PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 9/44

A1

(11) International Publication Number: WO 94/23360

(43) International Publication Date: 13 October 1994 (13.10.94)

(21) International Application Number: PCT/US94/00053

(22) International Filing Date: 3 January 1994 (03.01.94)

(30) Priority Data:
08/042,959 5 April 1993 (05.04.93) US

(71) Applicant: TALIGENT, INC. [US/US]; 10201 N. De Anza Boulevard, Cupertino, CA 95014 (US).

(72) Inventors: HENINGER, Andrew, G.; 1611 Vorte Via, Los Altos, CA 94024 (US). NAKANO, Russell, T.; 1326 Alridge Drive, Sunnyvale, CA 94087 (US).

(74) Agent: STEPHENS, Keith; Taligent, Inc., 10201 N. De Anza Boulevard, Cupertino, CA 95014 (US).

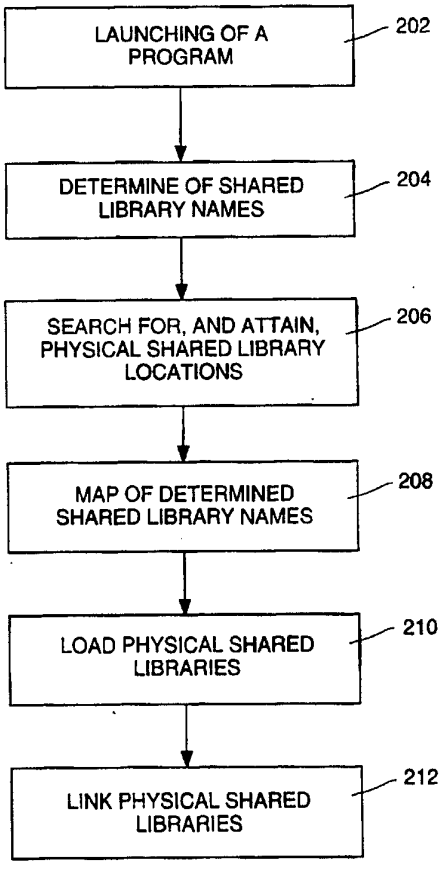
(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: SHARED LIBRARY LOCATING SYSTEM

(57) Abstract

A method and apparatus is disclosed for locating and loading shared libraries linked with a given shared library that has been launched by a program. The present invention operates by initially determining the names of the shared library names that are necessary to a program that has been launched. Thereafter, search objects are employed to search for and attain the physical shared library locations that correspond to the determined names. A collection of search objects can be placed within a team and employed in a selected sequential order. Upon the attainment of an appropriate location corresponding to each of the shared library names, each of the shared library names are mapped to its corresponding physical location. Thereafter, the contents of each physical shared library location can be loaded for purposes of a program.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

Shared library locating system

CROSS-REFERENCE TO RELATED PATENT APPLICATIONS

This application is related to the following U.S. patent applications: (i) the application entitled "Object Oriented Framework System," by Debra L. Orton, David B. Goldsmith, Christopher P. Moeller, and Andrew G. Heninger, filed December 23, 1992, and assigned to Taligent, Inc., the disclosure of which is hereby incorporated by reference; and (ii) the application entitled "Object Oriented Operating System," by Robert Dickinson, filed January 22, 1993, and assigned to Taligent, Inc., the disclosure of which is hereby incorporated by reference.

FIELD OF THE INVENTION

This invention generally relates to the loading of shared libraries and more particularly, to the mapping of shared library names with physical shared library locations by the employment of search objects.

BACKGROUND OF THE INVENTION

A library represents a collection of data and code stored at a certain location. A library maintains older data items as well as receives new data items. Thus, the number of data items can constantly increase. In addition, existing data items are dynamic and subject to change. Multiple versions of a library may also exist in a computer system.

A library may also be shared. That is, a library may be used concurrently by several applications or processes. Such a library is referred to as a "shared library".

When an application or program is launched, it is necessary to "load" all shared libraries that are necessary to implement the program. A program generally refers to the name of a shared library. Consequently, the shared library referred to within the program, as well as all other shared libraries it depends upon, must be located before loading can occur.

Current computer systems are unable to effectively ascertain the locations of requisite shared libraries from the name referred to by a program. This follows since current computer systems are unable to undertake complex searches for shared library locations. Furthermore, current computer systems are unable to distinguish between versions and copies of shared libraries. Still further, current

computer systems are unable to appropriately compensate for the compatibility attributes of different shared libraries. Thus, the prior art that the applicant is aware of provides a system that is insufficiently complex and effective for loading shared libraries.

SUMMARY OF THE INVENTION

The present invention overcomes the aforementioned deficiencies of the prior art by providing a method and apparatus that effectively searches for and locates physical shared library locations from shared library names.

The present invention operates by initially determining the names of the shared libraries that are necessary to a program that has been launched. Thereafter, search objects are employed to search for and locate a physical location corresponding to each of the determined names. Multiple search objects can also be formulated within a team. A team refers to an address space and a plurality of threads of execution executing within the address space. This allows for physical locations to be located by employing a collection of search objects in a selected sequential order. For instance, search objects can be consecutively aemployed in the order in which they were added to the team.

Once the search objects attains an appropriate physical location corresponding to each of the shared library names, each of the shared library names is mapped to its corresponding physical location. Thereafter, the contents of the attained physical shared library locations can be loaded for purposes of the program.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other aspects and advantages of the present invention will be better understood from the following detailed description of the preferred embodiment of the invention with reference to the accompanying drawings, in which:

Figure 1 is a block diagram of a computer system in accordance with the present invention;

Figure 2 illustrates a flow chart depicting the logic flow in accordance with the present invention;

Figure 3 illustrates a class inheritance hierarchy for search objects in accordance with the present invention;

Figure 4 illustrates a flow chart depicting the search for a named shared library, as followed by a search object, in accordance with the present invention; and

Figure 5 illustrates an example of shared libraries having varying compatibility numbers, pursuant to an example, in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE PRESENT INVENTION

COMPUTER SYSTEM

A representative hardware environment is depicted in Figure 1, which illustrates a suitable hardware configuration of a workstation **40** in accordance with the present invention. The workstation **40** has a central processing unit **10**, such as a conventional microprocessor, and a number of other units interconnected via a system bus **12**. The illustrated workstation **40** shown in Figure 1 includes a Random Access Memory **14** (RAM), a Read Only Memory **16** (ROM), an I/O adapter **18** for connecting peripheral devices such as disk units to the bus **12**, a user interface adapter **22** for connecting a keyboard **24**, a mouse **26**, a speaker **28**, a microphone **32**, and/or other user interface devices such as a touch screen device (not shown) to the bus **12**. The workstation **40** may also have a communications adapter **34** for connecting the workstation **40** to a data processing network **30** and a display adapter **36** connecting the bus **12** to a display device **38**.

