Federated Services

SunSoft
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

March, 1993

Federated Services

The Challenge of Open Systems

Wide acceptance of open systems and standards in the computer and networking industry have enabled customers to take advantage of lower prices, higher performance, and new features by evolving ("rightsizing") their once relatively homogeneous, static, and weakly connected information systems into multi-vendor, dynamic, and highly connected networks.

In this fast-paced, rapidly changing environment, information system managers and architects are responsible for anticipating technological evolution, dealing with proprietary or legacy systems, and expanding networks to include a wide range of users with a variety of requirements. At the same time, managers must select solutions that increase functionality, increase user satisfaction, and lower costs. Application developers and vendors are faced with similar challenges, in that they must make choices regarding the platform they will deliver to. In many cases, the customer base is so heterogeneous that the application vendor must either deliver significant amounts of non-application platform technology with their product or deal with multiple underlying platforms to ensure that their market is large enough to derive a profit.

While open systems and standards provide more options for incorporating vendors and technologies into an information system, they do not address the problem that there is often more than one open standard that can serve for a given component of the information system. Furthermore, different vendors will choose to implement different combinations of those open standards. For example, three computer systems might each support two of three dominant transport protocols without having a single transport protocol that all three computer systems support. In a typical information system, the number of components is large and this phenomenon can quickly become limit options, resulting in an unhappy user community or inadequate leverage over the vendors.

The solution to these problems is not to force the industry to a single, narrowly defined set of standards because:

- Large networks are unlikely to agree on a single standard. Different enterprises will make different choices, and within many enterprises, different organizations will justify different choices.
- The existing base of distributed computing technology is large and the dominant players are each growing. Diversity will be a factor for many years to come.
- It is unlikely that any one system vendor will be able to afford the engineering investment required to address a large enough market to eliminate competition.
- Over time, there will be new vendors, new technologies, and new standards offering new competitive advantages to their adopters.

This paper outlines the approaches being taken in the SunSoft™ Solaris© operating environment to address these problems. This approach overcomes the limitations of the Open Systems model by permitting simultaneous, transparent integration of multiple distributed computing technologies under existing standard APIs and by promoting standards that help insulate information system mangers and application vendors from the "implementation details" of particular distributed computing platforms. This collection of technologies is referred to as *federated services*.

Definition of Federation

A federation of services uses interfaces and implementation techniques to enable transparent coexistence or cooperation of those services within systems on a network.

Federation is an evolutionary idea that recognizes and extends a technology trend that is taking place in the industry. For example, TLI and XTI are examples of federation through coexistence. These interfaces provide transport independence, so an application written to the System V Release 4 (SVr4) standard TLI or X/Open's standard XTI interface can use any of the transports available in the underlying system as needed to communicate with a counterpart on another system. This means a single ubiquitous transport is not required: full interoperability (at the transport layer) is achieved if each pair of systems that must communicate at least one transport in common.

An example of federation through cooperation is STREAMS. (STREAMS is the UNIX® SVr4 modular method of communicating between layers of protocol stacks.) STREAMS-based modules achieve cooperation by permitting dynamic “stacking.” For example, a terminal driver module might be pushed onto any one of a number of underlying device driver or network protocol modules to permit the applications that expect terminal semantics to use a variety of underlying devices or connectivity solutions.

A Model of Federation

Federation is typically accomplished by layering interfaces. Figure 1 shows a hypothetical federation of a set of services.

The bulk of application code is written to use higher-level APIs rather than APIs provided by either a specific distributed computing service or federation of those services. In Figure 1, the topmost block represents the APIs used by the bulk of application code. For example, in the UNIX operating environment, these APIs include the System V Interface Definition (SVID), the POSIX standards, the XPG standards defined by X/Open, and many other de facto and de jure standards.

The higher-level APIs hide implementation details that use the underlying service type or types. For example, in the UNIX system access to all filing services is through the standard `read`, `write`, … API; in the kernel, the file system implementation is hidden by the well-established *vnode* and *VFS* interfaces. Thus, applications are insulated from the types of underlying filing service that may be available, and new filing services are easily integrated into the system.

Because of this hiding, the introduction of federation to the system typically does *not* require code changes or recompilation for applications to see the benefit of access to a richer set of resources through a variety of underlying service types.

The typical federation requires the definition of a *Federation Interface*; this interface abstracts the essence of the service type that most (if not all) users of the interface will find appropriate and adequate.

Note – To avoid confusion, the term *federated service type* refers to different vendor service packages. Novell, Banyan, OSF, and SunSoft market different federated service *types*. Each vendor includes in their federated services package a unique combination of *service classes*. Filing, naming, security, etc., are *classes* of federated service.

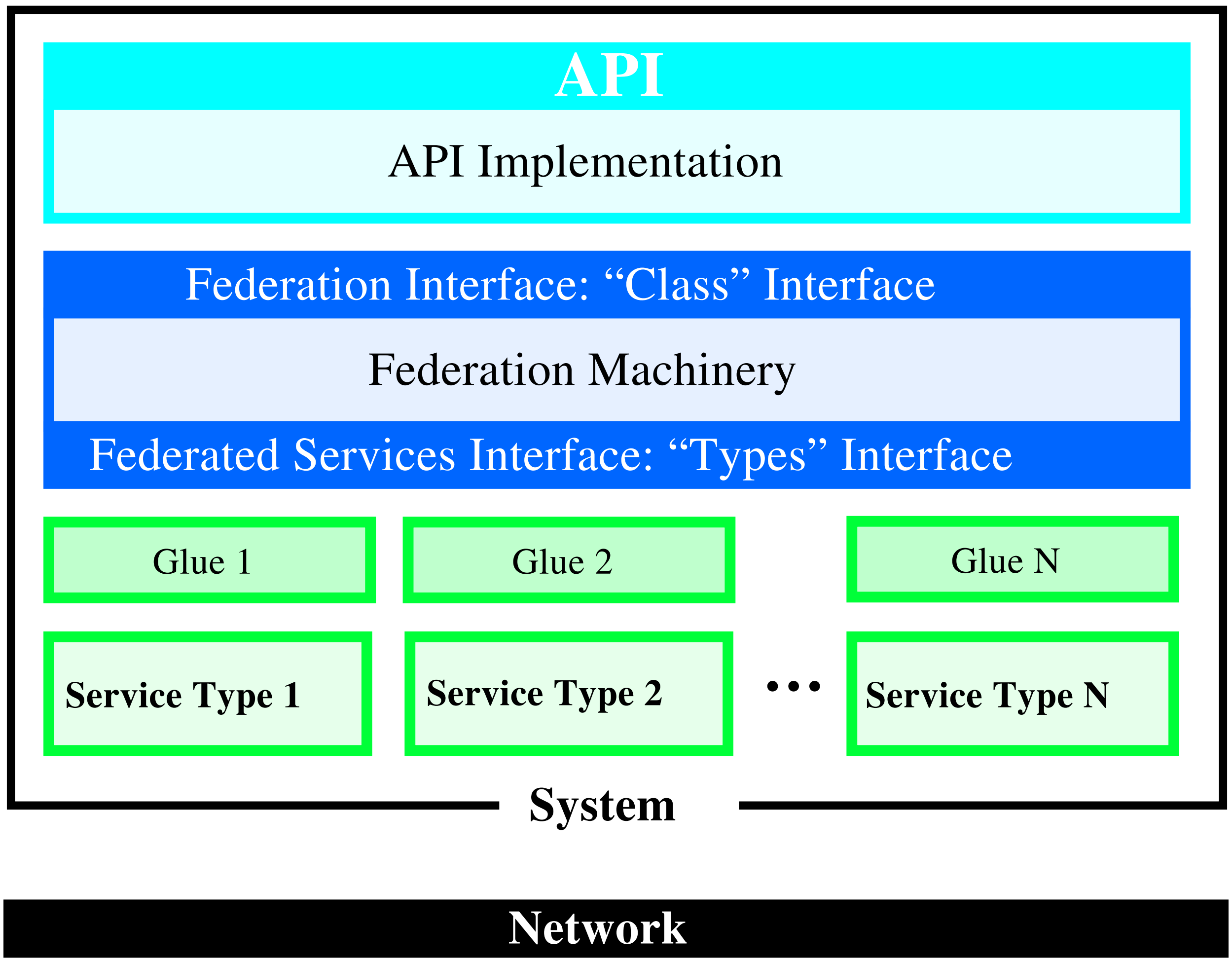


Figure 1 Generic Federation Model

Introduction of a Federation Interface does *not* preclude direct use of the underlying service interfaces. Existing applications that use a specific service type interface need not change when a federation of services is introduced into a system; however, these applications will be limited to the use of those specific services and will not benefit from federation.

Below the Federation Interface is the *Federation Machinery*: the code necessary to achieve coexistence or cooperation of underlying service types. It may simply act as a switchyard to direct calls through the Federation Interface to the appropriate underlying service type (coexistence), or translate calls through the Federation Interface into calls to more than one of the underlying service types (cooperation).

