the Tilde trees is mapped onto a UNIX subdirectory. As in UNIX, files are named either relative to the current working directory, or by a full pathname. Full path names begin with tilde, and the first component identifies the appropriate Tilde tree, while the remainder of the

† UNIX is a trademark of Bell Laboratories.

pathname completely specifies the file within the Tilde tree. (The naming scheme can be viewed as an extension to C-shell naming mechanism.) A particular process executing within the TILDE system need not know the details of the storage organization or file location in the distributed system; rather, there is a uniform mechanism for searching and accessing Tilde trees.

This paper describes the naming scheme and a prototype implementation. It discusses alternative designs and compromises imposed by the limitations of a single-processor UNIX environment, as well as the impact of using the Tilde environment for tasks such as software development. We show how our Tilde naming software makes it possible to move large programs from one directory to another or from one UNIX system to another without recompiling (even if path names have been bound into the code). Finally, future directions for this work, including extension to collections of UNIX systems and modifications to include the Tilde tree naming system in the UNIX kernel, are also discussed.

1. Introduction

TILDE is a multi-year project exploring distributed computing in an environment where the primary computing engine consists of a cluster of UNIX-like systems loosely coupled with high-speed local area networks. The project addresses many research issues, including naming objects in a distributed environment. The goal of TILDE is to extend UNIX to provide a transparent integration of local and remote facilities, allowing the user access to remote files and services without forcing the user to be aware of the location of data, processes or other objects such a servers. Naming is the key issue involved in making access transparent. In order to provide flexible, convenient and portable names, the notion of a single, rooted directory tree must be abandoned. For example, solutions that impose a virtual root above all UNIX machines (e.g., [1, 2]) are nontransparent because they imply that users understand locations of files and must change software when data or programs referenced by that software moves. We propose instead a new scheme that permits large, complex software systems to be constructed in such a way that the directory in which the system resides can be moved without requiring programs to be recompiled.

The conceptual change from UNIX to our new naming scheme is simple. UNIX employs a single directory tree per machine, with authority for names in the upper part of the tree vested with the system programmer. Our new scheme employs a set of

independent trees, with authority for the structure of each tree vested in its owner. We claim that:

> *By replacing the UNIX tree-per-machine naming system with multiple independent trees, we will integrate local and remote names into a single mechanism, improve software portability, and retain the advantages of a hierarchical directory system.*

Our new system substitutes a collection of directory hierarchies, known as *Tilde trees*, for the single UNIX directory hierarchy. A program executes in an environment composed of a *forest* of Tilde trees. Each user maintains a *private forest*, which can be thought of as an active subset of all Tilde trees available throughout the distributed computing system. Each Tilde tree has a name, selected by its owner when the tree is created. Thus, a private forest is represented by a set of tree names. The user's private forest establishes the Tilde environment for any processes executed by the user.

Running programs, including command interpreters (which are called *shells* in UNIX), reference files using names of the form:

~tree/path

where *tree* corresponds to the Tilde tree in which the file is located, and *path* gives a complete path from the root of the tree to the file. For example, the name "~dec/bin/xinu" references a file named *xinu* in directory *bin* located just under the root in Tilde tree *dec*. A particular process executing within the TILDE system need not know the details of the storage organization or file location in the distributed system; rather, there is a uniform mechanism for searching and accessing Tilde trees. Tilde trees may, in fact, be distributed across a network transparently to the process. The process knows to name a particular file by its Tilde name, regardless of whether the Tilde tree containing that file exists on a local or a remote system.

2. The Tilde Tree Abstraction

A distributed computing environment such as TILDE provides access to objects on many computing machines. The naming policies and name interpretation mechanisms in a distributed environment limit the extent to which programs, processes, connections, and data can be exchanged meaningfully. Moreover, poor names may impact program portability or may restrict the usefulness of a system merely by making it inconvenient to exchange objects or share computing services.

