Inferno: la Commedia Interattiva

Rob Pike
Sean Dorward
Phil Winterbottom
Dave Presotto
Dennis Ritchie
Howard Trickey

Computer Science Research Center
Bell Labs, Lucent Technologies

Inferno™ is an operating system for creating distributed services. It is a product of the Inferno Business Unit of Lucent Technologies and was developed by them and the Computing Science Research Center of Bell Labs, the R&D arm of Lucent Technologies.

Although it a new system designed specifically as a commercial product, it draws from many years of Bell Labs research in operating systems, languages, on-the-fly compilers, graphics, security, networking and portability.

Inferno is intended to be used in a variety of network environments, for example those supporting advanced telephones, hand-held devices, TV set-top boxes attached to cable systems, and inexpensive networked computers, but also in conjunction with traditional computing systems.

The most visible new environments involve cable television, direct satellite broadcast, the Internet, and other networks. As the entertainment, telecommunications, and computing industries converge and interconnect, a variety of public data networks are emerging, each potentially as useful and profitable as the telephone system. Unlike the telephone system, which started with standard terminals and signaling, these networks are developing in a world of diverse terminals, network hardware, and protocols. Only a well-designed, economical operating system can insulate the various providers of content and services from the equally varied transport and presentation platforms. Inferno is a network operating system for this new world.

Inferno's definitive strength lies in its portability and versatility across several dimensions:

- Portability across processors: it currently runs on Intel, Sparc, MIPS, ARM, HP-PA, and AMD 29K architectures and is readily portable to others.
- Portability across environments: it runs as a stand-alone operating system on small terminals, and also as a user application under Windows NT, Windows 95, Unix (Irix, Solaris, Linux, AIX, HP/UX) and Plan 9. In all of these environments, Inferno applications see an identical interface.
- Distributed design: the identical environment is established at the user's terminal and at the server, and each may import the resources of the other; aided by the communications facilities of the run-time system, applications may be split easily (and even dynamically) between client and server.
- Minimal hardware requirements: it runs useful applications stand-alone on machines with as little as 1 MB of memory, and does not require memory-mapping hardware.

- Portable applications: Inferno applications are written in the type-safe language Limbo™, whose binary representation is identical over all platforms.
- Dynamic adaptability: applications may, depending on the hardware or other resources available, load different program modules to perform a specific function. For example, a video player application might use any of several different decoder modules.

Underlying the design of Inferno is a model of the diversity of application areas it intends to stimulate. Many providers are interested in purveying media and services: telephone network service providers, WWW servers, cable companies, merchants, various information providers. There are many connection technologies: ordinary telephone modems, ISDN, ATM, the Internet, analog broadcast or cable TV, cable modems, digital video on demand, and other interactive TV systems. Finally, there are existing or potential hardware endpoints. Some are in consumers' homes: PCs, game consoles, newer set-top boxes. Some are inside the networks themselves: nodes for billing, network monitoring or provisioning. The higher ends of these spectra, epitomized by fully interactive TV with video on demand, may be fascinating, but have developed more slowly than expected. One reason is the cost of the set-top, especially its memory requirements. Portable terminals, because of weight and cost considerations, are similarly constrained.

Inferno is parsimonious enough in its resource requirements to support interesting applications on today's hardware, while being versatile enough to grow into the future. In particular, it enables developers to create applications that will work across a range of facilities. An example: an interactive shopping catalog that works in text mode over a POTS modem, shows still pictures (perhaps with audio) of the merchandise over ISDN, and includes video clips over digital cable.

Clearly not everyone who deploys an Inferno-based solution will want to span the whole range of possibilities, but the system architecture should be constrained only by the desired markets and the available interconnection and server technologies, not by the software.

Inferno interfaces

The role of the Inferno system is to *create* several standard interfaces for its applications:

- Applications use various resources internal to the system, such as a consistent virtual machine that runs the application programs, together with library modules that perform services as simple as string manipulation through more sophisticated graphics services for dealing with text, pictures, higher-level toolkits, and video.
- Applications exist in an external environment containing resources such as data files that can be read and manipulated, together with objects that are named and manipulated like files but are more active. Devices (for example a hand-held remote control, an MPEG decoder or a network interface) present themselves to the application as files.
- Standard protocols exist for communication within and between separate machines running Inferno, so that applications can cooperate.