Specific service types are integrated into a federation through the *Federated Services Interface*. This interface is often a subset of or straightforward adaptation of the Federation Interface. Since service types usually do not provide the Federated Services interface directly, specific "adaptors" or glue modules that translate between the Federated Services interface and a given service type are often needed.

In simple terms, federation enables different system service types, like file services from different vendors, to at least coexist, perhaps even cooperate. Without federation, users must:

- Bear the expense of migrating to a new technology across all departments and organizations on the network.
- Bear the expense of migrating to new applications, perhaps without significant benefits.
- Bear the expense of developing and maintaining applications targeted to multiple underlying specific services.
- Live with a variety of network protocols and service types that do not interoperate, cooperate or coexist.
- Limit the scope of their applications by programming to a subset of the available services.

The benefits of the federation approach is to expand access to resources, break down the barriers to multi-vendor solutions, and support multiple protocol distributed computing, transparently to the user and to applications.

Federation does not provide interoperability in the sense that different service types communicate directly. Federation provides a higher level of interoperability in the sense that service types appear to be a single service provider. The current state of practice in filing is a an example: systems can support both ONC™ NFS® and AFS protocols to access a file tree, without any communication between NFS and AFS (even if one mounts an AFS file system on a directory in the NFS part of the tree). This apparent cooperation has been provided by carefully designing the vnode and VFS interfaces around the UNIX kernel's pathname interpretation machinery. We believe that similar cases of apparent cooperation are possible with existing technologies and that future products making use of federation interfaces will benefit from the ability to cooperate directly.

The SunSoft Transition to Federation

Federation is a logical extension of the SunSoft distributed computing philosophy. The demands for distributed computing in the 90's require:

- Global access to network resources.
- More support for increasingly mobile systems and their users.
- Simple and affordable networking solutions.

To meet these needs, SunSoft has developed the Solaris operating environment and evolved ONC to a next generation, ONC+™, as well as incorporating the federation of ONC+ services with other third-party distributed computing service packages such as NetWare®. Figure 2 illustrates Sun's evolution from NFS, to ONC, ONC+, federated services, and ultimately to Distributed Objects Everywhere (DOE).

The evolutionary process shown in Figure 2 starts with NFS's ability to perform filing tasks for heterogenous systems. ONC provided more services in this heterogenous environment. ONC+ expanded the scope of providing services to the heterogenous network from LANs to WANs while improving robustness and power. As ONC and ONC+ grew, the Solaris operating environment evolved and adopted more technology for cleanly integrating distributed computing into the operating system under existing APIs, and paved the way for the integration of additional distributed computing service types. As federated services mature, more services for the heterogenous network are being integrated into the Solaris operating environment.

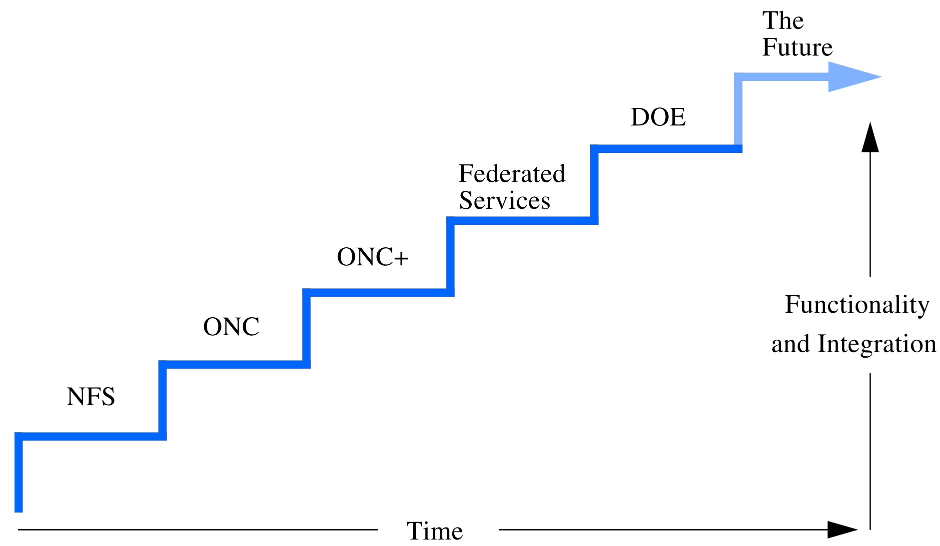


Figure 2 The SunSoft Evolution to Federated Services and Beyond

DOE is a logical extension of federation. DOE benefits from underlying federation technologies that enable it to operate in a wider range of distributed computing environments. More importantly, the DOE focus on interface definition technology for objects brings the power of federation to applications.

Standards

Successful development of federated services interfaces will require cooperation within the industry. Federation interfaces must be industry standards to realize their fullest value. An important goal of the SunSoft federation activities is to be a part of the industry standardization processes.

Where possible, SunSoft has adopted existing standard technologies. Where no solution currently exists, SunSoft is working with standards bodies to develop appropriate solutions.

The federation architecture being developed by SunSoft is consistent with the UNIX International ATLAS model for distributed computing.

Key Federated Services

The Solaris federated services architecture enables the federation of a variety of distributed computing service classes such as filing and naming. This document focuses on the key service classes presently prevalent in distributed computing environments that are candidates for federation. These key service classes are:

- Filing
- Naming
- Security
- Communication

In the sections that follow, we discuss approaches to federation of each of these services.

File Services

The UNIX operating system uses the file system and file system APIs as a fundamental abstraction mechanism: many things appear as part of the file system. For example, devices and interprocess communication mechanisms are treated as files. Federated file services preserves this aspect of the UNIX operating system.

Originally, the UNIX file system implementation was so much a part of the kernel that only one file system 'type' was feasible. Some implementations of UNIX differentiated themselves in part by having a different file system type (for example, UCB's Fast File System), but were still limited to that single type. In the early 1980's, the UNIX community recognized the need for multiple, coexisting file systems primarily due to interest in incorporating a distributed file service into their system without displacing the existing file system type. In the second half of the 1980's, the vnode and VFS interfaces began to emerge as de facto standards for incorporating new file systems types into the UNIX operating system, based on the success of NFS in both UNIX and non-UNIX operating systems. Many of the non-UNIX implementations of NFS either adapted to interfaces that played a role similar to that of the vnode and VFS interfaces in the host operating system, or modified the host operating system to add analogous interfaces.

Figure 3 illustrates the Solaris SVR4 implementation of the vnode and VFS interfaces.

Achieving Cooperation

The vnode and VFS interfaces provide federation of file systems by permitting coexistence. However, to be transparent, a minimal form of cooperation is required: the file tree must appear to be a single, large file system to the applications and users (at least those not concerned with system administration). Much of the machinery for this already existed. Since the UNIX system originally used the concept of "mounting" as a means to construct a single file tree from file trees residing on multiple disk drives, extending the machinery to enable mount points leading to a different file system type was straightforward. Thus, in the execution of `open`, the kernel may access a number of different file system types before it obtains the vnode for the file being opened. Subsequent `read`, `write`, … operations go directly to the service type for the file, and involve no further accesses to the file system types used to locate the file. This is a case where cooperation is a useful illusion: the file system types do not communicate directly, but are administered so the kernel can unify them.