The present invention preferably operates within an object oriented operating system as described by U.S. patent applications entitled "Object Oriented Framework System," filed December 23, 1992, and "Object Oriented Operating System," filed January 22, 1993. Both such patent applications having been assigned to Taligent, Inc.

LOGIC OPERATION OF THE PRESENT INVENTION

The present invention comprises a framework, to be used in conjunction with a computer system at run time of a program, for loading and linking shared libraries with one another. Such framework provides a manner by which a shared

library name, as referred to by a program, is linked with the shared libraries from which it depends upon the launching of the program.

Every shared library is identified by a specified name and is located within a designated name space. A shared library is dependent upon, or necessarily linked with, any number of other shared libraries. A shared library load module, corresponding to a particular shared library, stores: (i) the specified name of a particular shared library; and (ii) the specified names of shared libraries from which the particular shared library depends.

Whenever a particular shared library is loaded pursuant to a program, all shared libraries which the particular shared library depends upon must also be loaded. Accordingly, a shared library load module is resorted to in order to ascertain the names of shared libraries to be loaded pursuant to a program. Since the shared library name space is global, each shared library should be given an uncommon name in order to insure that the appropriate shared libraries are linked together. The present invention operates upon the premise that each shared library has such a unique name.

For a program to execute upon launching, a shared library must be linked with other shared libraries. This linking process takes place once the appropriate shared libraries are loaded.

A mechanism, referred to as a loader, initially determines the names of the shared libraries that must be loaded. The loader must thereafter be advised of the physical shared libraries which correspond to the determined shared library names. Locations of physical shared libraries typically include a computer disk, RAM ("random access memory") and ROM ("read only memory"). The loader does not, however, determine where the libraries are located. Instead, the loader uses search objects to determine the location of shared libraries. Search objects map library names to physical shared library locations. The resulting mapping information then enables the loader to load the appropriate shared libraries.

Referring to Figure 2, a flow chart describing the steps of linking one shared library with another is illustrated. The launching of a program typically requires shared libraries to be loaded and linked with one another. This is indicated by box **202**. Thereafter, a determination is made of the names of shared libraries that are necessary to the launched program. This is indicated by box **204**. Shared library

names that are necessary to a program include those names referred to by a program as well as shared libraries from which the launched program or referenced shared library are dependent upon. Such names are ascertained from the aforementioned shared library load module.

Once the names of the shared libraries are ascertained, search objects are employed to locate the corresponding physical shared library locations that correspond to the names. This is indicated by box **206**. Class TLibrarySearcher, as further described below, is a derived class that defines the interface for employed search objects. Given the ascertained names of the shared libraries, a search object attains the physical shared libraries by following the search instructions of the search object.

Attainment of the corresponding physical shared library locations results in mapping as indicated by box **208**. Mapping involves designating the shared library names with the attained physical shared library locations. Thereafter, the attained physical shared libraries are loaded and linked with one another. This is indicated by reference numerals **210** and **212**, respectively.

SEARCH OBJECTS

Object-orientation is a system technique that consists of a number of objects. Each object, which typically corresponds to a real life entity, offers a number of operations and a state which remembers the state of those operations. For instance, a person object would possess information corresponding to a person such as age, address, telephone number, occupation and so on. Operations are then created to access or affect this information. In addition, operations can be created to perform behavior that does not affect any internal information such as walking.

A class is a definition, a template or a mold to enable the creation of new objects. A class is thus a description of the common characteristics of several objects. The objects comprising a certain class have this template in common. The characteristics of a class may be inherited into a further class referred to as a subclass. A subclass thus inherits the information structure and behavior of a class. Further information and behavior may then be added to the subclass.

Classes lying below a class in the inheritance hierarchy are called descendants of the class. Classes lying above are called ancestors. The usage of the terms descendants and subclasses is thus interchangeable.

Referring to Figure 3, a hierarchy of search object classes is illustrated. Class MCollectible **300** represents the base class of all search objects. That is, class MCollectible **300** is the class from which all other classes descend from. The two primary descendants of class MCollectible **300** are class TLibrarySearcher **302** and class TTeamHandle **308**. Class TLibrarySearcher **302** has two further descendants, class TStandardLibrarySearcher and class TWorkspaceSearcher, as indicated by reference numerals **304** and **306**, respectively.

Class TStandardLibrarySearcher **304** defines a default search object which searches for physical shared libraries in a single directory. Since more than one directory is usually necessary to be searched, class TStandardLibrarySearcher **304** is typically incapable of locating all physical shared libraries in a system. Further, class TStandardLibrarySearcher **304** objects do not consider compatibility when faced with multiple copies of a library.

Accordingly, class TWorkspaceSearcher **306** is provided to locate physical shared libraries in lieu of a search object created pursuant to class TStandardLibrarySearcher **304**. Class TWorkspaceSearcher **306** permits search objects to be formed from a workstation. Search objects, defined by class TWorkspaceSearcher **306**, are capable of searching in more than one directory and are also capable of considering compatibility when necessary.

Objects derived from class TLibrarySearcher **302** can implement an arbitrarily complex search. For instance, such objects can: (i) consult file attributes; (ii) maintain a dictionary or database of libraries; and, (iii) interact with library installation procedures. A subclass of TLibrarySearcher **302,** namely class TStandardLibrarySearcher **304** and class TWorkspaceSearcher **306,** may also be created so as to check with the system if some version of a shared library is already in use.

Referring to Figure 4, a flow chart depicting the logic flow of a search for a shared library is illustrated. Initially, a search object is presented with the name of a shared library. From this named shared library that is presented, a search for the location of the shared library is undertaken. This is indicated by reference numeral **400**.

A search for the named library will reveal either: (i) multiple copies of the shared libraries; (ii) no location that corresponds to the shared library name; or (iii) one copy of the shared library. These outcomes are indicated by boxes having reference numerals **402, 410,** and **420,** respectively.

If a shared library is not found, the search object will return "NIL" as indicated by box **412.** If only one copy of the shared library is found, that shared library will be returned to the loader as indicated by box **422.**

If multiple copies of the shared library are found, however, further analysis is necessary before a copy can be returned. Such further analysis originally commences with a determination of whether the compatibility numbers of the shared library copies can be ascertained. This is indicated by box **404.** If the compatibility numbers of the shared library copies can be ascertained, the shared library copy with the highest compatibility number will then be returned. This is indicated by box **406.**

If the compatibility number of the shared library copies cannot be ascertained, a further method of selection is necessary. Such further method of selection can include a determination of whether any copies of the shared library are currently loaded elsewhere. Ultimately, the further method should return a selected copy of the shared library as indicated by box **416.**

Returned copies of shared libraries, as denoted by boxes **406, 416,** and **422,** are mapped with the shared library name. This is indicated by box **430.** Thereafter, the mapped information is disclosed to the loader.

