1987

An Experimental Implementation of the Tilde Naming System

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

Ralph E. Droms

Thomas P. Murtagh

Report Number:

86-642

An Experimental Implementation of the Tilde Naming System

Douglas Comer
Ralph E. Droms
Thomas P. Murtagh

Computer Science Department
Purdue University
West Lafayette, IN 47907

CSD-TR-642
February 1, 1987

An Experimental Implementation of the Tilde Naming System

Abstract

The Tilde naming system identifies files in a distributed computing system in a novel way, providing a consistent, transparent mechanism for referencing both local and remote files. Several contemporary distributed systems try to provide transparent naming by hiding complex, heterogeneous computing systems beneath a uniform, global name evaluation environment. In the Tilde naming system, on the other hand, transparency is achieved by providing each user with control of an independent, local name evaluation environment. The local naming environment supported by Tilde naming integrate local and remote names into a single naming mechanism while improving software portability and retaining the familiar advantages of hierarchical file naming systems.

We have constructed an experimental implementation of the Tilde naming system, and created a computing environment sufficiently rich to support significant software development. With this prototype, we have been able to study the effects of the Tilde naming scheme on user environment maintenance, software project development, information sharing and other issues. This paper summarizes Tilde naming and discusses insights into naming mechanisms gained through the use of the experimental system.

1 Introduction

The success with which a computing system provides an abstract view of the user's environment can be measured by the system's *transparency*. Transparency is important to the users of a computing system, in that increased transparency implies increased flexibility and decreased dependency on the specific architecture of the underlying system. To the extent that a system is not transparent, users must be conscious of changes in the computing environment, and software systems must be altered in response to modifications in the computing system itself.

The TILDE Project [Com84] is an investigation into distributed computing systems in which the user interface is used to hide the heterogeneous nature of the underlying hardware. A *TILDE Computing Engine* is a cluster of UNIX-like systems loosely coupled with high speed interconnection networks. These computing sites each provide a process-oriented style of computation, in which processes communicate with other local and remote processes through the interconnection networks.

The Tilde naming system is based on a collection of small, disjoint namespaces, which replace the single, global namespace of UNIX[1] and UNIX-like systems[2]. There are two primary motivations for this fundamental difference between Tilde naming and more familiar naming mechanisms. First, the local naming environment and name evaluation mechanism separate the *identification* of files from the *location* of files, increasing the transparency of the naming system. Second, the organization of files into disjoint namespaces, which can then be identified by local names, provides a valuable style of abstraction to software subsystems. This new naming mechanism provides transparent identification of local and remote files, while enhancing software system modularity and portability. The primary penalty associated with Tilde naming is the added burden to the user of managing a local naming environment.

To evaluate methods for minimizing the impact of name management, and to explore the effect of Tilde naming on file naming transparency and software portability, we have constructed an experimental implementation of the Tilde naming system. This paper concentrates on our experiences with the experimental system.

We have organized the remainder of this paper into three parts. The next section summarizes the Tilde naming system, which has been described in more detail elsewhere [CoD85,CoM85,Dro86]. Section 3 outlines the Tilde naming prototype system. The third

[1]UNIX is a trademark of AT&T.

[2]We assume the reader is familiar with *hierarchical naming systems* such as provided by UNIX and its derivatives. Thompson [Tho78] and Quarterman, et al. [QSP85], have written excellent descriptions of the details of the UNIX file naming mechanism.

part of the paper, Sections 4, 5 and 6, illustrates the use of the Tilde naming mechanisms, discusses insights into Tilde naming gained through the use of the experimental system, and presents future directions for this research.

2 The Tilde Naming System

In the Tilde naming system, a process identifies files within a local environment called its *Tilde Forest*, which is composed of disjoint naming hierarchies known as *Tilde Trees*. A Tilde Forest represents the entire file naming environment for a process. That is, a process can only access files in the Tilde Trees within its Tilde Forest. File names are of the form *~tree/path* where *~tree* selects a Tilde Tree, and *path* gives a complete path from the root of the Tilde Tree to the file itself[3]. The first, or *Tilde Name* component of a file name is interpreted relative to the collection of Trees in the Tilde Forest. The process chooses its own Tilde Names as identifiers for the roots of the Tilde Trees in its Tilde Forest. Figure 1 shows an example of a process and its Tilde Forest.

Each Tilde Tree also has a location independent name which is unique throughout the Computing Engine. The name resolution mechanism identifies a Tilde Tree by this unique name, which we call a *Medusa Name*[4]. A Tilde Forest is represented by a list of bindings from Tilde Names to Medusa Names, so that a file's Tilde Name is first resolved to a Medusa Name, which the resolution mechanism then uses to locate the Tree itself.