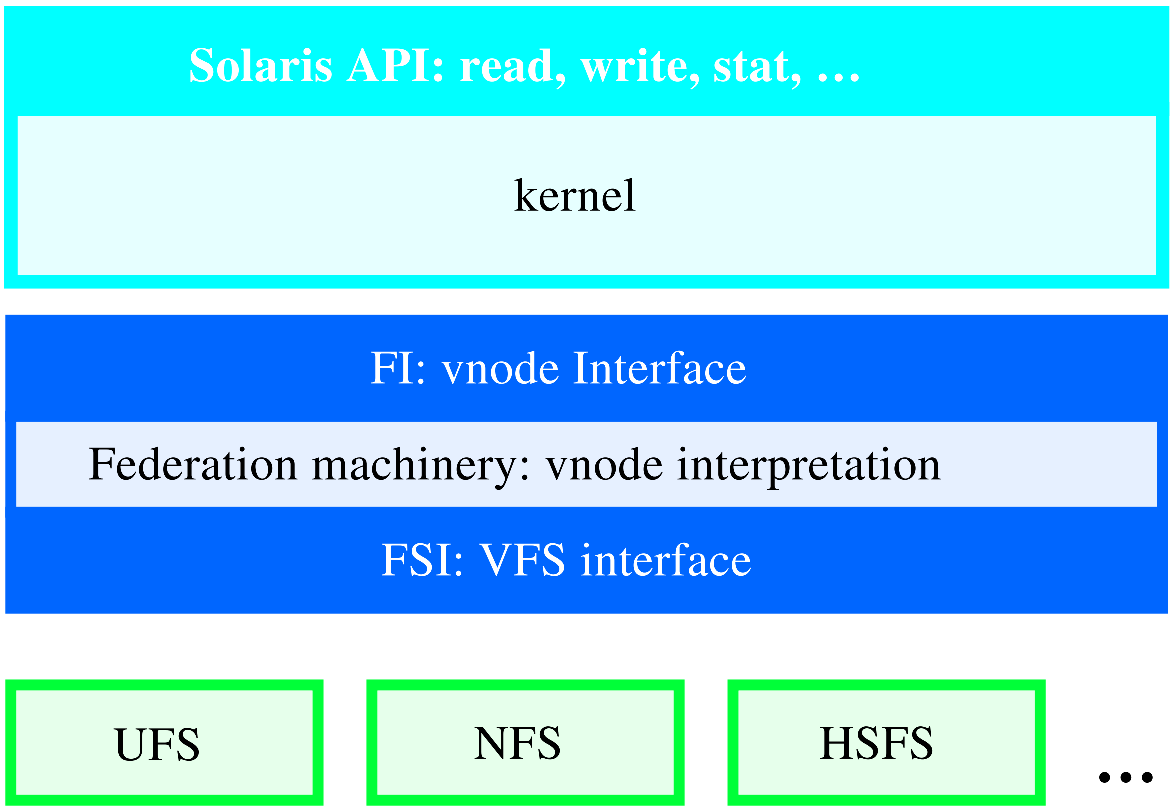
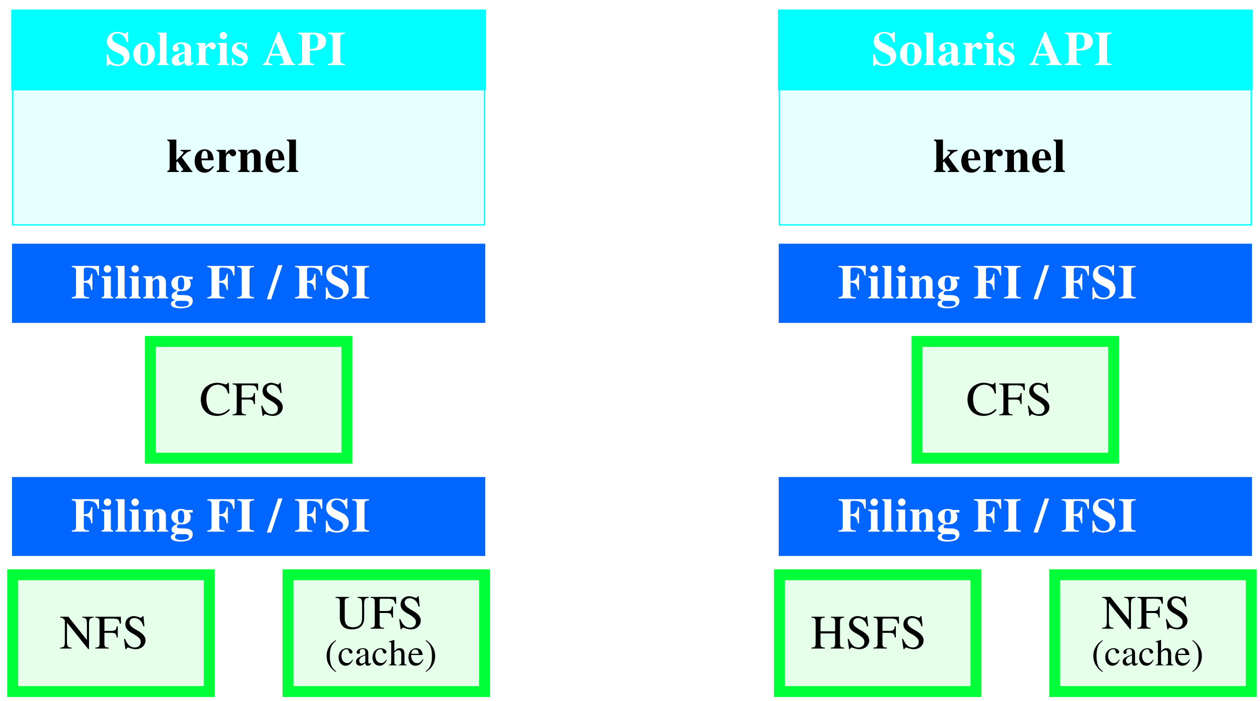


Figure 3 Solaris SVR4 Implementation of the vnode Interface

Direct cooperation between file system types is both possible and useful with the vnode and VFS interfaces: file systems can be layered, much in the spirit of STREAMS but without dynamic stacking and unstacking. This means that new features can be provided ubiquitously without having to change existing file system services.



Cache files from NFS service to a local UFS
- **Improves client performance**
- **Off-loads the server**
  - **more clients per server**
- **Off-loads the net**
  - **more clients per net**
  - **better WAN utilization**

Cache files from CD-ROM to NFS.
- **Improves client performance**
- **Can "fake" writability**

Figure 4 Cache File System

An example of direct cooperation is the Solaris Cache File System (CFS) shown in Figure 4. The CFS acts in much the same way a memory cache does: it tries to keep frequently used files in a fast file system type, while maintaining consistency with files in a slow file system type. When used to cache NFS files to a local UFS (Figure 4 left), the CFS can improve client, server, and network performance by keeping frequently used files on media with faster access and by off-loading the server and network. In the case of slow network media (for example, phone lines), it can bring client performance up to a usable level.

Since the CFS plugs into the VFS interface and relies on the vnode interface for underlying services, it can be configured to provide similar advantages for other combinations of file system types. With some customization, it can also be used in other situations. For example, if the cache is preserved across system restarts and files are not written back to the slow file system type, then CFS can be used to provide the appearance of writing to read-only media (Figure 4 right). This feature is useful for software or data distributed on CD-ROM where the user wants to personalize or update the software or data without making a complete copy of the CD-ROM.

The current vnode and VFS interfaces have proven themselves in service and have enabled value-added specialized file system types from system software vendors, successful businesses for NFS vendors, and businesses in other value-added file system types. Based on this usage, several areas for improvement of the interfaces have been identified, and a UNIX International working group is developing requirements for further evolution of the interfaces. New features of evolved vnode and VFS interfaces are likely to include:

- The ability to dynamically change layering (that is, insert and remove layers without having to rebuild the kernel or modules being layered)
- Interface improvements in the area of file service synchronization
- Cleaner interfaces between the VFS and kernel services such as virtual memory

These new features will enhance the interfaces as federation technology.

Naming Service

Naming will play an increasingly prominent role in the future of distributed computing. Operating systems will have to evolve to integrate name services under their existing APIs and to make naming a more fundamental part of the system architecture. The UNIX operating system began this evolution in the early 1980s; the Solaris operating environment continues that evolution with the development and adoption of federation technologies for naming services.

The Need for a Naming Service

In its early forms, the UNIX operating system used the contents of files, such as `/etc/passwd`, to provide naming services. This practice continued for a time, even after the development of networking features. Problems inherent in using this approach in a distributed systems environment are:

- Updates—No mechanism or policy exists for synchronizing updates to either a single, shared copy of the file, or to many distributed copies of the file.
- Throughput and capacity—If all systems share a single, common file, a bottleneck can develop. If each system owns a copy of the file, network bandwidth is consumed distributing new copies of the file for updates.
- Availability—If all systems share a single, common file, their ability to provide service depends upon the availability of that file. If each system has a copy of the file, then a failure of any system can affect the ability to update information for the other systems
- Autonomy—Neither shared files nor distributed copies of a single file subdivide the control of information that is used locally and that used by all systems such that portions of the file can be managed by (and only by) the administrator responsible for that portion.

Of course, these problems could be solved by adding rules for accessing the files and utilities for distributing and updating them. However, this is essentially what many name services do, so it makes sense to use a name service to meet these requirements rather than develop a specialized approach for each case. This was recognized from the start of the NFS's design. As a consequence, the Network Information Service (NIS, originally known as the Yellow Pages) was created for the first releases of NFS. Although NIS was primitive, it served adequately to replace the naming functionality of several

/etc files. Further evolution incorporated access to DNS naming services for naming within and beyond the enterprise, and led to the development of a more powerful naming service (NIS+) for ONC+. Meanwhile, the Solaris operating environment developed and adopted more facilities for integrating naming services under its existing APIs.

Hiding the Name Service

The solution to the problem of integrating naming services into the UNIX operating system was aided by the fact that most applications had followed the good programming practice of “data hiding” by using library functions to access files with naming information rather than accessing the files directly. Thus, the problem of replacing /etc/passwd with a name space managed by a naming service involved (in part) rewriting some library functions (for example, getpwent), and fixing some programs that had not yet adopted good practice. Since programs were statically linked to the libraries, it was necessary to relink, but not recompile, to get access to the name service. With the advent of dynamically linked shared libraries, it is now possible to change these library functions, say to use a different naming service, without having to touch the applications at all.