In distributed environments, the question of naming is central to system design. A good naming scheme permits efficient exchange of objects by permitting processes to exchange object names. However, names can only be exchanged freely among parts of the system that dereference them the same way. We capture this measure of exchangability by saying that naming is *transparent* if all names are known globally (i.e., if a given name refers exactly to the same object independent of the context in which it is interpreted). Name transparency is desirable, at least for names that users learn and manipulate, because it decouples object management from object naming.

Transparency can be achieved by making all names unique. Uniqueness requires either a central authority to designate all names, or an agreement by which individual "sites" (e.g., a computing mechanism) can assign names. The establishment of a central naming authority was appropriate when computing systems were independent. However, a central naming authority becomes inefficient and a bottleneck in a distributed system. A central naming authority also reduces the flexibility and imposes a rigidity of structure on distributed systems. All modifications to the system structure and naming conventions must be registered withe the central authority. Finally, naming provided by a central authority will not be convenient for users and will not meet the users' needs for short, mnemonic names.

Distributing the authority for name assignment introduces conflict possibilities because multiple sites may choose the same name for different objects. One way to solve the conflicts is to use location dependencies. For example, each site can be assigned a unique site identifier, denoted *siteid*, and assign names as a pair (*siteid,otherid*). In practice, the resulting name is only valid as long as the referenced object remains at its original site; moving the object requires inventing a new name.

Site identifiers have been used to extend UNIX file names to a multiple-machine environment in [1, 2]. Both systems overload the syntax of UNIX path names, relaxing the usual interpretation. IBIS prepends machine names onto file names, making names take the form *machine:file*. Names in which the first component does not end with a colon are assumed to be local file names. The Newcastle connection [1] also provides an extension to UNIX by allowing names of the form "/../siteid/path". Names that do not begin with "/.." are interpreted with respect to the local system, so all existing software continues to operate correctly. Because the reference "/.." does not occur in standard use, there is little possibility for inadvertent reference to a remote file. Note, however, that both examples employ non-uniform names for local and remote references.

In addition to syntactic inconsistencies, location dependent schemes tie the naming of objects to the physical organization of the distributed system, much the same as early file systems tied the name of a file to the physical storage device on which it resided. A connection between the hardware organization and the naming scheme is inflexible because it means that references to named objects must be changed whenever an object moves from one machine to another.

The Tilde naming scheme solves the problem of naming in a distributed environment in a novel way: it replaces a per-machine name binding mechanism with a per-user name binding mechanism. Thus, names are relative to the individual, not to the machine on

which the computation is performed. In a TILDE system, users can refer to a file without knowing the file's physical location or the location of the computational service they invoke. Furthermore, a default set of Tilde name bindings, established when the user logs on the system, allows users to refer to system directories and to each other's files almost exactly as they would on a single UNIX system. More importantly, the Tilde naming scheme makes local and distant file names uniform and transparent, allowing consistent access to files on both the local and distant computing mechanisms.

3. The Prototype System

We have implemented a prototype of Tilde naming system that executes on a conventional UNIX operating system. The prototype consists of a layer of software that runs outside the kernel, but captures and interprets system calls (e.g., **chdir**, **creat** and **open**). The goal of the prototype was to create an execution environment capable of supporting the typical edit-compile-test program development cycle. As an experiment, we converted a subset of the UNIX commands to run under our Tilde environment.

The prototype, which currently executes under 4.2bsd UNIX, can coexist with UNIX. It maps Tilde trees onto UNIX directories and even allows the user to specify that UNIX path names are allowed. Our prototype includes the following components:

- A *modified C library* that contains, in addition to conventional C library routines, a set of procedures that correspond to UNIX system calls.
- A *modified C shell* that supports the user interface to the Tilde environment.
- *Modified utilities* that have been converted to use the Tilde naming system.
- A *forest of Tilde trees* that provide a UNIX-like for Tilde system program execution

The components of the prototype combine to form an experimental environment in which users can:

- *Initiate a session* as though logging into a TILDE system
- *Create software* that uses the Tilde naming scheme and takes advantage of the resulting portability and access transparency
- *Change* their private Tilde forest

The next sections describe the prototype in more detail.