In the figure, the Tilde Forest extends across the Tilde Computing Engine to Tilde Trees residing on several remote nodes. The naming environment provided by the Tilde Forest is, therefore, not limited to a single node, but can span the entire Computing Engine.

In Figure 2, the process has three Trees in its Tilde Forest. The figure shows the internal structure of a Tree identified by the Tilde Name `~system`, and illustrates the resolution of the file name `~system/cmd/ls`. The resolution begins by parsing the file name into two components, `~system` and `/cmd/ls`. The Tilde Tree named by the component `~system` is identified by the entry in the Tilde Forest that maps `~system` to the root of the Tree. The

[3]The Tilde naming system also supports the resolution of file names relative to a current working directory, which are resolved in the same way as UNIX relative file names.

[4]So named because Medusa Names are ugly and always hidden from the user.

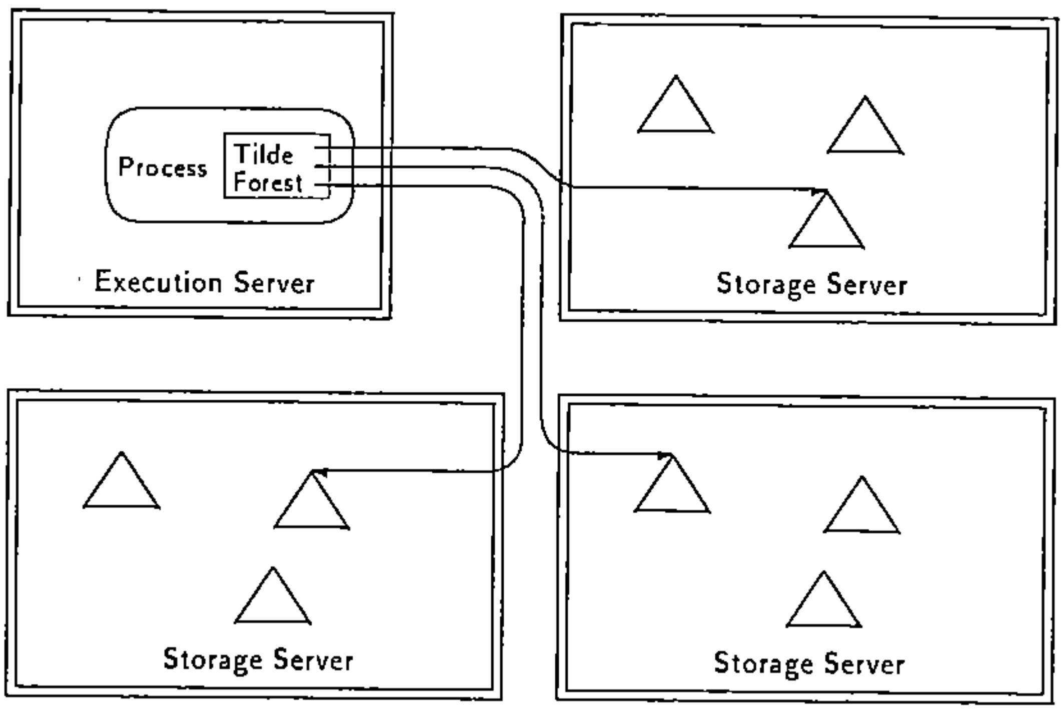

Figure 1: A Process and its Tilde Forest

remaining component of the file name, /cmd/ls, then identifies the desired file within the Tilde Tree through the file name resolution mechanism.

3 A Prototype Implementation of the Tilde Naming System

We have constructed a prototype implementation of the Tilde naming system, based on the 4.2 Berkeley Software Distribution (4.2BSD) of the UNIX operating system [BSD83] and Sun Microsystems' Network File System (NFS) [SUN85]. The prototype runs on a network of