Compatibility is a file system property of shared libraries which are files. Compatibility is measured by a compatibility number wherein a copy of a shared library with the highest compatibility number is likely to be compatible with other dependent shared libraries. Accordingly, a search object returns the copy of a shared library with the highest compatibility number. A search object is always able to do so when a shared library is a file since files always permit access to compatibility numbers. Shared libraries that are not files, however, may not allow access to the compatibility number of a copy. In such a case, the search object will not be able to ascertain the copy with the highest degree of compatibility.

Shared libraries are consecutively loaded where two conditions are met. First, a search object returns the copy with the highest compatibility number. Second, if among the locations where a search object has searched, a consistent set of shared libraries exist that will load.

Referring to Figure 5, an illustrative example of three libraries with different compatibility numbers is illustrated. Assume three libraries named A, B and C are being loaded. Library A, with compatibility number 1 ("library (A,1)" as indicated by reference numeral **502**), depends on library B with compatibility number 2 ("library (B,2)" as indicated by reference numeral **504**) and library C with compatibility number 2 ("library (C,2)" as indicated by reference numeral **506**). Library B with a compatibility number of 3 ("library (B,3)" as indicated by reference numeral **510**) depends on library C with compatibility number 3 ("library (C,3)" as indicated by reference numeral **508**). Library (B,2) **504** depends on library (C,2) **506**. Library (C,2) **506** and library (C,3) **508** do not, however, depend on any other libraries.

When the loader attempts to load library (A,1) **502**, it will also have to load a copy of the libraries which library (A,1) **502** is dependent upon. Loaded copies must possess a compatibility number that is equal to or greater than the compatibility number of the dependent shared library. Accordingly, a loader can load: (i) library (B,2) **504** or library (B,3) **510**; and (ii) library (C,2) **506** and library (C,3) **508**. Library (B,3) **510** must, however, be loaded with library (C,3) **508**. Library (C,2) **506** will not suffice to satisfy the dependency of library (B,3) **510**.

A search object, or plurality of search objects, will locate library (A,1) **502**, library (B,2) **504**, library (B,3) **510**, library (C,2) **506**, and library (C,3) **508**. If a search object returns the copy with the greatest compatibility number, it will return library (A,1) **502**, library (B,3) **510**, and library (C,3) **508**. These three libraries will then load since all dependencies are fulfilled.

If a search object did not return the aforementioned three libraries, loading may not have been possible. For instance, a search object could return library (A,1) **502**, library (B,3) **510**, and library (C,2) **506**. In such an instance, the libraries would not load because the dependency of library (B,3) **510** upon library (C,3) **508** is not satisfied. In order to avoid this situation, a search object must return the copy of a library with the highest compatibility number. Of course, this can only be accomplished if the search object can access the compatibility number of a shared library.

TEAM ATTRIBUTES OF SEARCH OBJECTS

Search objects are placed within a team. Each team has a list of one or more search objects which the loader then consults when loading a shared library associated with the team. There is no limit to the number of search objects that can be contained within a team. When a new team is created, a collection of search objects may be passed into a constructor object, as formed within class TTeamHandle (as disclosed within Figure 3 by reference numeral **308**). The collection of search objects is then added to the new team.

Two types of search objects, derived from class TLibrarySearcher **302**, exist within a team. First, search objects are added to all teams that are created. These are referred to as "system search objects". Second, search objects can be created that are only added to specified teams. These are referred to as team specific objects. When a team is accessed, both the team specific search objects and the system search objects are streamed into the address space of the team. Hence, TLibrarySearcher **302** subclasses TStandardLibrarySearcher **304** and TWorkspaceSearcher **306**, must have streaming operators.

The loader employs the collection of search objects within a team to map library names to library locations or modules. Thereafter, the shared libraries are placed within the address space of the team. The loader preferably employs the search objects in "LIFO order" ("last in, first out"). That is, the search objects are employed in the order with which they were added to the team. Accordingly, the most recently added search object is first employed. Search objects are consecutively employed in LIFO order until a shared library is located. If none of the search objects can locate the shared library, the loader raises an exception since it cannot proceed. In response to the exception, search objects may be added to a team after the team is accessed. However, subsequently added search objects can only locate shared libraries that will be dynamically loaded into the address space of the team after the addition of search objects.

Preferably, the loader employs search objects in LIFO order since it is assumed that the more recently added search objects are intended to override existing search objects. Thus, the collection of team search objects are always employed before any default search object as defined from class TStandardLibrarySearcher **304**. For example, there may be two versions of shared library X in the system. A default search object will map library X to a first version of library X. A search object

defined by class TWorkspaceSearcher **306** would, however, map to a second version of library X. If the workspace defined search object is employed before the default search object, library X will map to the second version of library X. Accordingly, the correct version of library X will be loaded into the address space of the team since the workspace defined search object is consulted before the system search object.

Alternatively, the same effect is achieved by specifying a temporary team search object. A temporary search object can be specified in a call to "TLibrary::LoadLibrary for library X." The team thus uses the temporary search object for the duration of the call so as to load the proper version of library X. Temporary search objects are especially useful after a team is launched. This is so since temporary search objects do not add to the collection of team search objects. In contrast, the addition of a search object to a team adversely affects the usage of the team with respect to other shared libraries. Thus, a temporary library search object should be utilized when dynamically loading libraries into a running team.

A system search object, which is associated with all teams, is employed by a loader only after all team specific search objects are employed. Thus, search objects are employed in the following order: (i) team specific search objects; (ii) system search objects; and (iii) default search objects. Furthermore, since the present invention employs system objects in LIFO order, an added system search objects only affects the loading of those shared libraries that are created after it was added to the system.

A search object may be added for application with respect to either all teams on a system or a particular team. If a system search object is added for application with respect to all teams (by invoking "TLibrarySearcher::AddSystemSearcher"), the system object will be applied to all subsequently created teams.
If, however, a search object is added for application with respect to a team (by the invocation of "TTeamHandle::AddLibrarySearcher"), the newly created search object will be applied only to library searchs by that team.

Preferably, search objects are not permitted to be removed once added to a team. This being so whether the added search object is a team specified search object or a system search object. This restriction follows since a search object is a global resource that is shared by all tasks of a team. Permitting removal of a team search object with respect to one task would adversely affect another task that is dependent upon the search object. For instance, suppose two independent tasks ("task 1" and

"task 2") on the same team have the same added search object. In the event that the search object is removed from task 1, task 2 will still assume that the search object is in place. Thus, both tasks should retain an identical list of search objects. Preferably, temporary search objects should be utilized when a search object need only be used with respect to a certain task so as to not require the search object to be added to a team.

