

**METHOD AND APPARATUS FOR  
DETERMINING AT EXECUTION  
COMPATIBILITY AMONG CLIENT AND  
PROVIDER COMPONENTS WHERE  
PROVIDER VERSION LINKED WITH  
CLIENT MAY DIFFER FROM PROVIDER  
VERSION AVAILABLE AT EXECUTION**

This is a continuation of application Ser. No. 08/058,345 filed May 5, 1993 now abandoned.

**LIMITED COPYRIGHT WAIVER**

A portion of the disclosure of this patent document contains material to which the claim of copyright protection is made. The copyright owner has no objection to the facsimile reproduction by any person of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office file or records, but reserves all other rights whatsoever.

**FIELD OF THE INVENTION**

This invention relates generally to compatibility between various components of a computer system and, more specifically, to checking whether interoperating components are compatible within a specified compatibility range.

**BACKGROUND OF THE INVENTION**

In a computer system, compatibility between various components, both hardware and software, of the system may be important. A new component or an upgrade of an existing component may be incompatible with other components in a system, rendering the entire system inaccurate or inoperative. Thus, it is important to have a mechanism for verifying compatibility between components of a system.

Compatibility plays an important role in a client-provider relationship wherein a provider supplies services to a client because it is important that the provider be able to satisfy expectations of the client. A client-provider relationship may exist between two computer software programs, two computer hardware components or between a software program and a hardware component. An example of a client-provider relationship is a relationship between a shared library (provider), that is a collection of routines which can be accessed by other computer programs, and a computer program (client) which accesses the routines in the library.

Typically, as computer programs are modified, upgraded or otherwise improved, new versions of the computer programs are released. To differentiate between different versions of a computer program, a version number is typically assigned to each release of the computer program. Usually, version numbers are assigned such that a newer version of a computer program has a higher version number than an earlier version of that computer program. For example, if a particular version of a computer program has a version number of 2, then a subsequent version of that computer program may have a version number greater than 2.

Computer programs are typically written originally in source code in a computer language such as C or Pascal, or in an assembly language. To prepare the program for execution on a computer system, a compiler (or assembler) converts one or more source code modules into an object code file. A compiler (or assembler) is specific to the language used and the computer system on which it will be executed. A linker routine, which is either a separate pro-

gram or is part of the compiler, combines the object code files into a single output file, known as an "executable" object code file. One or more executables are then loaded together into memory by a loader program, and control is transferred to a start address to initiate program execution.

Typically, in a client-provider relationship between two software programs, it is important for compatibility to exist between the version of a provider such as a shared library linked to a client and the version of the provider (shared library) used during execution of the client. During linking in a client/provider relationship, imports (unresolved external symbols in the client) are resolved to exports from the provider (symbols in the provider that are visible to the client). At link time, the provider supplies definitions of symbols (the API) but not the actual implementation of routines and variables (the code). Thus, the version of the provider used at link time is called a "definition version".

When a client is executed, the imports in the client are connected to the associated exports in the provider. The connection could be in hardware such as a wire between the two, or in software such as an operating system, a code fragment manager, or other shared library or code manager. At runtime, the provider supplies actual implementation of routines and variables, i.e. code. Since the API is supplied at link time and the code is supplied at runtime, it is important that the definitions supplied by a provider at link time are compatible with the implementation of the provider used at runtime.

In the VMS operating system by Digital Equipment Corp., Inc., typically a version of a provider is designed to be compatible with previous versions, i.e. a version of a provider is backwards compatible. VMS is a trademark of Digital Equipment Corporation. Thus, a client can be executed using a version of the provider which is newer than the version with which it was built. However, a version of the provider which is older than the version used to build the client may not support features available in newer versions of the provider. Therefore, a client can not be executed using a version of a provider which is older than the version used to build the client, because the older provider may not be compatible with the newer version.

In VMS, as shown in "VAX/VMS Internals and Data Structures", version 4.4, Lawrence J. Kenah, Ruth E. Goldberg, Simon F. Bate (Digital Press: 1988), section 21.1.1.2, a provider typically has a revision number comprising a major number component and a minor component, usually denoted as "major.minor" or "major/minor". VAX is a trademark of Digital Equipment Corporation. When a new revision of a provider contains substantial changes from a prior instance of the provider, then the major number is incremented. However, if the changes are only minor changes, then usually only the minor number is incremented. At link time, the revision number of the provider supplying the definitions is stored in the executable object code. At execution time, the major number of the stored revision number in the client is compared to the major number in the revision code of the provider being accessed to implement the client. If the major number of the revision of the implementation provider is less than the major number of the revision number stored in the client's executable object code, then the client and provider are considered incompatible, regardless of whether the two are in fact incompatible.

For example, if the client is linked with a provider having a revision number of 7.1, then it would be considered incompatible with a provider having a major number less than 7 such as 6.5, 5.0, etc. . . . , but it would be considered