At the same time, Inferno *uses* interfaces supplied by an existing environment, either bare hardware or standard operating systems and protocols.

Most typically, an Inferno-based service would consist of many relatively cheap terminals running Inferno as a native system, and a smaller number of large machines running Inferno as a hosted system. On these server machines Inferno might interface to databases, transaction systems, existing OA&M facilities, and other resources provided under the native operating system. The Inferno applications themselves would run either on the client or server machines, or both.

External Environment of Inferno Applications

The purpose of most Inferno applications is to present information or media to the user; thus applications must locate the information sources in the network and construct a local representation of them. The information flow is not one-way: the user's terminal (whether a network computer, TV set-top, PC, or videophone) is also an information source and its devices represent resources to applications. Inferno draws heavily on the design of the Plan 9 operating system in the way it presents resources to these applications.

The design has three principles. First, all resources are named and accessed like files in a forest of hierarchical file systems. Second, the disjoint resource hierarchies provided by different services are joined together into a single private hierarchical *name space.* Third, a communication protocol, called *Styx*, is applied uniformly to access these resources, whether local or remote.

In practice, most applications see a fixed set of files organized as a directory tree. Some of the files contain ordinary data, but others represent more active resources. Devices are represented as files, and device drivers (such as a modem, an MPEG decoder, a network interface, or the TV screen) attached to a particular hardware box present themselves as small directories. These directories typically containing two files, `data` and `ctl`, which respectively perform actual device I/O and control and status operations. System services also live behind file names. For example, an Internet domain name server might be attached to an agreed-upon name (say `/net/dns`); after writing to this file a string representing a symbolic Internet domain name, a subsequent read from the file would return the corresponding numeric Internet address.

The glue that connects the separate parts of the resource name space together is the Styx protocol. Within an instance of Inferno, all the device drivers and other internal resources respond to the procedural version of Styx. The Inferno kernel implements a *mount driver* that transforms file system operations into remote procedure calls for transport over a network. On the other side of the connection, a server unwraps the Styx messages and implements them using resources local to it. Thus, it is possible to import parts of the name space (and thus resources) from other machines.

To extend the example above, it is unlikely that a set-top box would store the code needed for an Internet domain name-server within itself. Instead, an Internet browser would import the `/net/dns` resource into its own name space from a server machine across a network.

The Styx protocol lies above and is independent of the communications transport layer; it is readily carried over TCP/IP, PPP, ATM or various modem transport protocols.

Internal Environment of Inferno Applications

Inferno applications are written in a new language called Limbo, which was designed specifically for the Inferno environment. Its syntax is influenced by C and Pascal, and it supports the standard data types common to them, together with several higher-level data types such as lists, tuples, strings, dynamic arrays, and simple abstract data types.

In addition, Limbo supplies several advanced constructs carefully integrated into the Inferno virtual machine. In particular, a communication mechanism called a *channel* is used to connect different Limbo tasks on the same machine or across the network. A channel transports typed data in a machine-independent fashion, so that complex data structures (including channels themselves) may be passed between Limbo tasks or attached to files in the name space for language-level communication between machines.

Multi-tasking is supported directly by the Limbo language: independently scheduled threads of control may be spawned, and an `alt` statement is used to coordinate the channel communication between tasks (that is, `alt` is used to select one of several channels that are ready to communicate). By building channels and tasks into the language and its virtual machine, Inferno encourages a communication style that is easy to use and safe.

Limbo programs are built of *modules*, which are self-contained units with a well-defined interface containing functions (methods), abstract data types, and constants defined by the module and visible outside it. Modules are accessed dynamically; that is, when one module wishes to make use of another, it dynamically executes a `load` statement naming the desired module, and uses a returned handle to access the new module. When the module is no longer in use, its storage and code will be released. The flexibility of the modular structure contributes to the smallness of typical Inferno applications, and also to their adaptability. For example, in the shopping catalog described above, the application's main module checks dynamically for the existence of the video resource. If it is unavailable, the video-decoder module is never loaded.