Hiding access to a single underlying name service is not adequate. In fact, the first conversion of getpwent to NIS looked first in /etc/passwd, then queried NIS. The reason for this was threefold: many early customers of NFS did not want to convert to NIS, those that did wanted make a smooth transition from the use of files to the use of NIS, and they also wanted a way for the local system to override the network. Later practice showed that many customers wanted to expand the search to include DNS so they could manage their local name spaces with NIS and default to a name service managed elsewhere for enterprise-wide or beyond the enterprise naming.

The need to deal with more than one name service resulted in the creation of the name service switch (NSS). The NSS is the Solaris implementation of concepts that have been used in other operating systems. The NSS moves policy out of the library functions and out of NIS (which has embedded a policy of forwarding requests it cannot satisfy to DNS), so system administrators can configure the policy without having to rewrite library functions. For example, a policy for a site moving from NIS to NIS+ might be to first look in the local file, then in NIS, then in NIS+; eventually, the order of search could favor NIS+ over NIS, and eventually NIS could be eliminated.

The NSS also makes introducing new naming services easier by better isolating the code changes required. Figure 5 illustrates the name service switch as it would come into play for a particular library function.

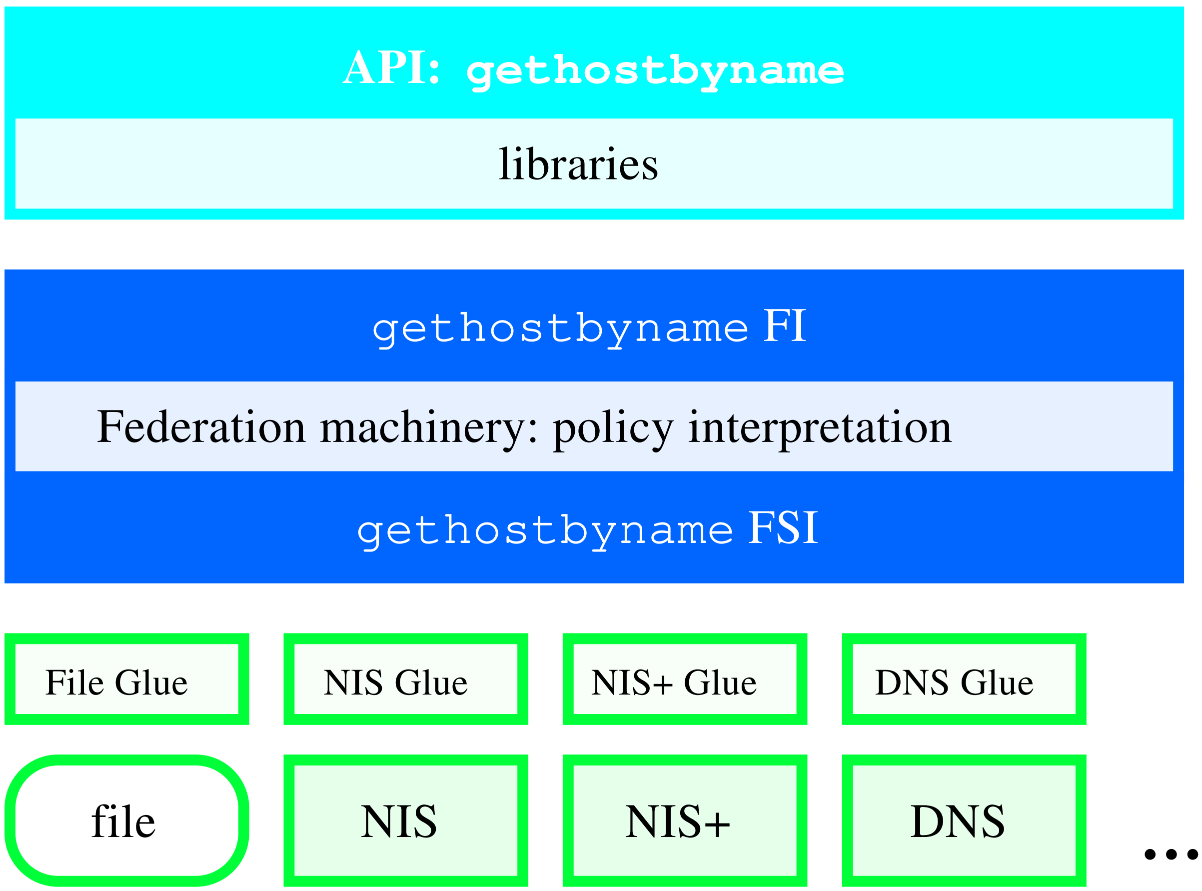


Figure 5 Name Service Switch

The NSS is an example of federation through the use of implementation techniques rather than the definition of a generic interface. Applications or services that define new name spaces should use this technique. This involves designing application-specific interfaces to the name space, implementing the application-specific interfaces so they can either initially or in subsequent releases make use of alternative or multiple naming services, and delivering the functions in dynamically linked shared libraries.

The Future Role of Naming

The role of naming in the UNIX system has been limited by the fact that it was built using the file system as a basic abstraction mechanism. In the future, it will be possible to name and interact with things more complex than files: namely, objects. In the world of objects, naming takes on increased importance because objects themselves may export naming interfaces. For example, a spreadsheet application may export its cells as objects that can be named by other applications. This cannot be handled efficiently by having the spreadsheet register its cells in the network naming service; that would be slow and might overload the naming service. Instead, the spreadsheet will have to implement a naming service and somehow integrate its name space into a system-wide name space, which includes the basic name spaces supported by the network, plus the name spaces provided by other application objects.

This points to the need for a "base type" naming interface that objects can inherit in order to export naming as part of their interface. In procedural programming terms, this is a simple, generic interface to naming that essentially becomes both the Federation Interface and Federated Services Interface, permitting a multiplicity of naming services to cooperate and provide what appears to be a uniform, rich name space.

Work on such interfaces is underway on more than one front. The naming interfaces being built into distributed object management by SunSoft will have this property, and discussions are being conducted within X/Open and UNIX International regarding a procedural "Simple Naming Interface" or "Federated Naming Interface."

The implementation of Solaris Federated Naming is illustrated in Figure 6. One of the key features of federated naming is the ability to use *composite names* — names that consist of elements take from more than one name space. The composite naming machinery resolves names by parsing the elements and directing requests to underlying name services. This is analogous to the way the kernel drives pathname resolution using the vnode interface.

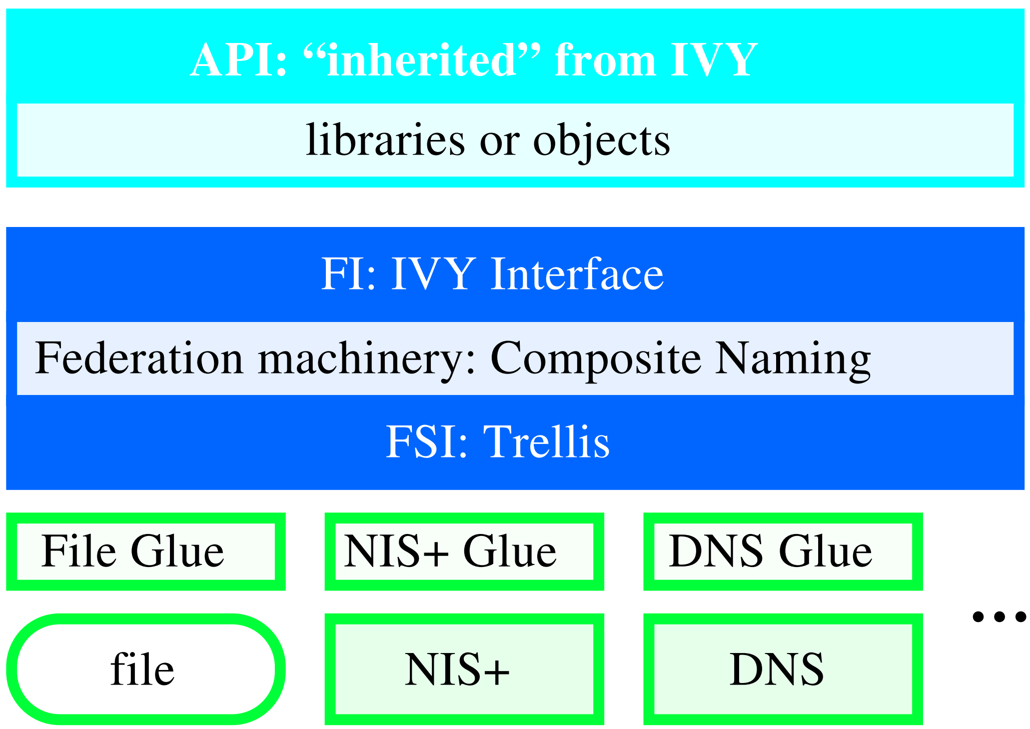


Figure 6 Solaris Federated Naming Interface (IVY)

Applying Federated Naming Technology Today

If the UNIX file system were re-implemented in the spirit of objects using the ideas outlined above, the naming portion of the federated filing interfaces would be inherited from a federated naming base type and naming of file objects would be seamlessly integrated with naming of all objects. Some of the advantages of integrating true naming services into the file system are illustrated by the technique the Solaris operating environment uses to hide some of the annoying aspects that result from scaling the UNIX file system implementation to a large network.