The present invention also undertakes an analysis to determine if a search object, that is sought to be added to a team, already exists within the team. If a search object is added to a team where a duplicate of the search objects already resides, the addition of the second copy of the search object will not take place. The duplicate search object already residing within the team is, however, moved to the end of the listing of team library search objects. This follows since search objects at the end of a team's list of search objects are first employed by the team.

The present invention analyzes the search object type to discover duplicate search objects. For instance, the C++ programming language undertakes a comparison through the exercise of an "Is Equal" method. Hence, all TLibrarySearcher **302** subclasses must override the IsEqual method or an equivalent thereof.

A subclass of TLibrarySearcher **302** can be contained within a shared library. However, in order for a loader to find a given shared library, the given library must not contain the TLibrarySearcher **302** subclass necessary to find it. Thus, a TLibrarySearcher **302** subclass must be in a location external to a shared library which must be searched by the TLibrarySearcher **302** subclass. In other words, a shared library which contains a TLibrarySearcher **302** subclass must be in a location place where it can be found without relying on the TLibrarySearcher **302** subclass it contains to locate it.

One way to insure that the loader can find a given shared library, that contains a TLibrarySearcher **302** subclass, is to put the given shared library where a default system search object can locate it. Class TStandardLibrarySearcher **304**, which defines the default search object and is associated with every team, can always locate a given shared library. A newly created team can thus rely only on a default system search object until workspace search object, as defined by class TWorkspaceSearcher **306**, is added to the team. Whenever a search object is added to a team, the team must load the shared library that contains the search object.

This is accomplished by only using system library search objects and team specific search objects, if any, that already have been added to the team.

Preferably, only one copy of a shared library is loaded into the address space of a team. If the loader thus encounters a given shared library more than once when dealing with a single team, it will assume that the currently loaded shared library is correct. Accordingly, the loader will not even employ a search object to locate the duplicate library. If the collection of search objects of a team are modified and "LoadLibrary" is invoked to load a library that is already in the address space of the team, the modified team would return a different version of the library. Thus, the invocation of LoadLibrary will not have the desired effect since the old version of the library will remain in the address space of the team.

Different teams resident on the same workstation may use different copies of the same shared library. Teams which use different copies of the same shared library may not, however, share obejcts of another team by means of shared memory even though the object class is defined in the shared library. Rather, a given team is limited to the use of those objects associated with the given team. Although teams using different versions of a shared library cannot share objects with one another, teams are permitted to stream objects whose classes are defined in the shared library to each other.

An embodiment of the present invention, developed by the inventor, is provided in the appendix to this application. This is done in order to clarify the detailed logic of the present invention. Critical elements of each portion of the embodiment are also further explained.

APPENDIX: PREFERRED EMBODIMENT OF THE PRESENT INVENTION

SOURCE CODE:

```
//Class TLibrarySearcher derived from base class MCollectible
class TLibrarySearcher : public MCollectible
        {
        ....
 public                                  _TLibrarySearcher();
        virtual TMappableDataContainer *CreateContainerAndFindLibrary(
                                                constTText&LibraryName,
                                                constCompatibilityNumber
                                            desiredRelease)
                                                = 0;
        virtual Boolean                 IsEqual(const MCollectible *) const = 0;
        //the following public methods are for use only by the workspace.
```

```
        static void                    AddSystemLibrarySearcher(TLibrarySearcher &);
        static TIterator   *CreateSystemLibrarySearcherIterator();

    protected:
                                       TLibrarySearcher();
                                       TLibrarySearcher(const TLibrary Searcher&);
        TLibrarySearcher&  operator= (const TLibrary Searcher &);
        virtual TStream&   operator>>=(TStream& toWhere) const;
        virtual TStream&    operator<<=(TStream& fromWhere);
        }


//Subclass library searcher used by the run time during system boot.
//Not intended for subclassing.
classTStandardLibrarySearcher : public TLibrarySearcher
        {
        ....
    public:
        virtual                        _TStandardLibrarySearcher();
                                       TStandardLibrarySearcher();
                                       TStandardLibrarySearcher(const
                                       TStandardLibrarySearcher &);
        virtual TMappableDataContainer *CreateContainerAndFindLibrary(const
                                         TText &libraryName, const
                                         CompatabilityNumber desiredRelease);

        virtual Boolean                IsEqual(const TLibrary Searcher *) const;
        TStandardLibrarySearcher & operator=(const TStandardLibrarySearcher    &);
        virtual TStream &  operator>>=(TStream& toWhere) const;
        virtual TStream &  operator<<=(TStream& fromWhere);
        }


//Excerpt of TTeamHandle.  These are the methods of TTeamHandle
//that use TLibrary Searcher
class TTeamHandle : public MCollectible
        {
        ....
                                       TTeamHandle(const TTaskProgram
&whatToRun,
                                         const TOrderedCollection & library
                                       Searchers);
        virtual void       AddLibrarySearcher(TLibrarySearcher &);
        virtual TIterator  *CreateLibrarySearcherIterator();
        ....
        }
```

COMMENTS ON SOURCE CODE:

//Class TLibrarySearcher derived from base class MCollectible

~TLibrarySearcher(): Destructor.

TMappableDataContainer*CreateContainerAndFindLibrary(constTText&libraryName, constCompatibilityNumberdesiredRelease): Search for the library with the specified name; return a TMappableDataContainer for that library. The caller owns the storage for the TMappableDataContainer. If a library searcher finds multiple copies of a shared library, it must decide which copy to return. It should consider which copy(ies) of the library are currently loaded by running teams in its decision. It also should consider the requested compatibility number of the shared library. If the searcher can access the compatibility number a shared library (the compatibility number is a property of shared libraries that are files), it should check that the compatibility number of the shared library it is considering returning is greater than or equal to the desired compatibility number. Return NIL, if the library is not found. This method must be overridden by all TLibrarySearcher subclasses.

Boolean IsEqual(constMCollectible*): Return TRUE if the two library searchers are equivalent; return FALSE if otherwise.

The following methods may be called ONLY by the workspace:
static AddSystemLibrarySearcher(const TLibrarySearcher &): Add a library searcher to the global list of library searchers that are automatically inherited by teams at the time they are created. This function only affects teams that are created after it has been called; it has no effect on any existing teams.

static TIterator*CreateSystemLibrarySearcherIterator(): Creates an iterator for the system library searchers. The iterator returns pointers to TLibrarySearcher subclass objects. The caller owns the storage for the iterator. The iterator owns the storage for the TLibrarySearchers it returns. The iterator returns the system library searchers in the order they would be applied. A kIteratorOutOfSync exception is raised if a system library searcher is added by another thread while the current thread is iterating the system library searchers.