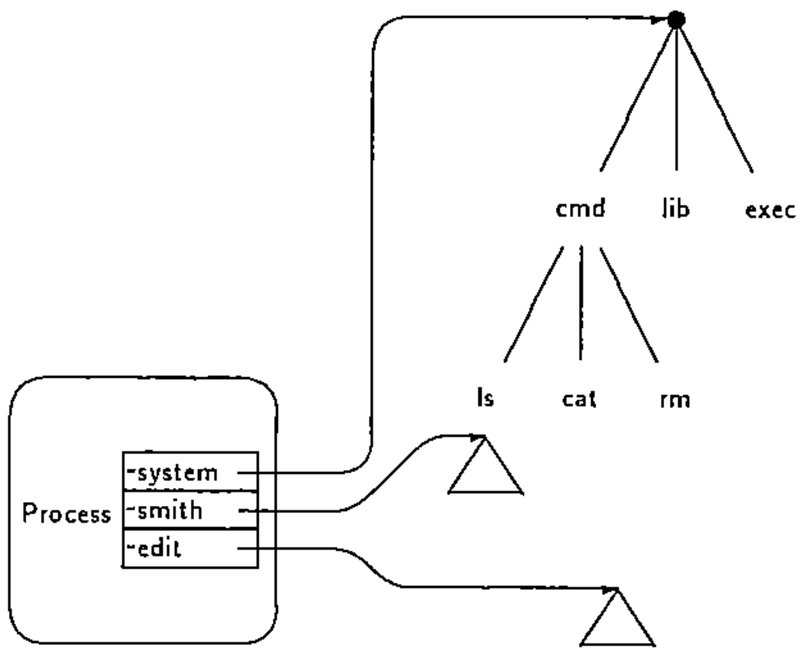

Figure 2: Tilde Name Resolution Mechanism

VAX[5]and SUN-2 computers[6]. The goal of the prototype was to gain insight and experience with distributed naming within a computing environment which is capable of supporting the typical edit-compile-test program development cycle. In addition to implementing support of Tilde naming in the modified UNIX kernel, we converted a subset of UNIX commands to work under the experimental Tilde environment.

The prototype includes several main components:

Modified UNIX kernel: The Tilde system functions, including simulation of Tilde Trees, interpretation of Tilde file names and management of Tilde Forests are implemented through changes to the UNIX kernel.

[5]VAX is a trademark of Digital Equipment Corporation.

[6]SUN-2 is a trademark of SUN Microsystems Inc.

Remote file access: Sun Microsystems' NFS is incorporated into the Tilde prototype through modifications to the UNIX kernel.

Forest of Tilde Trees: Users of the prototype operate within an environment of Tilde Trees that include UNIX utilities such as *ls* and *rm*, program development tools such as *cc* and *ld*, editors such as *vi* and *emacs*, and other software subsystems to support significant software development under the Tilde system prototype.

Modified UNIX utilities: We have converted a subset of the UNIX system utilities to use the Tilde naming system.

Modified command interpreter: The user interface to the Tilde naming system is incorporated into the UNIX command interpreter, *csh* [BSD83], along with other modifications to convert *csh* to use the Tilde file naming scheme.

The key problems in the development of the Tilde prototype included the simulation of Tilde Trees, the implementation of the Tilde Forest, the interpretation of Tilde file names, the use of NFS as a remote file access mechanism for Tilde Tree access and the implementation of the user interface to Tilde naming. We will concentrate on the management of the Tilde naming environment in this paper; other details of the prototype implementation are discussed in "Naming of Files in Distributed Systems" [Dro86].

The details of the Tilde Forest management mechanisms are the result of a conscious effort to build an experimental system suitable for extension. As a result, a) the kernel primitives are minimal and b) the *csh* interface is extensible. The Tilde Forest management system consists of four main components:

Kernel - new system services are used by a process to manage its Tilde Forest

Csh - *csh* commands give the user access to the kernel primitives for management of the *csh* process' Tilde Forest.

Csh scripts - the primitive *csh* interface is extended through the use of the *csh* command language to provide more powerful Tilde Forest management tools

Tilde Tree Registry - identifies the Trees within the collection of Trees managed by the Tilde Computing Engine.

The first two components were constructed as part of the initial prototype implementation, and are discussed here, while the remaining components were developed through the use of the prototype and are covered in Section 4.

3.1 The Kernel Interface

The TILDE Computing Engine employs the UNIX model of process creation, in which a new child process is instantiated as an exact copy of the requesting parent process. The child process inherits the execution environment of the parent process, including open files and current working directory. Under Tilde naming, this inheritance model is extended to include the Tilde Forest, so that a new process inherits its initial Tilde Forest from its parent. A child process, therefore, initially resolves file names in the same environment as its parent process, just as the child process shares other parts of its execution environment with its parent process. Once a process begins independent execution, it can dynamically alter its local Tilde Forest without affecting the naming environment of any other processes.

A process' requirements for modification of its Tilde Forest are simple. As mentioned earlier, the Tilde Forest consists of a list of bindings from Tilde Names to Medusa Names. A process can modify its Tilde Forest by adding a new binding to the Forest or by deleting an existing binding from the Forest. A process can also list the bindings in its Tilde Forest to obtain information about its current naming environment. Adding a new binding between a Tilde Name and a Medusa Name effectively adds a new Tilde Tree to the process' naming environment, making files in that new Tilde Tree accessible to the process. Similarly, deleting a binding from the Forest effectively removes the Tree from the process' naming environment. These simple management mechanisms can be combined into more complex operations, as we will see in a later section.