In the early days of the UNIX operating system, the users' file space requirements eventually grew to the point that more than one disk was needed to hold them. Administrators solved this problem each in their own way. For example, some moved users files from `/usr` to `/usr1`, `/usr2`, … while others moved them to `/usr/disk1`, `/usr/disk2`, …. An analogous problem occurs when one considers the file tree in a large network: the name for a user's home directory might look something like:

```
/net/server2/disk4/jruser
```

This pathname routes `open` requests to a specific server (`server2`), a specific disk partition on that server (`disk4`), and finally to the directory that contains the user's files (`jruser`). While this routing is necessary at some level, accomplishing it with the file naming machinery exposes administrative details that confuse the user and makes the inevitable task of changing (administering) systems and the network more difficult, as users come to rely on these administrative details to get to their own and to co-worker's files.

This problem can be eliminated by the introduction of a file system type we call AutoFS. The AutoFS uses the vnode, VFS, and federated naming interfaces as shown in Figure 7. To solve the problem outlined above, an AutoFS is mounted on `/home` and is pointed to a name space that resolves usernames to home directories. AutoFS uses a name service to look up the pathname element passed to it, creates a directory with the element name, mounts the directory returned by the lookup on the directory just created, then returns the vnode for that mount point. By this device, the name for any user's home directory becomes simple and constant:

```
/home/jruser
```

The management of the name space used to accomplish this is automatic in that it is coupled with the normal management tools for users' working directories.

Since the AutoFS uses federation interfaces to access the naming service, it can work with a variety of underlying services. The file systems mounts can be of any type, making AutoFS a valuable tool for unifying file trees being exported from a variety of distributed computing platforms. Further benefits are derived from the fact that use of a name service centralizes the administrative control and leverages the update, availability and hierarchical control features often present in a name service. This mode of operation was made the default with Solaris 2.0, and AutoFS will available as part of ONC+ in late 1993.

[*Caveat: the implementation presented here differs from that of the Solaris 2.0, 2.1 and 2.2 releases, but will match that of releases 2.3 and beyond.*]

While this implementation may not be as elegant as would be possible had the system been originally designed using object-oriented principles, it is a practical example of how the power of federation can be used to enhance existing APIs and practice.

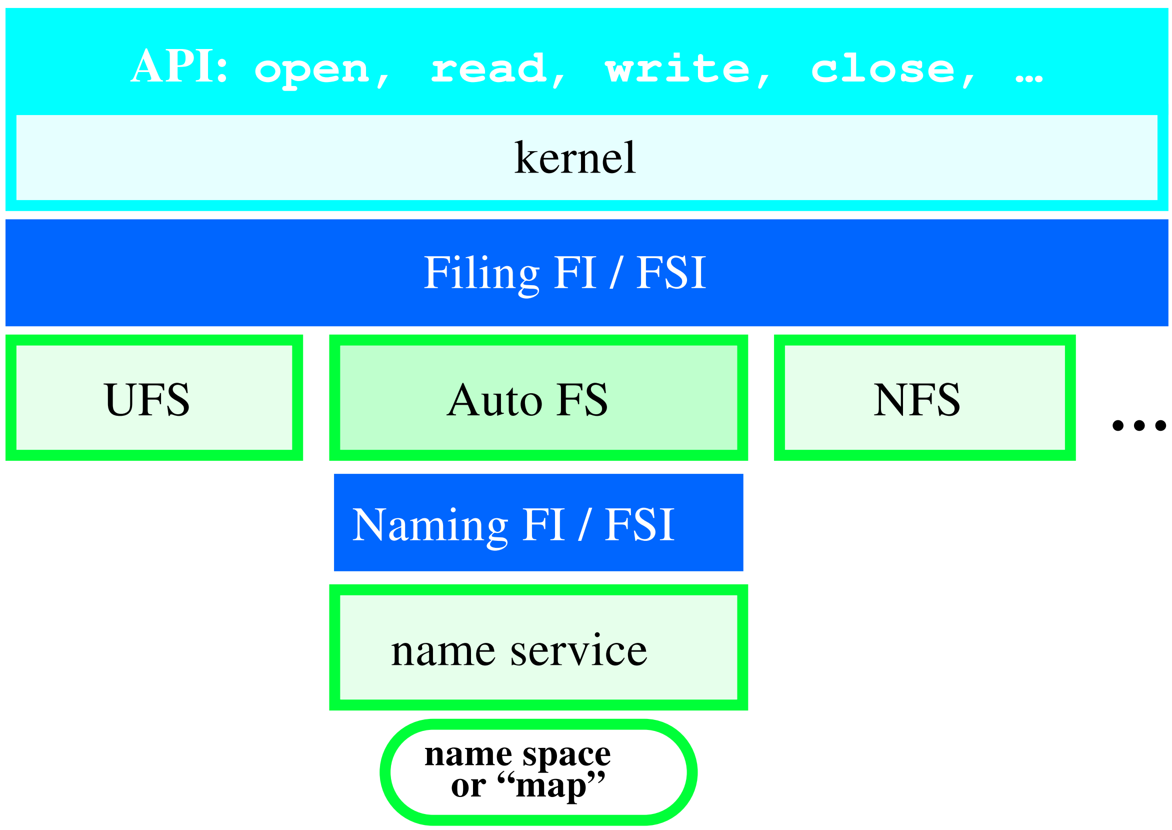


Figure 7 AutoFS Integrating Naming and Filing

Security Services

Many of the security mechanisms developed for mainframe systems no longer apply in the world of distributed computing. Systems on a heterogeneous network probably do not share a common operating system, and therefore have different views of what security is and how it is implemented, if at all. Furthermore, the network may be exposed to the possibility of passive listening or active masquerading, since it cannot always be physically secure.

The fundamental mechanism for implementing security in a distributed system is cryptography, or the "scrambling" of information in such a way that unscrambling is difficult for any but the intended recipient. With cryptography, two communicating parties can assure that they are in fact communicating with their intended party. This mutual assurance of identity is called *authentication*. Authentication is a fundamental service for secure distributed computing, since most other security measures rely on it. The logical placement for access to authentication services in a distributed computing architecture is with the communication service. It will then come into play naturally and transparently as part of passing information between entities that can be authenticated. Authentication services are not used solely by communication services, but other uses of authentication services deal with similar issues.

Accommodating multiple authentication schemes is necessary because:

- Encryption technology eventually becomes obsolete; the mathematics of a particular scheme may be found susceptible to compromise, or advances in computing power may enable cracking the scheme in a short enough time to make it useless.
- Different users have different requirements; the most protective schemes will be computationally expensive and require years to break, but many needs can be served with less computationally expensive schemes. Even different accesses to the same service (for example, filing) by the same user may have differing security requirements.
- If there are multiple communication services, each one should be able to adopt whatever authentication schemes are deemed appropriate by the site manager, thereby enabling common administration of security.

There is promising work going on in the industry to address some of these problems. The Internet Engineering Task Force (IETF) Common Authentication Technology workgroup is considering a Generic Security Service Application Programming Interface (GSSAPI). X/Open has placed GSSAPI on their agenda.

This interface would insulate the application from the need to choose a specific authentication flavor, and permit administrators to plug in the authentication flavors that best meets their mix of needs. Hopefully, suppliers of distributed computing technology will move toward adoption of a common means for introducing and accessing authentication flavors; if only one or a few make the move, there is still significant end user advantage in that those vendors' products can adapt to coexist with less flexible products.