//Subclass library searcher used by the run time during system boot.
//Not intended for subclassing.

TStandardLibrarySearcher(): Constructor.
~TStandardLibrarySearcher(): Destructor.
TMappableDataContainer*CreateContainerAndFindLibrary(constTText &libraryName, const CompatibilityNumberdesiredRelease): Search for the library with the specified name in the "Shared Libraries" folder on disk and return a TMappableDataContainer for that library; return NIL, if the library is not found. This library searcher does not consider the requested compatibility number in its decision. The caller owns the storage for the TMappableDataContainer object.

BooleanIsEqual(constMCollectible*): Return TRUE if the two standard library searchers are equivalent; return FALSE otherwise.

While the invention has been described in terms of a preferred embodiment in a specific system environment, those skilled in the art recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the spirit and scope of the appended claims.

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letter Patents is:

1. A method for locating shared libraries in a computer linked with a specific library, comprising the steps of:
   (a) ascertaining a plurality of shared library names linked with a specific library;
   (b) searching for and attaining a plurality of locations corresponding to the plurality of shared librray names; and
   (c) mapping the plurality of shared library names to the plurality of locations.

2. The method as recited in claim 1, wherein a plurality of directories are searched.

3. The method as recited in claim 2, wherein a plurality of directories are searched by employing a plurality of search objects.

4. The method as recited in claim 3, wherein one or more of the plurality of search objects are used in the event of default.

5. The method as recited in claim 3, wherein one or more of the plurality of search objects are placed within a team.

6. The method as recited in claim 5, wherein the search objects placed within the team are consecutively stored in a sequential order.

7. The method as recited in claim 6, wherein the sequential order is based on when each of the search objects is added to the team.

8. The method as recited within claim 1, wherein only one of the plurality of locations is selectively attained for each of the plurality of shared library names.

9. The method as recited within claim 8, wherein the one location attained is selected based on a degree of compatibility.

10. The method as recited within claim 1, wherein the plurality of shared library names linked with the specific library are ascertained by referring to a defined location.

11. A method for locating shared libraries in a computer linked with a specific library, comprising the steps of:
    (a) forming a plurality of search objects;
    (b) streaming the plurality of search objects into a designated location;
    (c) identifying a plurality of shared library names linked with a specific library;
    (d) consecutively employing the plurality of search objects streamed into the designated location to attain a plurality of locations corresponding to the plurality of shared library names; and
    (e) mapping the plurality of shared library names to the plurality of attained locations.

12. The method as recited within claim 11, including the step of streaming the plurality of attained locations into the designated location.

13. The method as recited within claim 11, wherein one or more of the plurality of search objects summon a procedure existing in a specified location.

14. The method as recited within claim 13, wherein the procedure can be changed.

15. The method as recited within claim 13, including the step of precluding one or more of the plurality of search objects from being streamed into the designated location when a duplicate of the one or more of the plurality of search objects already exists in the designated location.

16. The method as recited within claim 11, wherein one or more of the plurality of search objects is streamed into a plurality of designated locations.

17. The method as recited within claim 11, wherein the plurality of search objects are consecutively employed based on the order in which the plurality of search objects were streamed into the designated location.

18. The method as recited within claim 17, wherein a search object streamed at an earlier time is employed after a search object streamed at a later time.

19. A method for loading shared libraries in a computer linked with a specific library, comprising the steps of:
    (a) launching a program requiring a specific library;
    (b) identifying a plurality of shared library names linked with the specific library;
    (c) searching for and attaining a plurality of locations corresponding to the plurality of ascertained names;
    (d) mapping the plurality of shared library names to the plurality of locations; and
    (e) loading the plurality of locations.

20. An apparatus for locating shared libraries linked with a specific library, comprising:
    (a) means for ascertaining a plurality of shared library names linked with a specific library;
    (b) means for searching for and attaining a plurality of locations corresponding to the plurality of shared library names; and
    (c) means for mapping the plurality of shared library names to the plurality of locations.

21. The apparatus as recited in claim 20, wherein a plurality of directories are searched.

22. The apparatus as recited in claim 21, wherein a plurality of directories are searched by employing a plurality of search objects.

23. The apparatus as recited in claim 22, wherein one or more of the plurality of search objects are used in the event of default.

24. The apparatus as recited in claim 22, wherein one or more of the plurality of search objects are placed within a team.

25. The apparatus as recited in claim 24, wherein the search objects placed within the team are consecutively employed in a sequential order.

26. The apparatus as recited in claim 25, wherein the sequential order is based on when each of the search objects is added to the team.

27. The apparatus as recited within claim 20, wherein only one of the plurality of locations is selectively attained for each of the plurality of shared library names.

28. The apparatus as recited within claim 27, wherein the one location attained is selected based on a degree of compatability.

29. The apparatus as recited within claim 20, wherein the plurality of shared library names linked with the specific library are ascertained by referring to a defined location.

30. An apparatus for locating shared libraries linked with a specific library, comprising:
    (a) means for forming a plurality of search objects;
    (b) means for streaming the plurality of search objects into a designated location;
    (c) means for ascertaining a plurality of shared library names linked with a specific library;
    (d) means for consecutively employing the plurality of search objects streamed into the designated location to attain a plurality of locations corresponding to the plurality of shared library names; and
    (e) means for mapping the plurality of shared library names to the plurality of attained locations.

31. The apparatus as recited within claim 30, including means for streaming the plurality of attained locations into the designated location.

32. The apparatus as recited within claim 30, wherein one or more of the plurality of search objects summon a procedure existing in a specified location.

33. The apparatus as recited within claim 32, wherein the procedure can be changed.

34. The apparatus as recited within claim 33, including a processor for precluding one or more of the plurality of search objects from being streamed into the designated location when a duplicate of the one or more of the plurality of search objects already exists in the designated location.

35. The apparatus as recited within claim 30, wherein one or more of the plurality of search objects is streamed into a plurality of designated locations.

36. The apparatus as recited within claim 30, wherein the plurality of search objects are consecutively employed based on the order in which the plurality of search objects were streamed into the designated location.

37. The apparatus as recited within claim 36, wherein a search object streamed at an earlier time is employed after a search object streamed at a later time.

38. An apparatus for loading shared libraries linked with a specific library, comprising:
    (a) a processor for launching a program requiring a specific library;
    (b) a processor for ascertaining a plurality of shared library names linked with the specific library;
    (c) a processor for searching for and attaining a plurality of locations corresponding to the plurality of ascertained names;
    (d) a processor for mapping the plurality of shared library names to the plurality of locations; and
    (e) a processor for loading the plurality of locations.