3.2 The Csh Interface

The *csh* interface to Tilde naming is simply an extension of the Forest management mechanisms described in the previous section. A user's initial Tilde Forest, established by the Computing Engine's session initiation mechanism, consists of a small set of Tilde Trees, including a Tree in which the user can store session customization information[7]. Once the initial Forest is established, the user can interactively establish a Tilde Forest with bindings to a working collection of Tilde Trees. The basic *csh* interface consists of a small set of functions similar to the Tilde Forest management functions available to a process, so that the user can add a new tree to his Forest, delete a Tree from his Forest, and can list the current contents of his Forest.

The *csh* interface is implemented as follows: The Tilde Computing Engine uses a process-based command interpreter mechanism, an extension of the UNIX *csh*, which executes commands on behalf of the user as separate, child processes. Command processes, instantiated by the command processor, inherit the command processor's Tilde Forest and, therefore, execute in the same naming environment as the command processor itself. The command processor's Tilde Forest can be thought of as the user's Tilde Forest. Modifications to the command processor's Forest alter the naming environment perceived by the user. Thus, the *csh* interface includes additional command interpretation to recognize user requests for Forest modification and to execute the appropriate kernel commands for modifying *csh* process' Tilde Forest as requested.

4 Experience with Tilde Naming

We now present an informal summary of our experiences with our experimental Tilde naming implementation. First, we describe extensions to the *csh* interface that support two different classes of users: *novice users*, who work in a static environment of globally shared Tilde Trees, and *expert users*, who modify their Tilde Forests dynamically to share Tilde Trees with other groups of users. Section 4.3 explains the Tilde Tree Registry mechanism for identification of Tilde Trees. The next two subsections illustrate the use of Tilde naming to solve naming conflicts between software subsystems and how Tilde naming provides transparent file identification in a distributed computing environment. This section then concludes with a description of a uniform organization of the components of a Tilde Tree that enhances the project subtree abstraction.

[7]This mechanism is analogous to initial profile mechanisms such as the UNIX *.login* or the VM/CMS *PROFILE EXEC* files.

4.1 Novice Users

The goal of minimizing the impact of Tilde naming on novice users led us to design static, unobtrusive mechanisms that can be used and managed with little interaction on the part of the user. Ideally, the user interface should consist of a single command, perhaps under the control of a system administrator, that establishes a standard, consistent Tilde Forest at the initiation of a computing session. The novice user or the system administrator can add that single command to the user's session initiation file, so that the user need never issue any explicit Tilde Forest management commands.

We take advantage of the ability of *csh* to read sequences of commands from *command scripts* to extend the basic Tilde naming interface mechanisms to include simple primitives for management of a standard Tilde Forest. For example, the Computing Engine administrator can maintain command scripts that define lists of Trees, such as standard system Trees or users' home Trees, so that the user can, with a single *csh* command, incorporate standard sets of Trees into his Forest when initiating a computing session. These lists of Tilde Trees also allow the system administrator to install new system Trees by simply changing the lists of standard Trees.

A user who wishes to establish a personalized Forest, different from the standard system Forests, at the beginning of each computing session, can define his own list of Trees in a local file. Reading the file during session initiation then automatically sets up the desired local naming environment. The mechanism for listing the user's Tilde Forest is integrated with the other Forest management primitives, so that a personalized Forest can be created by interactively constructing the desired Tilde Forest and saving a listing of the Forest in a local file.

4.2 Expert Users

More experienced users can also use command scripts to enhance their interface to the Tilde naming system. A user who makes modifications to his Tilde Forest may want to maintain a consistent environment across computing sessions, retaining changes to the environment rather than establishing a standard, default environment. If the user saves a copy of his Forest when he terminates a session, and reestablishes that Forest when he initiates the next session, the user's naming environment can be retained across sessions.

The ability to quickly switch between multiple naming environments is useful to software developers. For example, when developing a software subsystem for distribution to other Computing Engines, the user will want to test the components of the system in a minimal Tilde naming environment composed of only commonly available Tilde Trees, while developing the code in a more extensive environment. This management function can be accomplished by listing both Tilde Forests in disk files, and creating a command script that deletes the existing Forest and reads in a new Forest from a disk file.