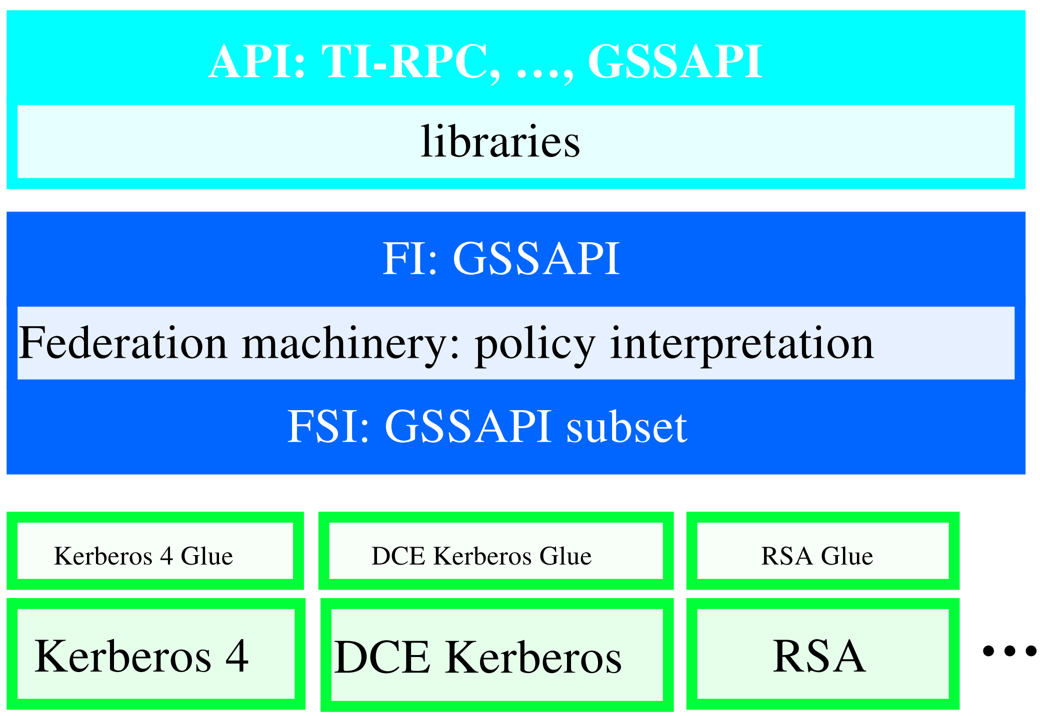


Figure 8 Federated Security Interfaces

Beyond Authentication

Authentication is fundamental, because most other security mechanisms build on the assumption that authentication is in place. For example, security measures known as *authorization* answer questions of the form "can α perform operation γ on β" under the assumption that the identities of α and β are authentic. These higher level security measures tend to be service or application specific, and common ground is likely to be found only within a class of services or applications. The European Computer Manufacturers' Association (ECMA) model for security suggests machinery to deal with some of these higher level issues. This may be an area for further work once the more fundamental issues of authentication are addressed.

Communication Services

Most applications today are not distributed; they simply use a small number of distributed services through interfaces that hide the fact the service may be provided from a remote machine. To be distributed, the application itself must be designed as a collection of two or more modules that (potentially) reside on different machines and communicate among themselves to cooperatively solve problems.

Distributed Application Structure

Distributed services are examples of applications, so considering their structure may help gain an understanding of the issues in building distributed applications. The NFS implementation, for example, consists of two modules: the NFS client side and the NFS server side, as illustrated in Figure 9.

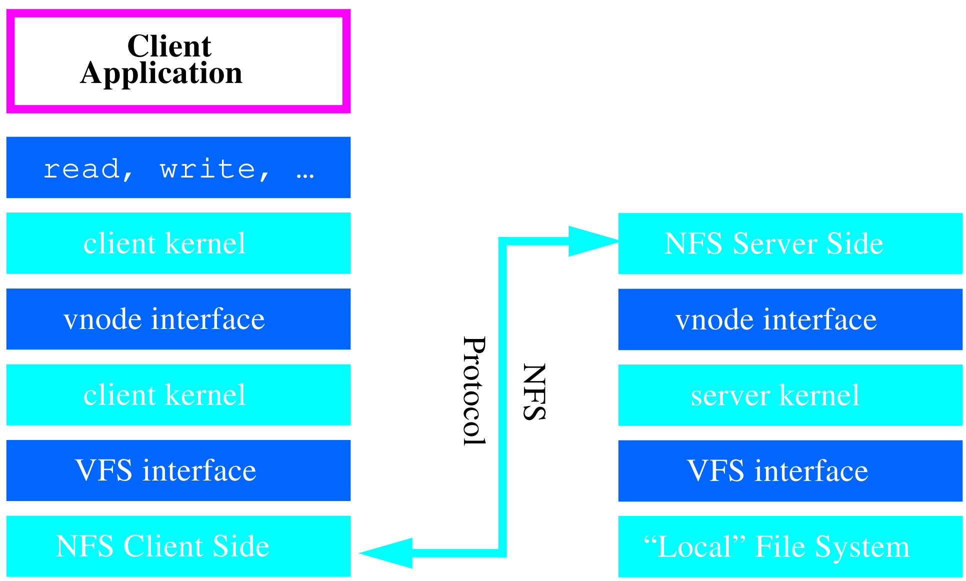


Figure 9 Structure of the NFS "Application"

The NFS client side code plugs into the VFS federated services interface (or whatever analogous interface is provided by the client side operating system). The NFS server side exists on top of the vnode federation (or analogous) interface; in this way, the NFS server does not depend upon a specific file system type. Since NFS clients communicate only with NFS servers and vice-versa, the choice of how they communicate is private to that service.

A different example of distributed application structure is a name service: a typical name service, such as NIS+, is replicated to provide better performance and high availability. In this case, there are multiple server side peer modules that are essentially identical in terms of the code they execute and the data they store, though they may have an agreement among them that determines the definitive instance of the service so they can resolve consistency issues. The client application views the service as being provided by a single entity, while the service application side implements machinery to dynamically attach to the most available service side. Again, the choice of how the service components communicate is private.

Most distributed services and applications are designed around the client/server or client/server peer model and communicate privately among modules within that service or application type. Thus, for the most trivial form of communication services federation, multiple communication services can and do coexist in systems and on the network. While some economies may result from trying to limit the number of communication services used, there will always be a requirement for multiple communication services for vendors addressing diverse markets, and for end users managing large, heterogeneous networks. If handled correctly, the costs of having multiple communication services can be minimal or essentially nil (for example, see the section "Federated Authentication").

Communication Service Layering

The development of communication services is a history of defining, refining, and then hiding the layers of mechanisms used to communicate. Early efforts were essentially experiments. Over time the industry moved toward standardization of layers and a model for layering. Applications and services have evolved through modification or replacement to use higher layers of the model in order to work in a wider range of networking environments. TLI and XTI are examples of federation technology evolved to achieve protocol independence at the transport layer. These interfaces are gaining wide acceptance by a vendor community that recognizes the problems of dealing with multiple transports as a cost of doing business.

The STREAMS technology further enhances protocol layering by defining interfaces for dynamically stacking and unstacking protocol modules to allow more flexible construction of protocol stacks. For example, one could stack either a TTY module or a TCP/IP module on an asynchronous port driver module.

Communication and Application Structure

Communication services discussed so far do little to address the problem of application structure. Remote procedure call (RPC) technology evolved to give developers a familiar program structuring paradigm to apply to distributed applications. RPC technology caught on rapidly; almost all recently developed distributed services and applications make use of some form of RPC technology.

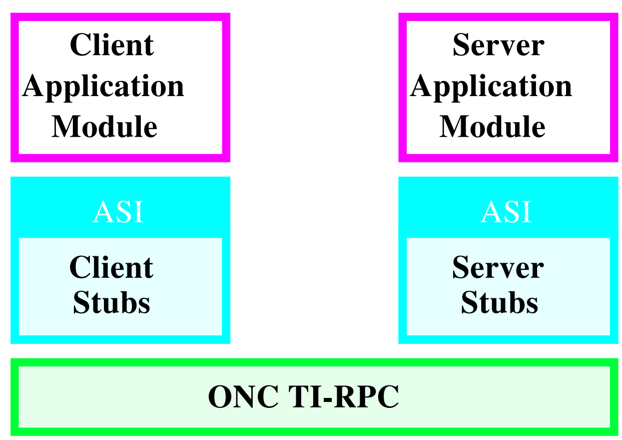


Figure 10 Early Use of RPC Technology

While some early distributed applications were designed and developed directly to an RPC interface, designers and developers quickly realized that the important interface was the one being built *over* the RPC; namely, the interface that defined the respective roles of application modules. We refer to these as

Application-Specific Interfaces (ASIs). Good development methodology identified the ASIs so library functions could be implemented to provide the ASIs and hide the RPC calls, as shown in Figure 10.