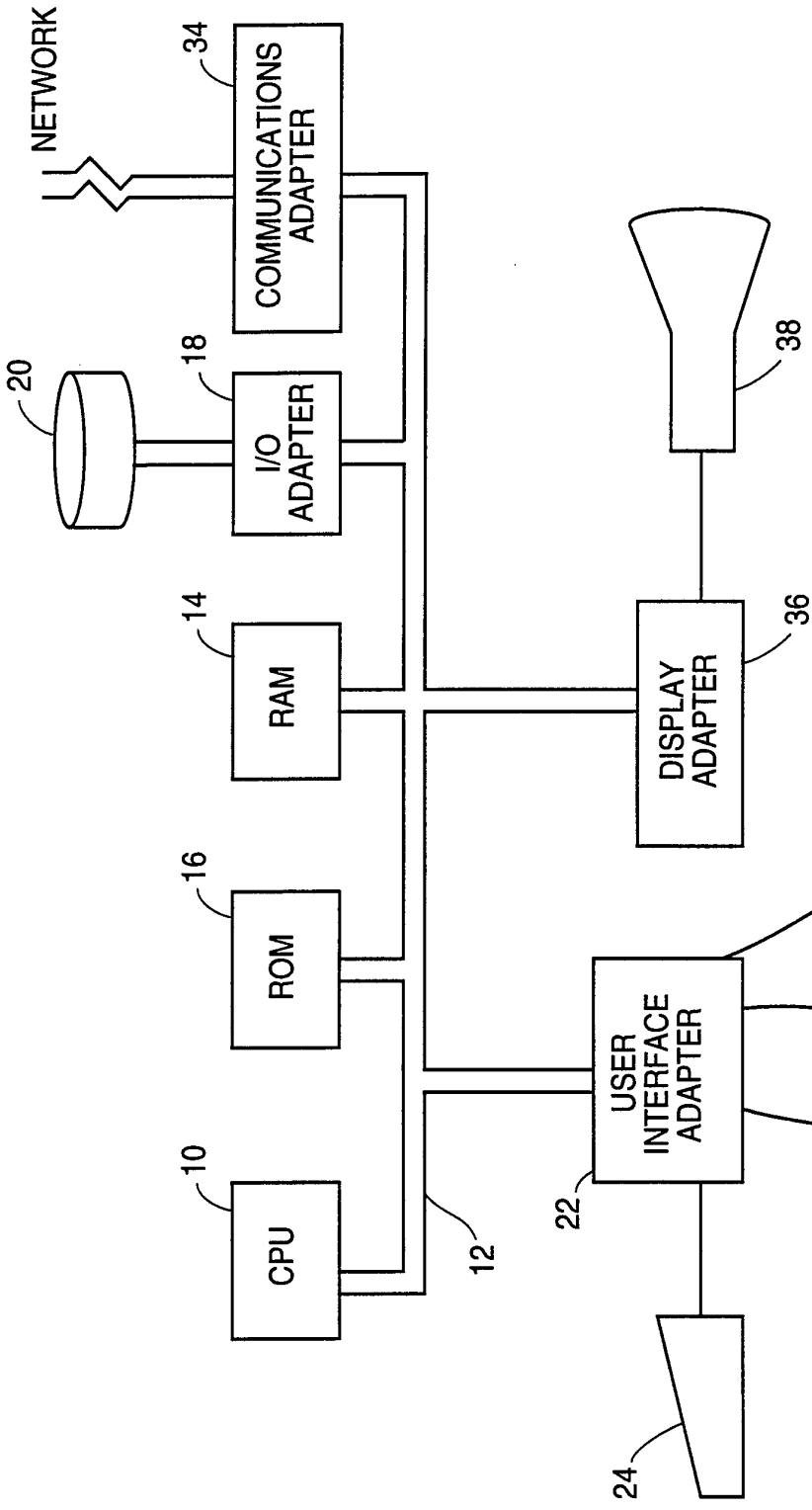

NETWORK
34
COMMUNICATIONS ADAPTER
18
I/O ADAPTER
20
38
14
RAM
36
DISPLAY ADAPTER
16
ROM
28
USER INTERFACE ADAPTER
26
10
CPU
12
22
32
24