Modified C Library. The new C library includes procedures that intercept system calls, map references to Tilde names onto UNIX names according to the user's Tilde mapping, and call appropriate operating system routines to access files. The new library is constructed so that programs can be prepared to execute in the Tilde environment merely by linking them with the new library.

The prototype passes the user's Tilde tree name bindings to subprocesses through the UNIX environment. *Tilde tables* are files of mappings from Tilde names onto UNIX names. These files contain pairs of bindings. They are represented as ASCII text according to the simple syntax:

~tildename | UNIX-path-name

where the symbol "|" is used to separate the Tilde tree name from the UNIX directory to which it is bound. For example, the standard system Tilde table TILDE_SYS includes the entries:

```
~bin | /usr/tilde/bin
~tmp | /usr/tilde/tmp
~etc | /etc
```

Because they contain ASCII text representations, Tilde tables can be manipulated just as any other text files using utilities such as **vi**, **sed** and **grep**. The environment variable **TILDE_PATH** defines a list of Tilde tables which are to be used in looking up a Tilde name. When a Tilde name is encountered, the Tilde environment software performs a linear search of the Tilde tables in the order specified by TILDE_PATH to find the UNIX

name bound to the Tilde name. For the sake of efficiency, an in-core Tilde directory is constructed from the environment variables at the time of the first Tilde reference. Since the in-core directory is constructed at the time of the first reference to a Tilde name, a process can give itself a trusted Tilde environment by making a call to a new procedure **resetpath()** that resets the Tilde directory and TILDE_PATH to a set of trusted system Tilde tables.

Modified C shell. We modified the C shell to:

- process names that begin with "~" according to the Tilde naming system,
- initiate a Tilde system session without logging out of UNIX,
- include builtin commands to manipulate the Tilde environment.

The Tilde csh presents users with an environment closely analogous to the initial UNIX system environment when they begin. For example, ~**userid** refers to the home directory of the user with login id **userid**, just as with the conventional csh.

The Tilde csh allows simulated log in making it possible to switch to a Tilde environment and experiment without permanently changing login shells. The Tilde csh initializes its TILDE_PATH to include the system Tilde tables TILDE_SYS and TILDE_USR. As mentioned above, TILDE_SYS includes standard system mappings. TILDE_USR includes mappings of userid to home directory for all system users, providing a mechanism similar to the UNIX csh tilde shorthand naming. To simulate login, the Tilde csh changes its working directory to the users home directory, and maintains an internal copy of the name of its current working directory. This current Tilde directory is passed to any subprocesses through the TILDE_CWD environment variable. Once logged in, a user invokes commands as usual, except that all full paths must begin with "~tree". Tilde csh expands arguments into full Tilde names as well, so "echo ~dec/bin/*" produces names like

"~dec/bin/myspell" and "~dec/bin/manual". The Tilde csh allows modification of its environment variables to allow the user to dynamically alter the tilde mappings. To keep the in-core Tilde directory up to date, it is reinitialized after every modification of an environment variable with the prefix TILDE_.

Modified commands. We converted a subset of the utilities provided by the UNIX system to use the Tilde naming system. Many of the conversions involved only changing all UNIX system names to Tilde names, and recompiling using the Tilde library.

Default Forest. The Tilde forest is composed of directories dedicated to the Tilde prototype and directories shared with the standard UNIX system. For example, ~bin maps to a new directory that holds the modified commands. Similarly, ~lib and ~tmp map to new directories which we created just for the Tilde prototype. Some directories must be shared, however, with the existing UNIX system. For example, ~dev, ~usradm and ~etc are bound to UNIX directories /dev, /usr/adm and /etc, respectively. Thus, Tilde prototype utilities such as **df** and **who** continue to function normally concurrently with the UNIX system.