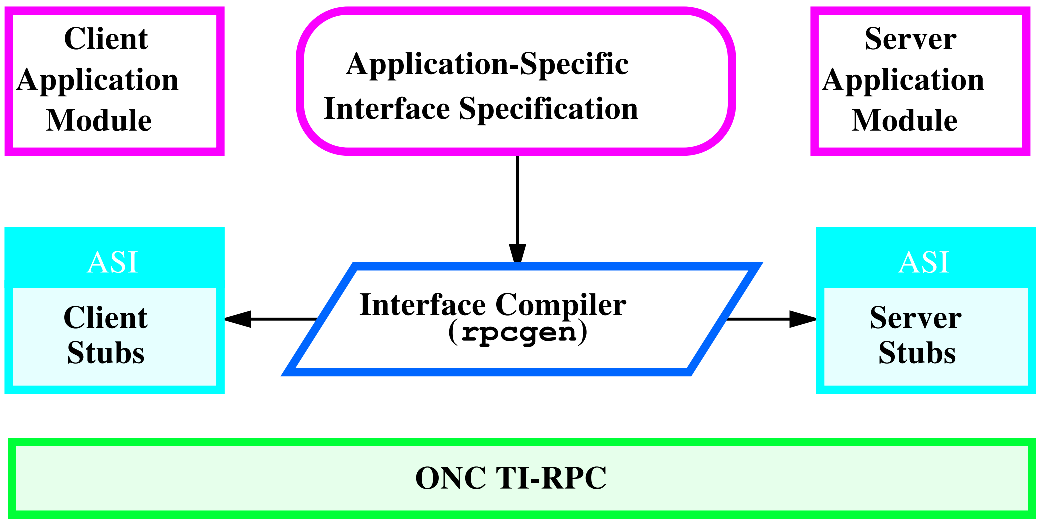


Figure 11 Early Interface Compiler Technology

Developers quickly realized that, even with this discipline, writing RPC calls was unpleasant and mechanical. So they built interface compilers (also called "protocol compilers") to take an interface specification for an ASI as input and generate source code for the ASI library functions, which were then processed by their favorite programming language compiler. Figure 11 illustrates layering using interface compiler technology. One of the early interface compilers was `rpcgen` for ONC; the first usable version was developed in about a week using commonly available compiler generator tools.

The early interface compilers were very RPC specific; that is, though they relieved the designers and developers from writing the RPC calls directly, they still had features that exposed details of which RPC was used to the higher layers of the software. More recent interface compilers have come a long way toward providing RPC independence at the application source level. These compilers can target any one of several communication mechanisms and the choice of RPC can be delayed or changed, as shown in Figure 12. An example of this technology is the ONC+ RPCTool, which targets ONC+ TI-RPC, and is available from NetWise to target other mechanisms. Similar products are available from other ISVs, such as NobleNet EZ-RPC. This represents approximately the state of the art in distributed applications structuring and communications, but the technology continues to move forward to create a more powerful distributed applications design, development, test, and execution environment.

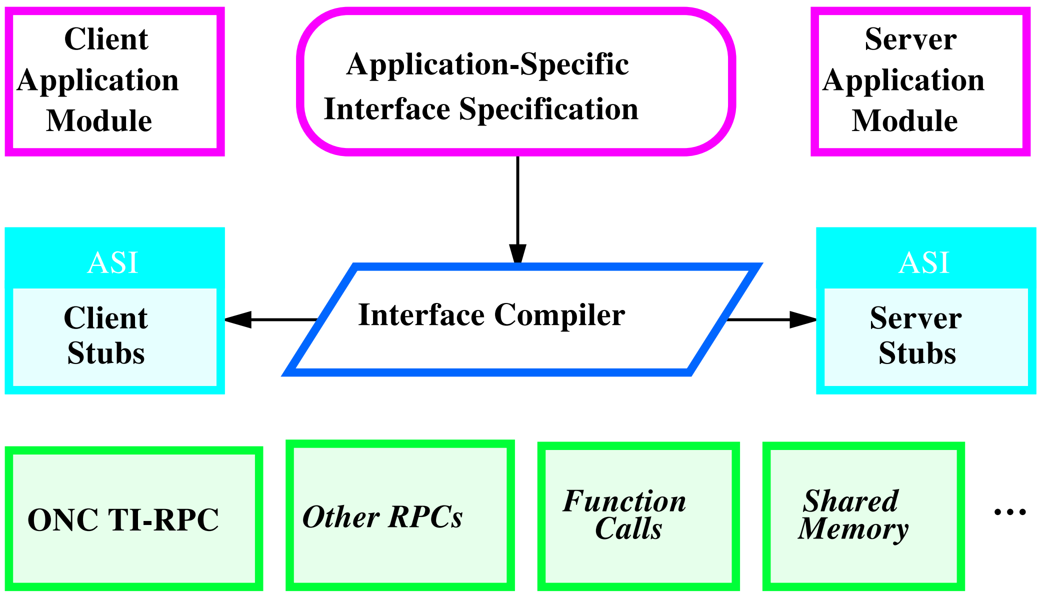


Figure 12 Federated Communication Services

One of the primary advantages of not targeting a specific RPC is that once the application is up and running, one may decide that the partitioning of the application was incorrect and that it would be more appropriate to collocate modules and use direct function calls, shared memory or other means of communication. Interface compilers that provide this flexibility should be available soon. This achieves federation to the extent that there is freedom to select one of many technologies as a compile-time option. Since this does not involve the adoption of a generic interface to the underlying service types, it is another example of federation as an implementation technique.

To realize the full benefit of this federation technology, it must be used extensively. Since interface specification technology need not be specific to RPCs, one can define many ASIs without a preconceived notion of where the application crosses a network boundary. Figure 13 illustrates an application structured with many client and server modules with many interfaces. This approach has a number of advantages:

- The design and development methodology is forced to formalize the specification of application structure and interfaces early
- The application can be tuned once it is operational, since there are many options for collocation and breakage
- Choice of communication technology becomes more a business issue than a technical one
- It is a good intermediate step towards moving the application, designers, developers, and development methodology into an object-oriented environment

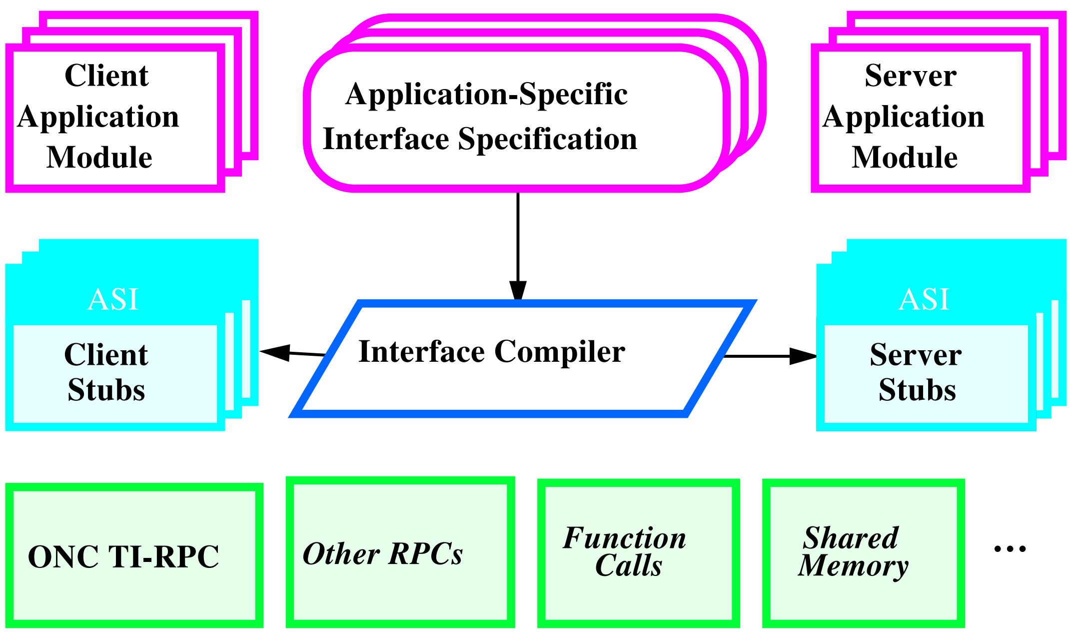


Figure 13 Federated Communication Evolving to Objects

A next step in this evolution is to have the interface compiler target a system that does run-time selection of the communication technology. This permits dynamic tuning since the communication technology can be selected based on what best fits the situation. For example, if the modules are collocated, they may use shared memory or function calls without excluding the possibility that they can also use RPCs to communicate with remote instances. This represents a higher level of communication federation in that applications can use multiple underlying services without recompilation.

One may note that the level of functionality begins to approximate certain aspects of DOE, thereby completing an evolutionary linkage to what SunSoft sees as the future of distributed computing. One of the most important steps developers can take today to move toward objects is to begin to incorporate interface specification technology into their design and development methodology, even if it means "hand-compiling" the interface specifications. The market will probably support a variety of interface specification languages and interface compilers. The most likely candidate for dominant share is OMG IDL. However, adoption of interface compiler technology in some form is the most important decision an application developer can make. Even if porting to a different interface specification language and tool set is necessary at some later time, the cost of doing this would be small compared to the value of the investment in methodology and application architecture, as well as the productivity gains from early adoption of tools.

Conclusion

Federation is a recognition of existing APIs, techniques and evolutionary trends in the computing industry that address problems of dealing with multiplicity. In particular, federation preserves existing investments, increases flexibility for application of existing technologies, and opens opportunities for further innovation.

The Solaris distributed computing federation architecture has evolved through long experience in adding value through existing APIs and providing means to add new functionality in the form of ONC. In evolving to ONC+, increased ability to accommodate multiple distributed computing technologies has been a feature, but it was also a necessity for support of ONC features being replaced. As a result, the Solaris operating environment has the strongest architecture for integrated distributed computing, support of alternative distributed computing platforms, and for support of multiple distributed computing platforms.