4.3 The Tilde Tree Registry

The environment management primitives discussed in the previous sections assume a mechanism for the identification of Tilde Trees that are not in the Tilde Forest. While the kernel and csh primitives for Forest management use Medusa Names for this purpose, Medusa Names are not an appropriate mechanism for users to identify Tilde Trees. Medusa Names are not mnemonic and are not chosen by the user; they exist only for the convenience of the underlying name resolution mechanism. Therefore, in the Tilde naming system prototype, we constructed another mechanism for the identification of Tilde Trees.

The *Tilde Tree Registry* is a collection of information describing each of the Trees managed by the Tilde Computing Engine. Each Tilde Tree has an entry in the Registry that lists the Tree's distinguishing characteristics. The Registry accepts database-style queries and returns information about identified Trees. The Registry mechanism is coordinated with the user interface to the Tilde Forest, so that the results of Registry requests can be composed with Tilde Forest management commands into more powerful mechanisms. Figure 3 illustrates the information retained in the Tilde Tree Registry database. The information stored in the Registry includes the Tilde Tree's owner and creation date, a short comment describing the contents of the Tilde Tree, and the Tree's Tilde and Medusa Names.

4.4 Establishment of New Project Subtrees

Frequently, new subtrees are added to existing naming organizations to install new software subsystems, add a new user to a computing environment or develop an experimental version

Medusa Name	Tilde Name	Owner	Creation Date	Comment
blays0000	~system	binary	11/02/85	blays' system Tilde Tree
merlin0000	~system	binary	01/04/86	merlin's system Tilde Tree
blays0080	~edit	binary	01/10/86	experimental Edit
blays0079	~edit	binary	01/05/86	production Edit
merlin0132	~smith	smith	12/10/86	user smith's home Tilde Tree

Figure 3: Tilde Tree Registry Database

of an existing software subsystem. In a globally shared hierarchical naming system, each new subtree must be given a system-wide unique name.

Consider the development of a new version of a software subsystem, in an environment where there is a large user community that depends on the availability and stability of the original, or production, version of the subsystem. One solution is to create a copy of the subsystem's project subtree, which can then be modified without affecting the users of the original version. Figure 4 gives an example of a global naming hierarchy in which a new, experimental version of the `edit` project subtree has been installed under the directory `edit-exp`. In Figure 4, references to the components of the experimental version of the editor subsystem must all be changed from `/cmds/edit` to `/cmds/edit-exp` to reflect the new name of the root of the project subtree. In typical software subsystems, these references are scattered throughout the subsystem source code, and generated in obscure and arcane ways, so that it is inconvenient to locate and alter all these internal references.

In the Tilde naming environment, the experimental version of the editor can be created as a new Tilde Tree containing copies of all the components of the production Tilde Tree. The experimental Tilde Tree is entered into the Tilde Tree Registry with the same Tilde Name as the production Tree, but with a unique Medusa Name differentiating the experimental and production Trees. In this instance, then, a user developing the new version of the editor can bind the development Tree into his Tilde Forest under the Tilde Name `~edit`, while the rest of the user community will continue to bind the production Tree to the name `~edit`. Figure 5 gives an example of two processes, both of which use `~edit` to reference the production and experimental versions of the editor subsystem. In the figure, the two

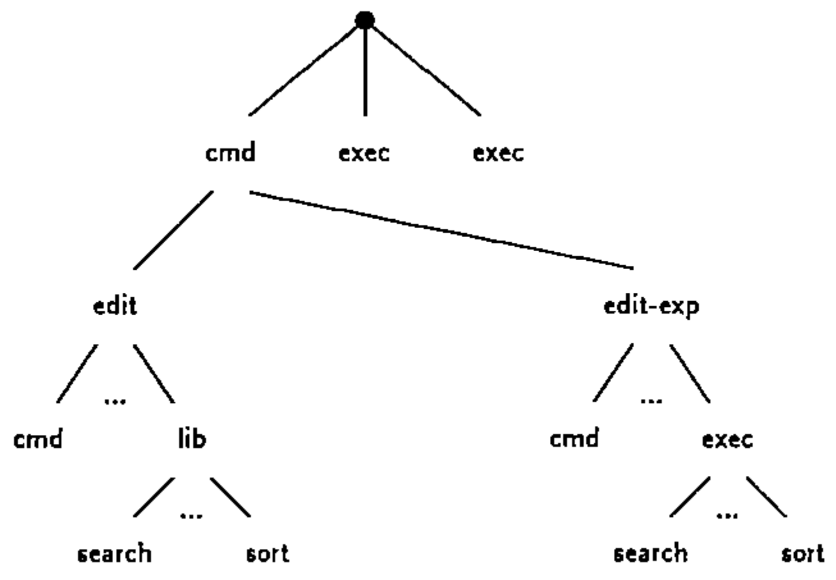

Figure 4: Creation of an Experimental Version of a Software Subsystem

Tilde Trees are identified by their respective Medusa Names. The two processes are able to use the two versions of the editor without making changes to the software subsystems; the bindings in the Tilde Forests cause internal references between editor system components to resolve to files in the appropriate Tilde Tree.