Limbo is fully type-checked at compile- and run-time; for example, pointers, besides being more restricted than in C, are checked before being dereferenced, and the type-consistency of a dynamically loaded module is checked when it is loaded. Limbo programs run safely on a machine without memory-protection hardware. Moreover, all Limbo data and program objects are subject to a garbage collector, built deeply into the Limbo run-time system. All system data objects are tracked by the virtual machine and freed as soon as they become unused. For example, if an application task that creates a graphics window and then terminates, the window automatically disappears the instant the last reference to it has gone away.

Limbo programs are compiled into byte-codes representing instructions for a virtual machine called Dis™. The architecture of the arithmetic part of Dis is a simple 3-address machine, supplemented with a few specialized operations for handling some of the higher-level data types like arrays and strings. Garbage collection is handled below the level of the machine language; the scheduling of tasks is similarly hidden. When loaded into memory for execution, the byte-codes are expanded into a format more efficient for execution; there is also an optional on-the-fly compiler that turns a Dis instruction stream into native machine instructions for the appropriate real hardware. This can be done efficiently because Dis instructions match well with the instruction-set architecture of today's machines. The resulting code executes at a speed approaching that of compiled C.

Underlying Dis is the Inferno kernel, which contains the interpreter and on-the-fly compiler as well as memory management, scheduling, device drivers, protocol stacks, and the like. The kernel also contains the core of the file system (the name evaluator and the code that turns file system operations into remote procedure calls over communications links) as well as the small file systems implemented internally.

Finally, the Inferno virtual machine implements several standard modules internally. These include `Sys`, which provides system calls and a small library of useful routines (e.g. creation of network connections, string manipulations). Module `Draw` is a basic graphics library that handles raster graphics, fonts, and windows. Module `Prefab` builds on `Draw` to provide structured complexes containing images and text inside of windows; these elements may be scrolled, selected, and changed by the methods of `Prefab`. Module `Tk` is an all-new implementation of the Tk graphics toolkit, with a Limbo interface. A `Math` module encapsulates the procedures for numerical programming.

The Environment of the Inferno System

Inferno creates a standard environment for applications. Identical application programs can run under any instance of this environment, even in distributed fashion, and see the same resources. Depending on the environment in which Inferno itself is implemented, there are several versions of the Inferno kernel, Dis/Limbo interpreter, and device driver set.

When running as the native operating system, the kernel includes all the low-level glue (interrupt handlers, graphics and other device drivers) needed to implement the abstractions presented to applications. For a hosted system, for example under Unix, Windows NT or Windows 95, Inferno runs as a set of ordinary processes. Instead of mapping its device-control functionality to real hardware, it adapts to the resources provided by the operating system under which it runs. For example, under Unix, the graphics library might be implemented using the X window system and the networking using the socket interface; under Windows, it uses the native Windows graphics and Winsock calls.

Inferno is, to the extent possible, written in standard C and most of its components are independent of the many operating systems that can host it.

Security

Inferno provides security of communication, resource control, and system integrity.

Each external communication channel may be transmitted in the clear, accompanied by message digests to prevent corruption, or encrypted to prevent corruption and interception. Once communication is set up, the encryption is transparent to the application.

Key exchange is provided through standard public-key mechanisms (e.g. a modified station-to-station protocol); after key exchange, message digesting and line encryption likewise use standard symmetric mechanisms (MD5, SHA, RC4, DES).

Inferno is secure against erroneous or malicious applications, and encourages safe collaboration between mutually suspicious service providers and clients. The resources available to applications appear exclusively in the name space of the application, and standard protection modes are available. This applies to data, to communication resources, and to the executable modules that constitute the applications. Security-sensitive resources of the system are accessible only by calling the modules that provide them; in particular, adding new files and servers to the name space is controlled and is an authenticated operation. For example, if the network resources are removed from an application's name space, then it is impossible for it to establish new network connections.

Object modules may be signed by trusted authorities who guarantee their validity and behavior, and these signatures may be checked by the system the modules are accessed.

More information

For more information about the system, including a downloadable version complete with Limbo source for all the applications, visit the Inferno web site `http://inferno.lucent.com`.