The Solaris distributed computing federation architecture also represents a safe investment, since it is based on a combination of standard interfaces, openly licensed technologies and widely accepted implementation techniques. Little is unique to the Solaris operating environment other than early adoption of technologies and early recognition of how to best apply them; others are likely to follow this lead. Furthermore, the federation message is to avoid APIs and implementation techniques that limit applications to a single underlying technology; as a result, any application developed to the Solaris federation architecture should port easily to other environments.

Glossary

API

Application Programming Interface. A set of source-level interfaces (usually procedure calls) used by application developers.

ASI

Application Specific Interface. A set of source-level interfaces (usually procedure calls) used by an application that define its internal structure. As applications evolve to an object oriented implementation, these interfaces may evolve to be interfaces to the collection of objects that implement the application.

Application

A collection of one or more objects that provide services to users.

ATLAS

See UI ATLAS.

CFS

Cache File System. A file system type that can be interposed over two existing file system types to improve filing service operation.

Composite Name

A concatenated sequence of names from more than one naming system. An application treats a composite name as simply a name, while a federation of naming services cooperate to resolve the composite name into a reference to the thing being named.

CORBA

Common Object Request Broker Architecture. This element of the OMG object architecture corresponds to the DOMF implementations being developed by HP and SunSoft.

DCE

See OSF DCE.

DES

The Data Encryption Standard defined by a Federal Information Processing Standard.

DFS

A generic term for Distributed File System. ONC+ NFS and DCE DFS are specific DFS types. Other DFS types include AFS (the Andrew File System, from CarnagieMellon University and Transarc) and RFS (the Remote File System, from USL).

DNS

See Internet DNS.

DOE

Distributed Objects Everywhere. The code name for a SunSoft project to make object oriented technology the pervasive model for distributed computing.

DOMF

Distributed Object Management Facility. The core functionality to implement objects; this functionality includes naming, location and communication. The DOMF corresponds to the CORBA element of the OMG object architecture.

ECMA

European Computer Manufacturers Association.

Federated Naming Service

An aggregation of autonomous naming systems cooperating through a standard interface and protocol to provide name resolution of composite names.

GSSAPI

Generic Security Service Application Programming Interface. An API for authentication intended to permit more flexible introduction of authentication engines into distributed systems.

HSFS

High Sierra File System; an industry standard file system for storing information on CD-ROM media.

IDL

see Interface Definition Language.

Inheritance

The ability to define a new interface based on one or more existing interfaces.

Interface

The set of operations an object provides.

Interface Compiler

A source code generator that accepts input in the form of an interface specification written in some interface specification language. The compiler then generates one or more stub libraries.

Interface Definition Language

The standard language for interface specification adopted by OMG.

Interface Inheritance

The definition of an interface in terms of one or more existing interface definitions.

Interface Specification Language

A language used to formally describe the syntax and limited semantics of programmatic interfaces.

Internet DNS

Internet Domain Name System.

ISO

International Organization for Standardization which developed the OSI standards.

ISV

Independent Software Vendor. A company that develops and sells software products.

Naming System

A set of contexts of the same object type that is connected (directly or indirectly) to one another by bindings.

NFS

Network File System, protocol version 2. A distributed filing protocol in wide use on a large variety of systems.

NIS

Network Information Service. Formerly known as Yellow Pages. (Yellow Pages is a registered trademark of British Telecom, Inc.) NIS is a directory service available with SVR4, and is part of ONC. NIS is commonly used to store passwords as well as file system and network configuration information.

NIS+

An enhanced version of NIS that provides improved update properties, better performance, and improved security.

NSS

Name Service Switch.

Object

An object provides a behavior defined by an interface that consists of a set of operations. An object hides the representation of its state and the implementation of its operations. An object is an instance of an object type that defines the object's interface.

OMG

The Object Management Group. A widely backed industry consortium working to standardize object management architecture. The DOE and DOMF projects in SunSoft are based on and contribute to the OMG effort.

ONC+

Open Network Computing architecture. ONC+ is based on ONC includes TI-RPC, NFS, and NIS+.

Operation

The syntax and semantics of an object invocation.

OSF

Open Software Foundation, Inc. An industry consortium developing software technologies.

OSF DCE

OSF's Distributed Computing Environment.

OSI

Open Systems Interconnection. A set of international standards, developed by The International Organization for Standardization (ISO), including the seven-layer reference model for data communication systems and the X.500 naming and directory service recommendations.

POSIX

The Institute of Electrical and Electronics Engineers (IEEE) standard for a portable operating systems environment. POSIX interfaces are based on the UNIX operating system.

Protocol Stack

A specific set of protocol choices for all layers of communication. (The stack includes application through the physical layers.)

RFS

Remote File System. A System V file system type that provides a distributed file service.

RPC

Remote procedure call. The mechanism for passing values and control across a network to a remote procedure.

Security Domain

A set of objects defining the scope of a set of security policies.

Security Policy

A set of rules for providing security services.

Server

As used in object-oriented development, a relation on objects. Object A is a server of object B if A provides services to B.

Service

Value-added functionality (such as filing, word processing, display management, and so on) that can be provided directly to users or to programs on the system or network.

STREAMS

A SVR4 mechanism for communication between layers of protocol stacks. A primary advantage of STREAMS over UCB sockets is a modular approach to the implementation and control of protocols.

Stub Library

A collection of procedures defining an API or ASI in terms of the communication mechanisms for invoking operations on an object (client side) and for accepting invocations (server side).

SVID

The System V Interface Definition. It serves as a reference for the definition of the service interfaces provided by all System V environments. The SVID facilitates the development of application source code that is portable across all System V implementations.

SVR4

UNIX System V Release 4.

Systems Integrator

A company whose primary product is an integrated computing system, including hardware, system software, and applications.

TIRPC

Transport Independent RPC. The RPC technology in ONC (available in System V and in a wide variety of other computing environments).

TLI

Transport Layer Interface. A System V API to transport-layer protocols permitting application programs to remain independent of any particular transport-layer protocol.

TP

Transaction Processing.

TP Monitor

A distributed system of objects comprising a transaction manager together with facilities for TP-oriented communications, transaction scheduling, administration, and naming.

Transaction

A sequence of operations on one or more objects. Transactions either execute to completion or have no effect on the states of any of the objects.

Transaction Manager

A subsystem of objects within a TP Monitor interacting with resource managers (for example, databases) to supply transaction semantics to sequences of distributed operations.

UCB

University of California, Berkeley.

UFS

UNIX File System. A file system type derived from Berkeley's fast file system. It is now the default file system type for System V Release 4.

UI ATLAS

UNIX International's model for computing systems architecture. The UI ATLAS distributed computing architecture is a subset of UI ATLAS.

USL

UNIX System Laboratories.

VFS

Virtual File System. See also vnode interface.

Vnode

virtual inode. A memory-resident data structure that represents an open file or directory.

Vnode Interface

An interface that permits multiple file system types to coexist in the same operating system. The interface specifies operations on a virtual file system (VFS) and on a virtual inode (vnode).

XPG

X/Open Portability Guide. X/Open publishes the XPG specifying interfaces to most standard system software services.

X/Open

X/Open is an international consortium of information systems providers and consumers dedicated to promoting application portability through open systems interfaces.

XTI

X/Open Transport Interface. A transport-independent API derived from TLI.

References

The following is a list of articles that provide detailed information on federated service issues:

UI-ATLAS Distributed Computing Architecture: A Technical Overview, September 13, 1991.

Distributed Computing: Ready Today, A Path to Tomorrow by Netwise, Inc. 1989. Discusses the strategies of major distributed computing vendors (for example, Sun, Novell, and Netwise).

Evolving the Vnode Interface by David Rosenthal, Sun Microsystems, June, 1990. Discusses the development of the vnode interface and its role in supporting federated file services.

Federated Naming in Distributed Systems by SunSoft Inc., draft December 7, 1992.

Administering NIS+ and DNS by SunSoft Inc., June 1992.

Generic Security Service Application Program Interface by John Linn, Digital Equipment Corporation, June 11, 1991.

"Transport Level Interface," *Network Interfaces Programmer's Guide* by SunSoft, June, 1992.

Project DOE Distributed Objects Everywhere, a white paper by SunSoft, 1991.

UNIX System V STREAMS Primer by AT&T, 1987.

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
FAX 415 969-9131

For U.S. Sales Office locations, call:
800 821-4643
In California:
800 821-4642

Australia: (02) 413 2666
Belgium: 32-2-759 5925
Canada: 416 477-6745
Finland: 358-0-502 27 00
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Italy: 039 60551
Japan: (03) 221-7021
Korea: 822-563-8700
Latin America: 415 688-9464
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
PRC: 861-831-5568
Singapore: 224 3388
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
UK: 0276 20444

Elsewhere in the world,
call Corporate Headquarters:
415 960-1300
Intercontinental Sales: 415 688-9000