4.5 Use of Tilde Naming in the Client-Server Model

We have explained how the Tilde naming mechanism supports transparency through inheritance of the Tilde Forest by child processes. There is another mechanism for remote execution, called the Client-Server model, which uses inter-process communication between clients and servers for the execution of services, as opposed to the command interpreter's use of child processes. Inheritance of the Tilde Forest does not apply, since the two communicating processes do not share a common ancestor.

Members of the DASH project[Kor84] experimented with the sharing of Tilde Forests between Client and Server processes [Wil86]. The DASH system uses a Client-Server mechanism for service execution, in which Client processes on workstations request services from

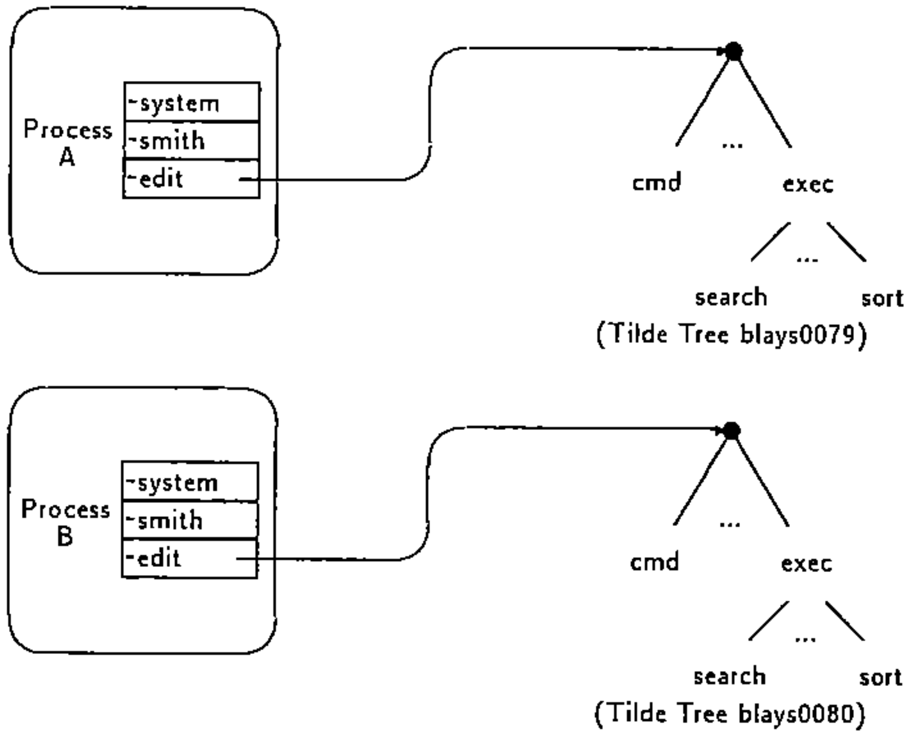

Figure 5: Creation of an Experimental Tilde Tree

Server processes on hosts. To preserve transparency, a Client must transmit its execution environment to the Server, so that the service can be performed in the expected environment. For example, a Client can share its Tilde Forest with a Server by first listing the Forest, sending a representation of the Forest bindings to the Server, and requesting that the Server establish those bindings in its Tilde Forest.

This mechanism for passing a Tilde Forest between processes is equivalent to the inheritance of an initial Tilde Forest by a child process. However, there is a potential for conflict between the Client and the Server, because the two processes may bind a particular Tilde Name to two different Tilde Trees. In practice, this conflict did not arise and was not thought to be a serious problem.

4.6 Organization of Tilde Trees as Project Subtrees

The set of Tilde Trees in the experimental system represents a reorganization of the files supplied with the UNIX operating system. The Trees are organized according to the project subtree paradigm, so that each Tilde Tree represents a separate software subsystem. Components of a software subsystem are collected into a single Tilde Tree, rather than stored in directories shared by many software subsystems.

As work on the experimental system progressed, we found that many software subsystems share a common internal organization. Reorganizing the files in these subsystems according to a standard structure creates a uniform external interface for the software packages. Components of software subsystems fall into seven categories[8]:

Commands - commands executed by the user; the system's external interface.

Executables - executable modules not directly invoked by the users.