FIG. 1

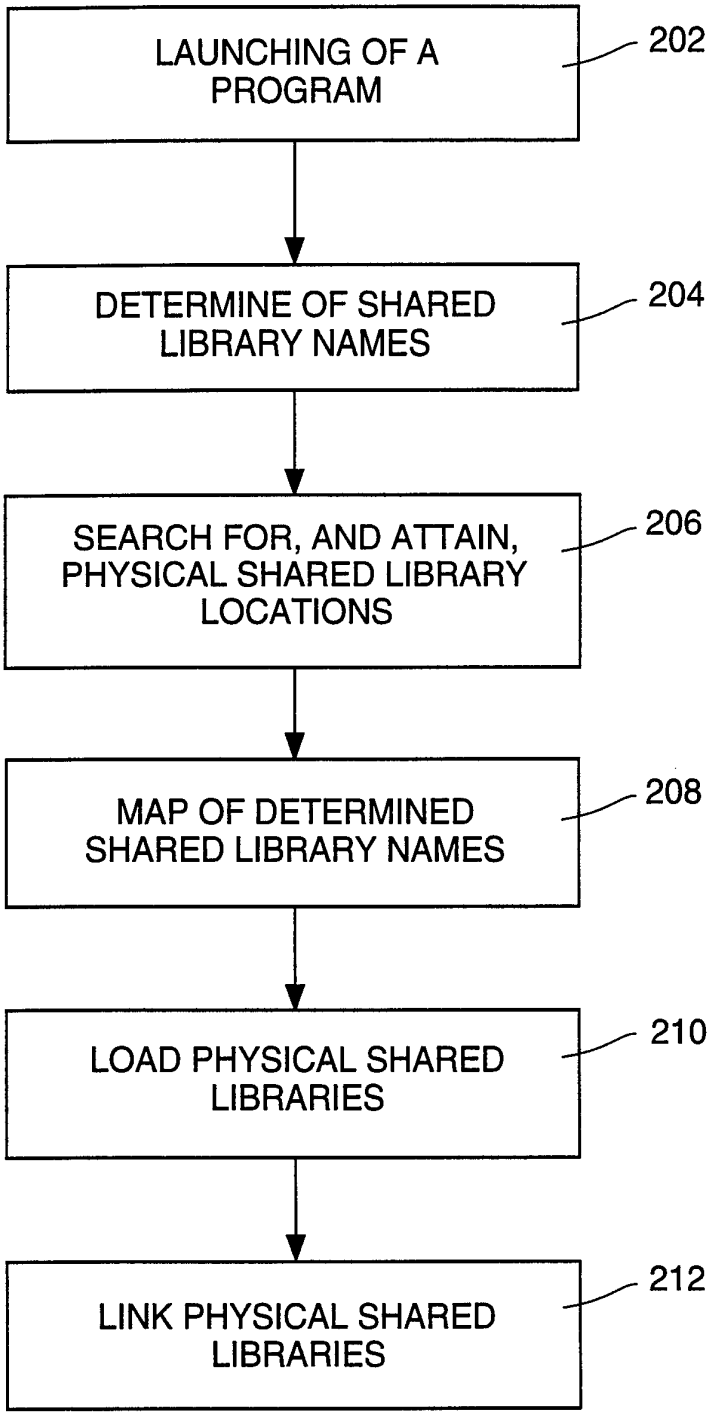

LAUNCHING OF A PROGRAM
202
DETERMINE OF SHARED LIBRARY NAMES
204
SEARCH FOR, AND ATTAIN, PHYSICAL SHARED LIBRARY LOCATIONS
206
MAP OF DETERMINED SHARED LIBRARY NAMES
208
LOAD PHYSICAL SHARED LIBRARIES
210
LINK PHYSICAL SHARED LIBRARIES
212