Current prototype status. The Tilde execution environment has been used to bootstrap a copy of all the prototype software. Temporarily, full UNIX path names are allowed to ease the migration to the Tilde prototype (they can be disallowed by the user). The prototype reports warnings of UNIX path name references. The user can suppress warnings setting environment variable TILDE_WARN to "NO". Setting environment variable TILDE_FULL_PATH to "NO" disallows any full path references. Tilde trees, as implemented in the prototype, do not fully conform to the abstract definition of Tilde trees. The major difference is that the prototype's Tilde trees are not entirely disjoint, due primarily to the mapping onto the UNIX file hierarchy. The prototype maintains its notion of current working directory based on the Tilde name specified by **cd**, and does prohibit exiting the root of a Tilde tree by changing to directory "..". No effort is made, however, to

determine if a Tilde tree is a subset of another Tilde tree, or if subtrees of different Tilde trees are connected by links.

Another restriction on users of the Tilde prototype is the small subset of the UNIX utilities which has been converted to date. For example, we have not yet tackled the conversion of EMACS to the Tilde environment. We expect to convert more utilities after expanding the Tilde prototype to include network file access.

4. Experiences with the Tilde System

Most of the work done under the Tilde prototype has been software development. We are currently using the Tilde prototype to work on the next revision of the system itself. During development of the Tilde prototype's user environment, we discovered many instances of the close connection between the UNIX file system naming scheme and the UNIX utilities. We attempted to construct the Tilde forest to mimic the UNIX file hierarchy as closely as possible by defining Tilde trees such as ~bin, ~lib, ~tmp, etc., that correspond to existing directories in the UNIX file hierarchy. We expected that this correspondence would allow easy substitution of tilde names for UNIX names in the utility programs (e.g., globally substitute ~bin for /bin), and would ease the conversion of existing software. Unfortunately, UNIX software derives many names in non-obvious (and generally undocumented) ways. For example, the loader ld searches libraries in directories /usr/lib and /lib. This search is performed by first constructing a string "/usr/lib/libname.a" and attempting to open the file. If the open fails, the pointer is simply incremented by 4 (to point at "/lib/libname.a"), and the open is attempted again. Merely replacing "/usr/lib" with "~usrlib" does not produce the intended result. Csh makes use of the shared string generation package xstr. To prevent certain strings (which are dynamically altered during csh execution) from being shared, they are declared as arrays of characters:

```
char pstr[ ] = { '/', 'b', 'i', 'n', ... 0 };
```

(xstr does not recognize such a declaration as a character string). Of course, a global replacement of "/bin" by "~bin" also fails to replace these declarations correctly. Much effort was spent in tracking down these declarations and converting them to tilde names.

5. Future directions

One project of immediate interest is the incorporation of the Tilde naming scheme into the make utility. We expect to take advantage of the Tilde naming scheme concept to allow the compilation of one set of source code (maintained in a single Tilde tree) into executable code in several Tilde trees (e.g., one Tilde tree of code for each machine in a network). We also realize that there is work to be done to make the interface to the user's Tilde environment more convenient. Utilities for easy modification of the Tilde environment are under development. The long range goal of this project is to incorporate the Tilde naming system into the UNIX kernels of the machines in the TILDE computing engine. The next step is to move the translation step of the Tilde naming scheme into the kernel. We expect, as a transition step, to build a system which supports both a standard UNIX directory and a collection of Tilde trees, perhaps as entries in the root directory. Finally, we will incorporate a remote file access system to provide user access to remote Tilde trees in a transparent fashion.

6. References

[1] Brownridge, D. R., L. F. Marshall and B. Randell, "The Newcastle Connection or UNIXes of the World Unite!", *Software Practice and Experience*, Vol. 12 (1982), 1147-1162.

[2] Tichy, Walter F. and Zuwang Ruan, "Towards a Distributed File System," *Proceedings of the Summer USENIX Conference* (June, 1984), 87-97.