Databases - text-oriented data files shared by the system components.

Libraries - collections of non-textual information (e.g., separately compiled modules)

Sources - used to construct the system components.

[8]These categories are defined based on our experience with UNIX-based software subsystems. We do not claim that this organization would be useful in other environments.

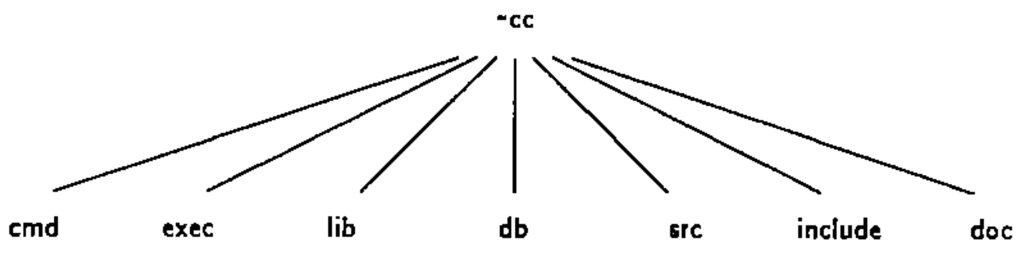

Figure 6: Skeletal Tilde Tree Structure

Included files - text modules shared among system source files.

Documentation - all documentation such as user manual entries and descriptive papers.

Figure 6 gives a diagram of the skeletal Tilde Tree structure showing the file names chosen for the directories representing each of the seven major components. The components of the C language subsystem illustrate the use of several of the component directories in Figures 7 and 8.

5 Conclusions

In the introduction to this paper, we state that the design of flexible, effective and convenient naming systems is a difficult problem, especially in distributed computing environments. Through our experience with distributed computing environments, we have established several important features desirable in naming mechanisms. The Tilde naming system incorporates those features by replacing the single, globally shared naming environment with local, per-process naming environments.

To investigate Tilde naming in a real-world environment, we have constructed an experimental network operating system that incorporates a prototype of the Tilde naming mechanism. This experimental computing environment includes modifications to the UNIX operating system kernel, modified versions of existing application programs and newly created software which supports significant computation such as software development.