FIG. 2

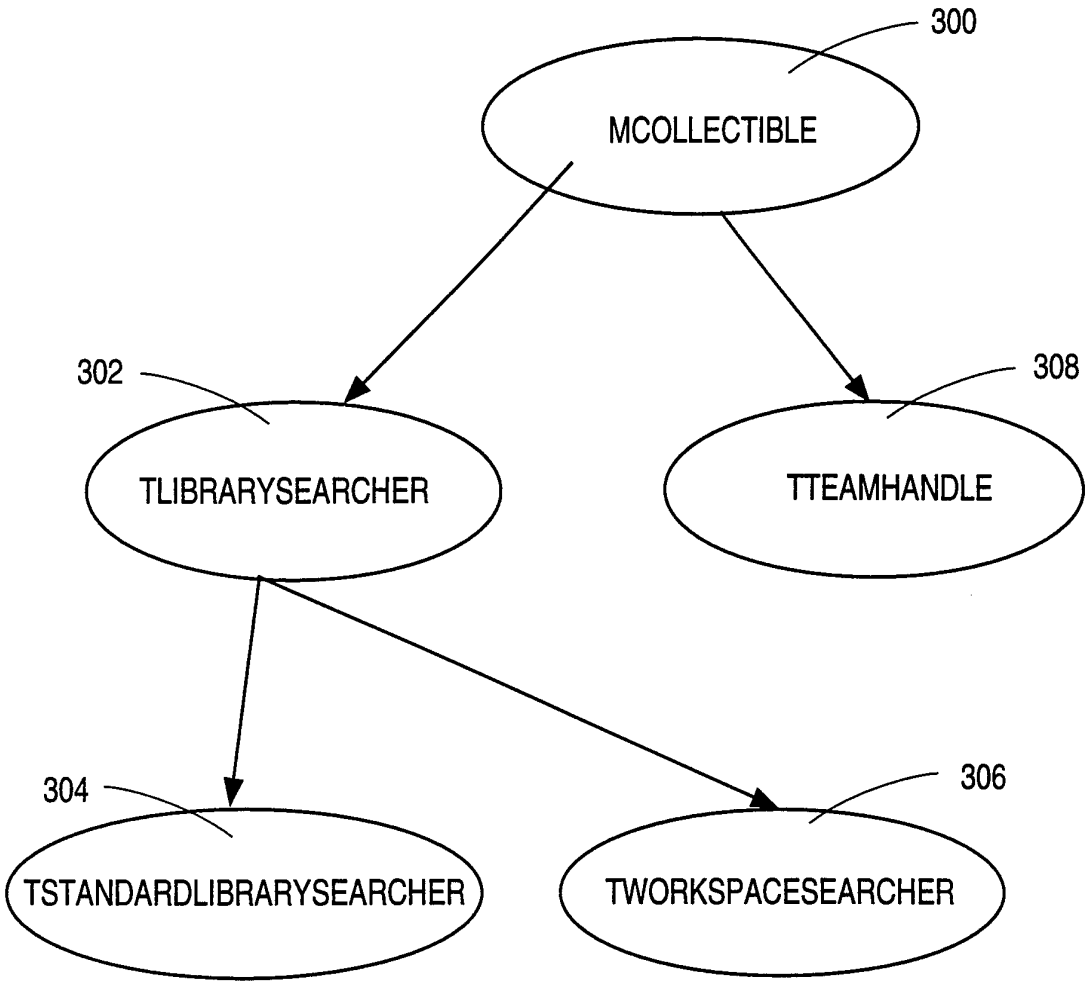

300
MCOLLECTIBLE
302
TLIBRARYSEARCHER
308
TTEAMHANDLE
304
TSTANDARDLIBRARYSEARCHER
306
TWORKSPACESEARCHER

FIG. 3

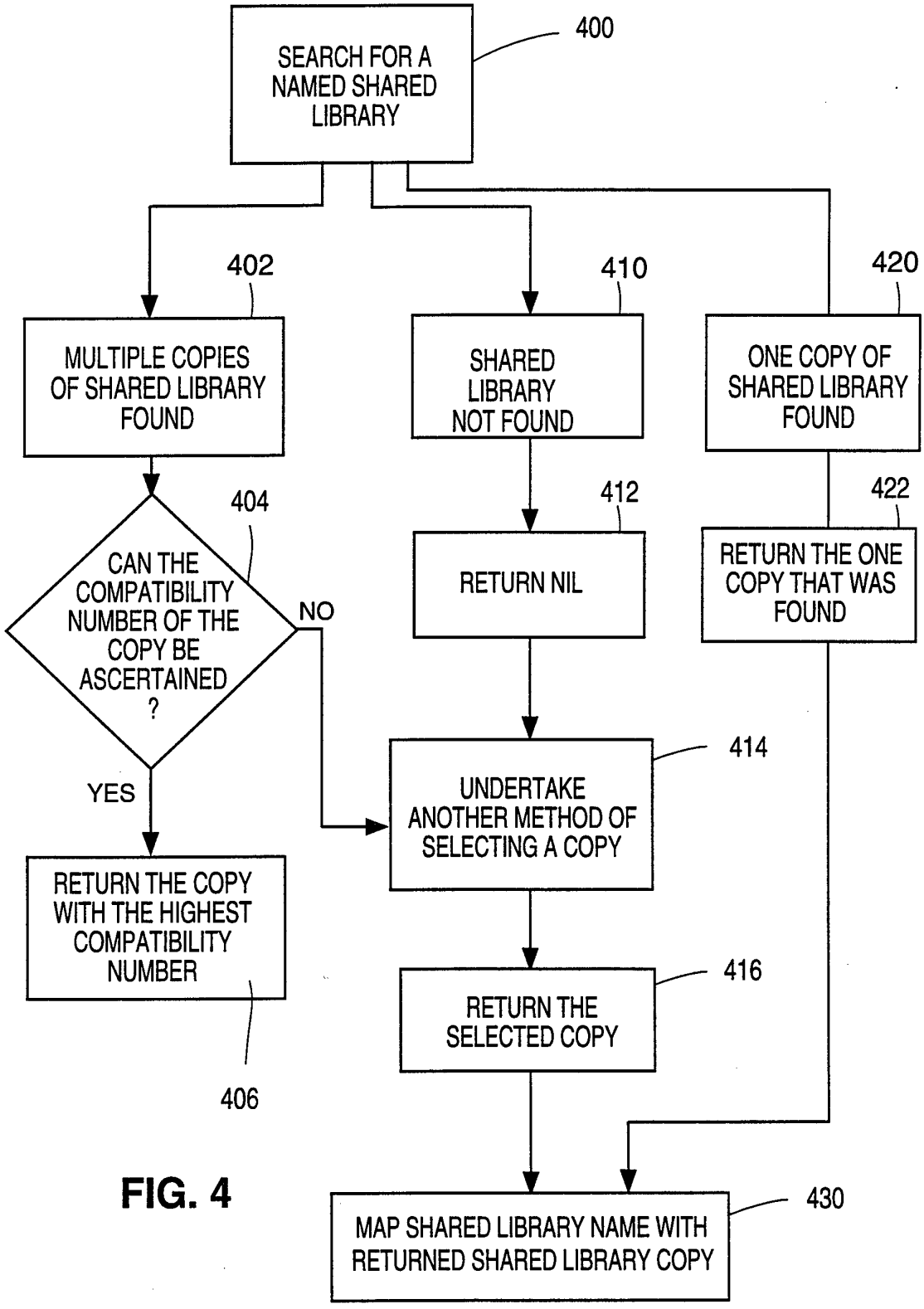

SEARCH FOR A NAMED SHARED LIBRARY
400
402
MULTIPLE COPIES OF SHARED LIBRARY FOUND
404
CAN THE COMPATIBILITY NUMBER OF THE COPY BE ASCERTAINED ?
NO
YES
RETURN THE COPY WITH THE HIGHEST COMPATIBILITY NUMBER
406
410
SHARED LIBRARY NOT FOUND
412
RETURN NIL
420
ONE COPY OF SHARED LIBRARY FOUND
422
RETURN THE ONE COPY THAT WAS FOUND
414
UNDERTAKE ANOTHER METHOD OF SELECTING A COPY
416
RETURN THE SELECTED COPY
MAP SHARED LIBRARY NAME WITH RETURNED SHARED LIBRARY COPY
430

FIG. 4

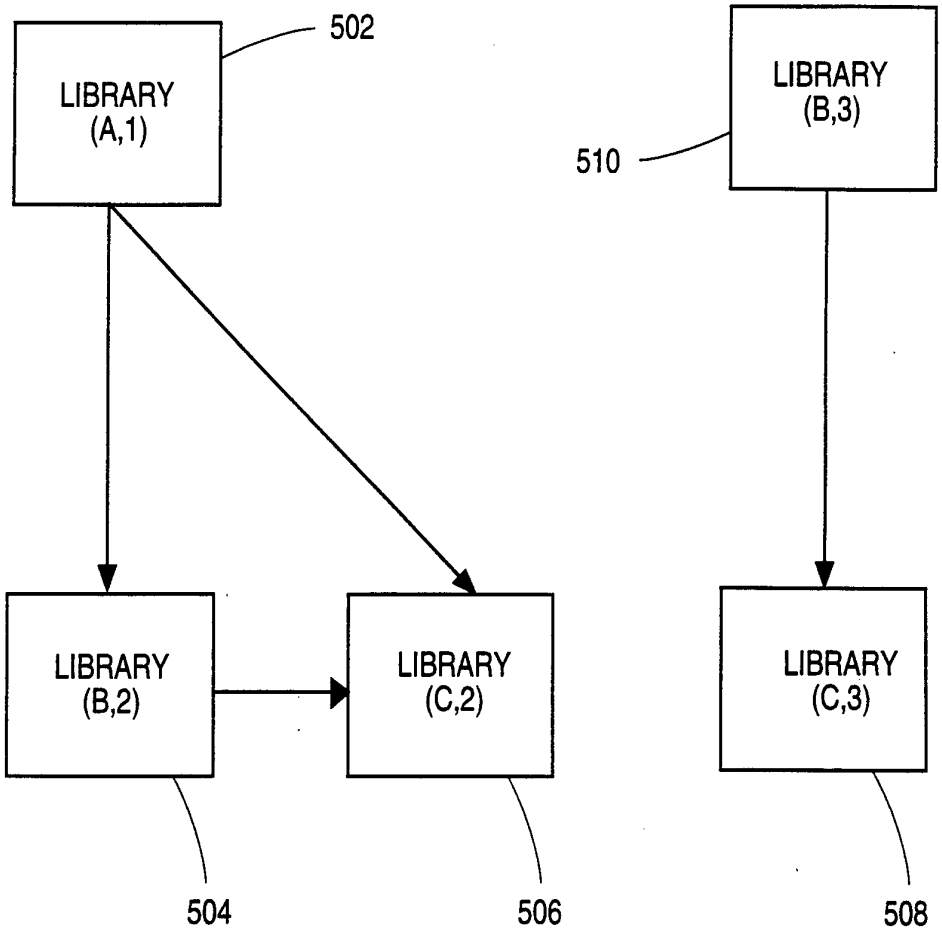

LIBRARY
(A,1)
502
LIBRARY
(B,3)
510
LIBRARY
(B,2)
LIBRARY
(C,2)
LIBRARY
(C,3)
504
506
508

FIG. 5

INTERNATIONAL SEARCH REPORT

Intern. ial Application No
PCT/US 94/00053

A. CLASSIFICATION OF SUBJECT MATTER
IPC 5 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 5 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	PROGRAMMER'S JOURNAL vol. 7, no. 2 , April 1989 pages 40 - 47 R. DUNCAN 'OS/2 Dynamic Link Libraries' see the whole document ---	1,2, 8-10, 19-21, 27-29,38 3-7, 11-18, 22-26, 30-37
Y	EP,A,0 352 447 (I.B.M. CORPORATION) 31 January 1990 see claim 1 -----	3-7, 11-18, 22-26, 30-37

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
21 April 1994	02.05.94

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Katerbau, R

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

...formation on patent family members

Inter ...al Application No
PCT/US 94/00053

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0352447	31-01-90	US-A-	5134696	28-07-92
		JP-A-	2079141	19-03-90
		US-A-	5247647	21-09-93