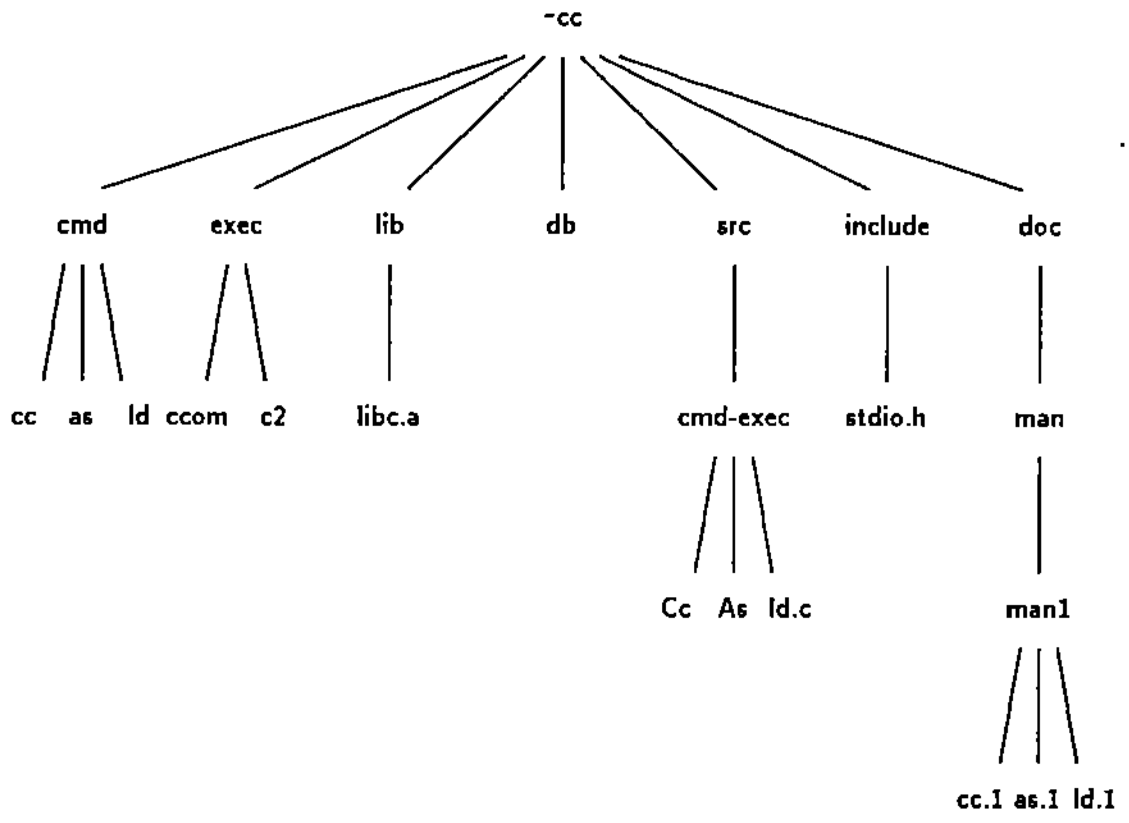

Figure 7: Structure of the C Compiler Tilde Tree

Component	Description	Location
cc,as,ld	application programs	~cc/cmd
ccom,c2	executable submodules of compiler	~cc/exec
libc.a	subroutine library	~cc/lib
Cc,As,ld.c	source code files	~cc/src/cmd-exec
stdio.h	shared header files	~cc/include
cc.1,ls.1,ld.1	online documentation	~cc/doc/man/man1

Figure 8: Components of the C Compiler Tilde Tree

Experience with the prototype implementation has supported the basic design of the Tilde naming system, and has directed our research into distributed naming systems. We have experimented with extensions to the user interface of the Tilde naming environment, and have constructed new software subsystems that utilize the enhanced portability provided by Tilde naming. As a result of our effort to convert existing UNIX software to the Tilde naming system, we have identified a useful strategy for the organization of the files within a Tilde Tree.

There are several directions for future work in this area:

- The current experimental system uses a broadcast mechanism to locate a Tilde Tree in the Computing Engine. This mechanism will not scale well to larger Computing Engines, and should be replaced by a dynamic location mechanism.
- Trusted software can be spoofed by the current Tilde Forest inheritance mechanism. To ensure security, a trusted process must be able to verify its file naming environment by examining and specifying its Tilde Forest.
- There is no inherent limitation that precludes the extension of the Tilde naming environment across administrative boundaries between TILDE Computing Engines. However, the details of Tilde Tree identification and location in an inter-Computing Engine environment will require changes in our basic assumptions. For example, a Tree's Medusa Name may be dependent on the Computing Engine at which the Tree is stored (but not on the Tree's specific location within the Computing Engine).

The Tilde naming system increases the transparency of a distributed system and enhances software portability by isolating the *identification* of a file from the *access* to that file. The local, per-process Tilde naming environment provides storage and resolution site transparency, and an abstract style of identifying collections of files with a single name. These advantages incur an additional overhead of name environment management; this overhead is minimized through careful design of the process and user interfaces to the naming mechanism. Through the use of an experimental computing environment based on the Tilde naming system, we have found Tilde naming to be a viable, productive and useful alternative to contemporary distributed naming systems.

References

[BSD83] *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1983.

[Com84] Comer, D., Transparent Integrated Local and Distributed Environments (TILDE) Project Overview, CSD-TR-466, Department of Computer Sciences, Purdue University, West Lafayette, IN, 1984.

[CoD85] Comer, D. and R. E. Droms, Tilde Trees in the UNIX Environment, *Proc. of the Winter 1985 Usenix Conf.*, Dallas, TX, Jan. 1985.

[CoM85] Comer, D. and T. P. Murtagh, The Tilde File Naming Scheme, Proceedings of the Sixth Int. Conf. on Distributed Computing Systems, Cambridge, MA, May, 1986 (509-514).

[Dro86] Droms, R. E., Naming of Files in Distributed Systems, Ph. D. dissertation, Department of Computer Sciences, Purdue University, West Lafayette, IN, 1986.

[Kor84] Korb, J. T., An Overview of the DASH Intelligent Terminal Project, CSD-TR-492, Department of Computer Sciences, Purdue University, West Lafayette, IN, Sep. 1984.

[QSP85] Quarterman, J. S., A. Silberschatz and J. L. Peterson, 4.2BSD and 4.3BSD as Examples of the UNIX System, *ACM Comp. Sur. 17*, 4 (Dec. 1985), 379-418.

[SUN85] *Networking on the Sun Workstation*, Sun Microsystems, Inc., Mountain View, CA, May 1985.

[Tho78] Thompson, K., UNIX Implementation, *The Bell System Technical Journal 57*, 6 (July-Aug. 1978), 1931-1946.

[Wil86] Wills, C. E., The Use of Services in the TILDE Environment, CSD-TR-656, Department of Computer Sciences, Purdue University, West Lafayette, IN, Dec. 1986.